

Heuristic Search

Heuristic search prefers to visit states that appear to be better.

A search* visits states based on cost from initial to a given state plus *heuristic function*.

A *heuristic function* estimates the cost from a given state to the closest goal state.

```

function A*(initial, EXPAND, GOAL,
              COST, HEURISTIC)
  q ← NEW-PRIORITY-QUEUE()
  INSERT(initial, q, HEURISTIC(initial))
  while q is not empty
    do current ← EXTRACT-MIN(q)
      if GOAL(current) then return solution
      for each next in EXPAND(current)
        do INSERT(next, q, COST(next)
                  + HEURISTIC(next))
  return failure

```

IDA*: Iterative Deepening A* Search

function IDA*(*initial*, EXPAND, GOAL,
 COST, HEURISTIC)

limit \leftarrow HEURISTIC(*initial*)

loop

do *result*, *limit* \leftarrow CONTOUR (*initial*, *limit*)

if *result* **then return** *result*

if *limit* = ∞ **then return** failure

function CONTOUR(*current*, *limit*)

cost \leftarrow COST(*current*) + HEURISTIC(*current*)

if *limit* < *cost* **then return** null, *cost*

if GOAL(*current*) **then return** solution, *cost*

new-limit \leftarrow ∞

for each *next* in EXPAND(*current*)

do *result*, *cost* \leftarrow CONTOUR(*next*, *limit*)

if *result* **then return** solution, *cost*

new-limit \leftarrow min(*new-limit*, *cost*)

return failure, *new-limit*

Properties of A* Search

Let n be a state/node.

Let $g(n)$ be the cost from the initial state to n .

Let $h(n)$ be the estimate from n to a goal state.

Let $f(n) = g(n) + h(n)$.

h is *admissible* if it is never an overestimate.

If h is admissible, then A finds optimal path.*

Proof Sketch: Let f^* be optimal path cost.

Because h never overestimates, then all states n on optimal path have $f(n) \leq f^*$.

Because of priority queue, A* will visit optimal path before any suboptimal goal state.

Assume *tree-structured* state space ($b =$ branching factor, $d =$ goal depth), single goal state, each edge costs 1, and maximum error of ϵ .

A and IDA* visit $O(db^{\epsilon/2})$ states.*

A uses $O(db^{\epsilon/2})$ memory. IDA* uses $O(db)$.*

Performance of Heuristic Functions

Consider these 8-puzzle heuristic functions:

h_1 : number of tiles in goal position.

h_2 : Manhattan distance from tiles to goals.

Both never overestimate and $h_1 \leq h_2$

Characterize by *effective branching factor*.

If N states visited and solution depth d ,

then solve for $N = \sum_{i=0}^d x^i$

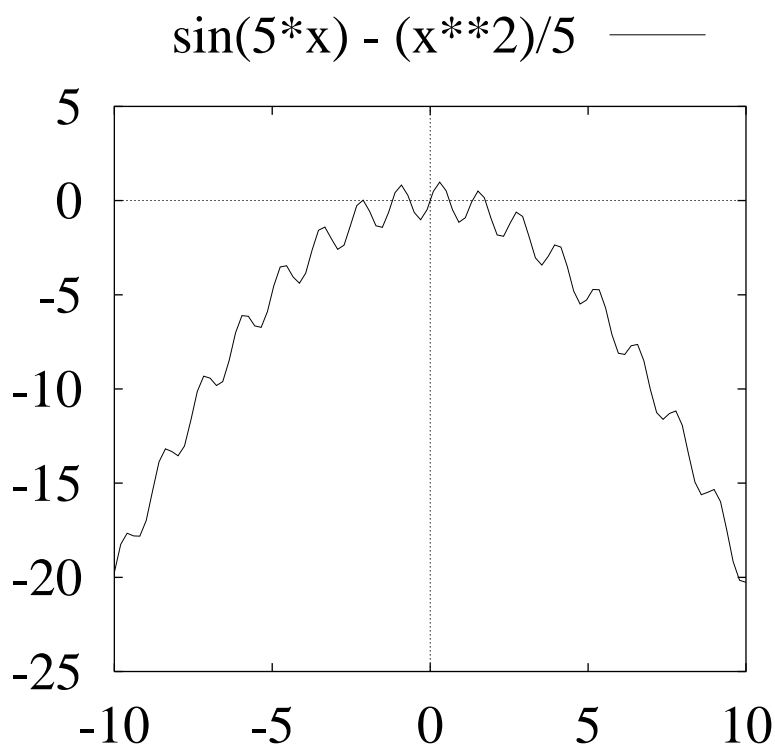
Book Experiment Avoiding Bidirectional Edges

	States Visited (Effective BF)		
d	IDS	IDA*(h_1)	IDA*(h_2)
4	52 (2.35)	10 (1.35)	7 (1.17)
8	569 (2.03)	42 (1.36)	14 (1.11)
12	5357 (1.92)	315 (1.47)	45 (1.19)
16	47271 (1.87)	2410 (1.52)	226 (1.28)
20		17646 (1.55)	764 (1.29)

Local Search

A *local search* algorithm keeps track of one state at a time, using an evaluation function and a selection procedure to decide what state to visit next.

Local search gives up optimality guarantees in hopes of finding good solutions more efficiently. The main difficulty is local minima/maxima.



Local Search Algorithms

```
function LOCAL-SEARCH(initial, EXPAND,  
                    GOAL, SELECT)  
    current ← initial  
    loop  
        do if GOAL(current) then return solution  
        current ← SELECT(EXPAND(current))
```

Hill-Climbing, Gradient Descent:

Select state improving an evaluation function.

Random-restart hill-climbing:

Repeat hill climbing from random initial states.

Simulated Annealing:

Hill-climbing with randomized selection.

Genetic Algorithms:

Maintain a set of “current states.” Crossover generates new states from pairs of states.

Tabu Search: Like hill-climbing, but avoid recently visited states or recently used operators.