

CSC 550: Introduction to Artificial Intelligence

Fall 2004

heuristics & informed search

- heuristics
- hill-climbing
 - bold + informed search
 - potential dangers, variants
- best first search
 - tentative + informed search
 - best-first search vs. DFS vs. BFS

1

Search strategies so far...

bold & uninformed: STUPID

tentative & uninformed: DFS (and variants), BFS

bold & informed: *hill-climbing*

- essentially, DFS utilizing a "measure of closeness" to the goal (a.k.a. a *heuristic function*)
- from a given state, pick the *best* successor state
 - i.e., the state with highest heuristic value
- stop if no successor is better than the current state
 - i.e., no backtracking so only need to store current path

EXAMPLE: travel problem, e.g. Omaha → LosAngeles
heuristic(State) = -(crow-flies distance from goal)

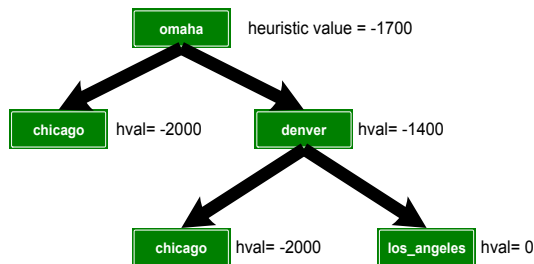
2

Hill-climbing example

```
(define MOVES
  '( (Omaha --> Chicago)
      (Omaha --> Denver)
      (Chicago --> Denver)
      (Chicago --> LosAngeles)
      (Chicago --> Omaha)
      (Denver --> LosAngeles)
      (Denver --> Omaha)
      (LosAngeles --> Chicago)
      (LosAngeles --> Denver)))
```

```
(define DISTANCES
  '((Omaha (Omaha 0) (Chicago 500) (Denver 400)
          (LosAngeles 1700))
    (Chicago (Chicago 0) (Omaha 500) (Denver 700)
            (LosAngeles 2000))
    (Denver (Denver 0) (Omaha 400) (Chicago 700)
           (LosAngeles 1400))
    (LosAngeles (LosAngeles 0) (Omaha 1700)
                (Chicago 2000) (Denver 1400))))

(define (HEURISTIC state goalState)
  (- 0 (cadr (assoc goalState (cdr (assoc state DISTANCES))))))
```



the heuristic guides the search, always moving closer to the goal

what if the flight from Denver to L.A. were cancelled?

3

Hill-climbing implementation

```
(require (lib "compat.ss"))

(define (hill-climb startState goalState)

  (define (climb-path path)
    (if (equal? (car path) goalState)
        path
        (let ((nextStates (sort better-state (GET-MOVES (car path)))))
          (if (or (null? nextStates) (not (better-state (car nextStates) (car path))))
              #f
              (climb-path (cons (car nextStates) path))))))

  (define (better-state state1 state2)
    (> (HEURISTIC state1 goalState) (HEURISTIC state2 goalState)))

  (climb-path (list startState)))
```

can utilize the `sort` function from the `compat.ss` library

- requires a comparison function (evals to #t if input1 should come before input2)

`climb-path` takes the current path:

- if the current state (car of path) is the goal, then SUCCEED
- otherwise, get all possible moves and sort based on heuristic value (high to low)
- if no moves or "best" move is no better than current state, then FAIL
- otherwise, add "best" move to path and recurse

4

Hill-climbing in practice

```
> (hill-climb 'Omaha 'LosAngeles)
(losangeles denver omaha)
```

given these flights, actually finds optimal paths

```
;;; REMOVED Denver to L.A. FLIGHT
> (hill-climb 'Omaha 'LosAngeles)
#E
```

WHY?

```
;;; ADDED Omaha ↔ KC, KC ↔ LA FLIGHTS
;;; ASSUME Omaha is 200 mi from KC, KC is 1900 mi from LA
> (hill-climb 'Omaha 'LosAngeles)
(losangeles denver omaha)
> (hill-climb 'LosAngeles 'Omaha)
(losangeles kansascity omaha)
```

get different answers

WHY?

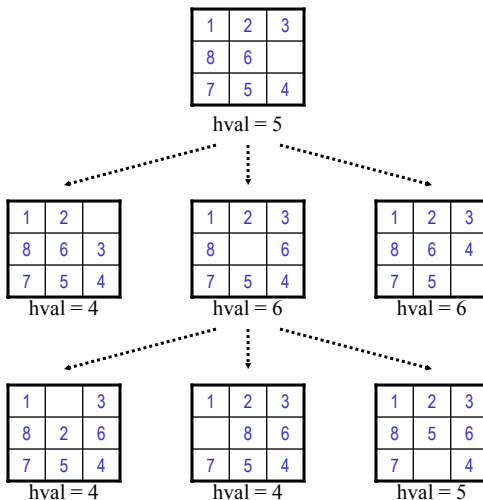
hill-climbing only works if

- the heuristic is accurate (i.e., distinguishes "closer" states)
- the path to the goal is direct (i.e., state improves on every move)

5

8-puzzle example

heuristic(State) = # of tiles in place, including the space



start state has 5 tiles in place

of the 3 possible moves, 2 improve the situation

if assume left-to-right preference, move leads to a dead-end (no successor state improves the situation) so STOP!

6

Intuition behind hill-climbing

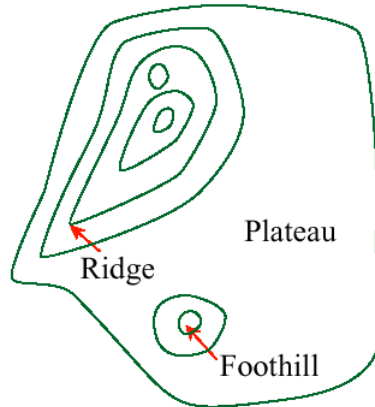
if you think of the state space as a topographical map (heuristic value = elevation), then hill-climbing selects the steepest gradient to move up towards the goal

potential dangers

plateau: successor states have same values, no way to choose

foothill: local maximum, can get stuck on minor peak

ridge: foothill where N-step lookahead might help



7

Hill-climbing variants

could generalize hill-climbing to continue even if the successor states look worse

- always choose best successor
- don't stop unless reach the goal or no successors

dead-ends are still possible (and likely if the heuristic is not perfect)

simulated annealing

- allow moves in the wrong direction on a probabilistic basis
- decrease the probability of a backward move as the search continues

idea: early in the search, when far from the goal, heuristic may not be good
heuristic should improve as you get closer to the goal

approach is based on a metallurgical technique

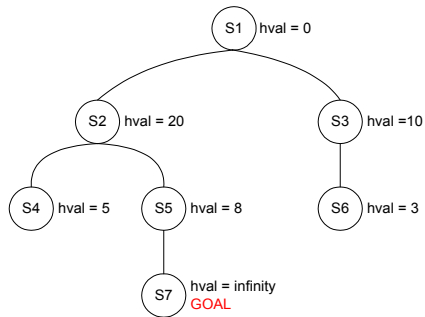
8

Best first search

since bold, hill-climbing is dependent on a near-perfect heuristic

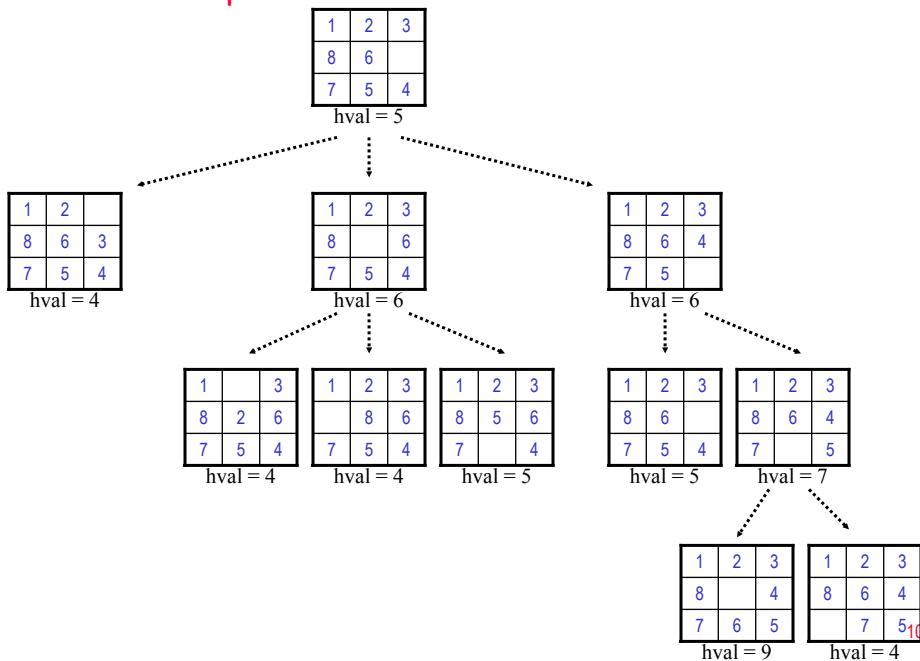
tentative & informed: best first search

- like breadth first search, keep track of all paths searched
- like hill-climbing, use heuristics to guide the search
- always expand the "most promising" path
i.e., the path ending in a state with highest heuristic value



9

Best first example



Breadth first search vs. best first search

procedural description of breadth first search

- must keep track of all current paths, initially: `((startState))`
- look at first path in the list of current paths
if the path ends in a goal, then DONE
otherwise,
 - remove the first path
 - generate all paths by extending to each possible move
 - add the newly extended paths to the **end of the list**
 - recurse

procedural description of best first search

- must keep track of all current paths, initially: `((startState))`
- look at first path in the list of current paths
if the path ends in a goal, then DONE
otherwise,
 - remove the first path
 - generate all paths by extending to each possible move
 - add the newly extended paths to the list, **sorted by (decreasing) HEURISTIC value**
 - recurse

11

Best first implementation

```
(require (lib "compat.ss"))

(define (best startState goalState)

  (define (best-paths paths)
    (cond ((null? paths) #f)
          ((equal? (caar paths) goalState) (car paths))
          (else (best-paths (sort better-path
                                   (append (cdr paths)
                                           (extend-all (car paths)
                                                         (GET-MOVES (caar paths))))))))))

  (define (extend-all path nextStates)
    (cond ((null? nextStates) '())
          ((member (car nextStates) path) (extend-all path (cdr nextStates)))
          (else (cons (cons (car nextStates) path)
                      (extend-all path (cdr nextStates))))))

  (define (better-path path1 path2)
    (> (HEURISTIC (car path1) goalState) (HEURISTIC (car path2) goalState)))

  (best-paths (list (list startState))))
```

only difference from BFS implementation:

- newly extended path list is sorted by heuristic value of end states (high-to-low)
- since new paths are added at end & sort is stable, ties will favor older/shorter paths

12

Travel example

```
> (best 'Omaha 'LosAngeles)
(losangeles denver omaha)

> (best 'LosAngeles 'Omaha)
(omaha denver losangeles )
```

given these flights, actually finds optimal paths

```
;;; REMOVED Denver to L.A. FLIGHT

> (best 'Omaha 'LosAngeles)
(losangeles chicago omaha)
```

WHY?

```
;;; ADDED Omaha ↔ KC, KC ↔ LA FLIGHTS
;;; ASSUME Omaha is 200 mi from KC, KC is 1900 mi from LA

> (best 'Omaha 'LosAngeles)
(losangeles denver omaha)

> (best 'LosAngeles 'Omaha)
(losangeles kansascity omaha)
```

still different answers

WHY?

note: best first search does not guarantee an "optimal" solution

- does not take the past into account, simply tries move that gets closest
- would select LA → KC → Omaha → DesMoines even if
LA → Chicago → DesMoines were possible

13

8-puzzle example

```
(define (HEURISTIC state goalState)
  (cond ((null? state) 0)
        ((equal? (car state) (car goalState))
         (+ 1 (HEURISTIC (cdr state) (cdr goalState))))
        (else (HEURISTIC (cdr state) (cdr goalState)))))
```

heuristic counts number of tile matches with goal

```
> (best '(1 2 3 8 6 space 7 5 4) '(1 2 3 8 space 4 7 6 5))
((1 2 3 8 space 4 7 6 5)
 (1 2 3 8 6 4 7 space 5)
 (1 2 3 8 6 4 7 5 space)
 (1 2 3 8 6 space 7 5 4))
```

- DFS-nocycles required > 400 moves
- BFS found same soln, but 6 times slower

```
> (best '(2 8 3 1 space 6 7 5 4) '(1 2 3 8 space 4 7 6 5))
((1 2 3 8 space 4 7 6 5)
 (1 2 3 8 6 4 7 space 5)
 (1 2 3 8 6 4 7 5 space)
 (1 2 3 8 6 space 7 5 4)
 (1 2 3 8 space 6 7 5 4)
 (1 2 3 space 8 6 7 5 4)
 (space 2 3 1 8 6 7 5 4)
 (2 space 3 1 8 6 7 5 4)
 (2 8 3 1 space 6 7 5 4))
```

- DFS-nocycles hangs
- BFS found same soln, but > 200 times slower

14

Other examples

heuristic function for the Water Jug Problem?

heuristic function for Missionaries & Cannibals?

15

Comparing best first search

unlike hill-climbing, best first search can handle "dips" in the search

→ not so dependent on a perfect heuristic

depth first search and breadth first search may be seen as special cases of best first search

DFS: heuristic value is distance (number of moves) from start state

BFS: heuristic value is inverse of distance (number of moves) from start state

or, procedurally:

DFS: assign all states the same heuristic value
when adding new paths, add equal heuristic values at the *front*

BFS: assign all states the same heuristic value
when adding new paths, add equal heuristic values at the *end*

16

Next week...

MIDTERM EXAM (10/11)

- designed to be completed in 60 – 90 minutes, will allow full 120 minutes

types of questions:

- TRUE/FALSE
- short answer, discussion
- explain/modify/augment/write Scheme code

study advice:

- review online lecture notes
- review text
- reference other sources for examples, different perspectives
- look over quizzes