Complexity of A*

- Complexity is exponential unless

  $|h(n) - h(n^*)| \leq O(\log h^*(n))$

  where h*(n) is the true cost of going from n to goal.

- But, this is AI, computers are fast, and a good heuristic helps a lot.

Performance of Heuristics

- How do we evaluate a heuristic function?
- effective branching factor
  - If A* using h finds a solution at depth d using N nodes, then the effective branching factor is

$$b \mid N \cong 1 + b^2 + b^3 + \ldots + b^d$$

- Example

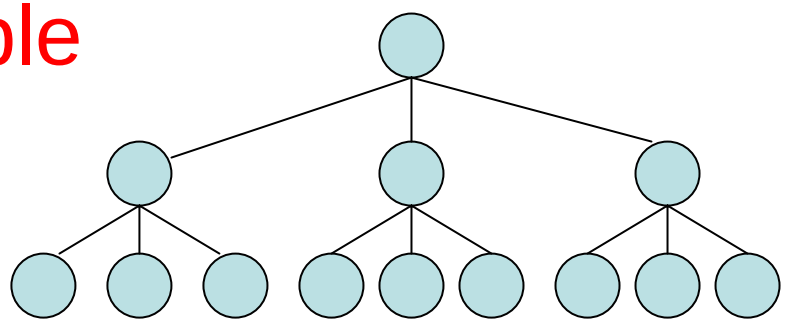


Table of Effective Branching Factors

b	d	N
2	2	7
2	5	63
3	2	13
3	5	364
3	10	88573
6	2	43
6	5	9331
6	10	72,559,411

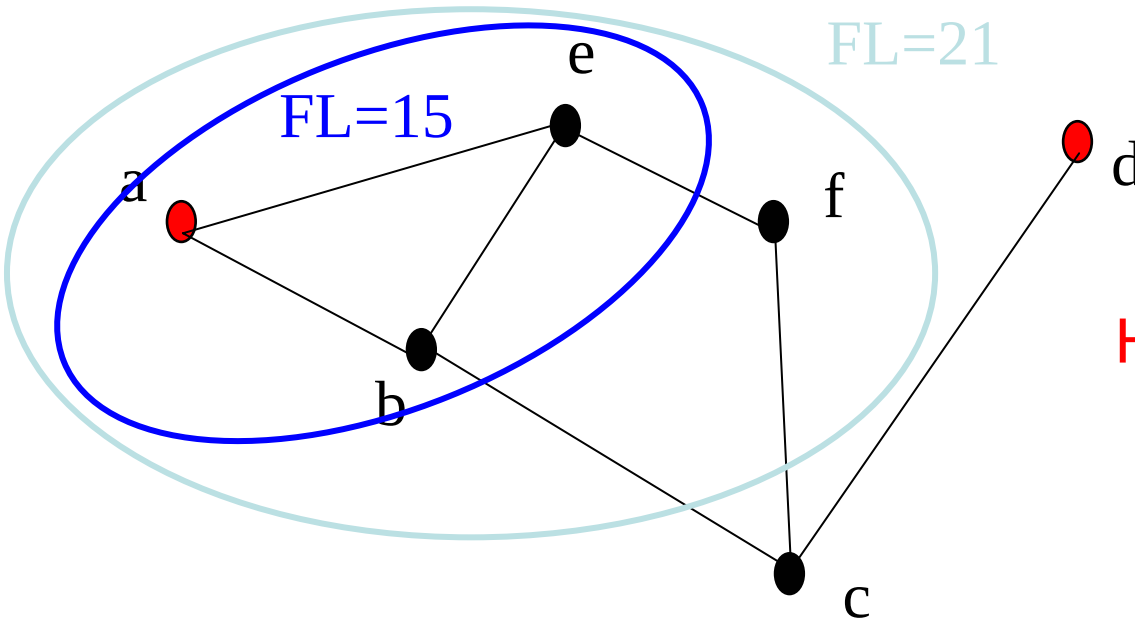
How might we use this idea to evaluate a heuristic?

Why not always use A*?

- Pros
- Cons

Iterative-Deepening A*

- Like iterative-deepening depth-first, but...
- Depth bound modified to be an **f-limit**
  - Start with limit = h(start)
  - Prune any node if f(node) > f-limit
  - Next f-limit=min-cost of any node pruned



How would this work?

Depth-First Branch & Bound

- Single DF search
  - → uses linear space
- Keep track of best solution so far
- If $f(n) = g(n)+h(n) \geq$ cost(best-soln)
  - Then prune n

- Requires
  - Finite search tree, or
  - Good upper bound on solution cost

Adapted from Richard Korf presentation

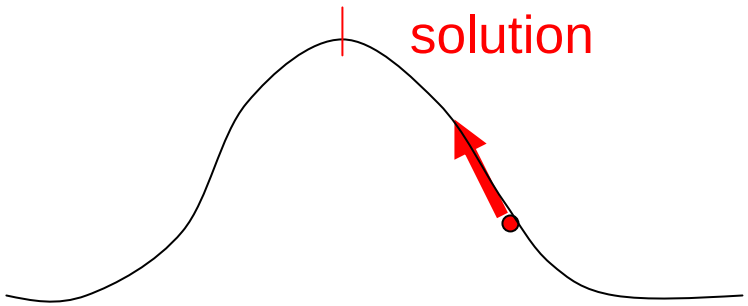
(Global) Beam Search

- Idea
  - Best first but only keep N best items on priority queue
- Evaluation
  - Complete?
  - Time Complexity?
  - Space Complexity?

Local Search Algorithms and Optimization Problems

- Complete state formulation
  - For example, for the 8 queens problem, all 8 queens are on the board and need to be moved around to get to a goal state
- Equivalent to optimization problems often found in science and engineering
- Start somewhere and try to get to the solution from there
- Local search around the current state to decide where to go next

Hill Climbing "Gradient ascent"



Note: solutions shown here as max not min.

Basic Hill Climbing

- current <- start state; if it's a goal return it.
- loop
  - select next operator and apply to current to get next
    - if next is a goal state, return it and quit
    - if not, but next is better than current, current<-next
- end loop

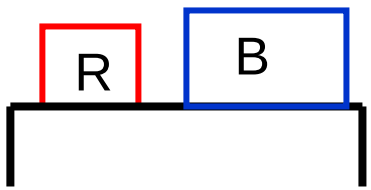
No queue!

Hill Climbing

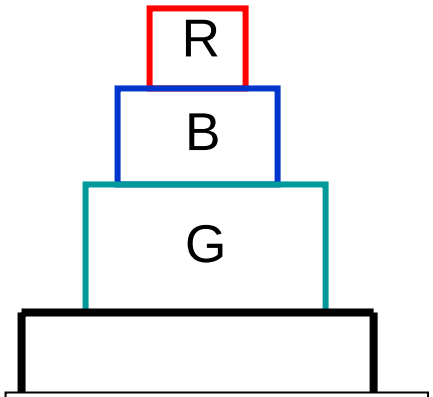
Steepest-Ascent Hill Climbing

- current <- start state; if it's a goal return it.
- loop
  - initialize best_successsor
  - for each operator
  - apply operator to current to get next
    - if next is a goal, return it and quit
    - if next is better than best_successor, best_successor <- next
  - if best-successor is better than current, current <- best_successor
- end loop

Robot Assembly Task



Initial State
- on(R,Table)
- on(B,Table)

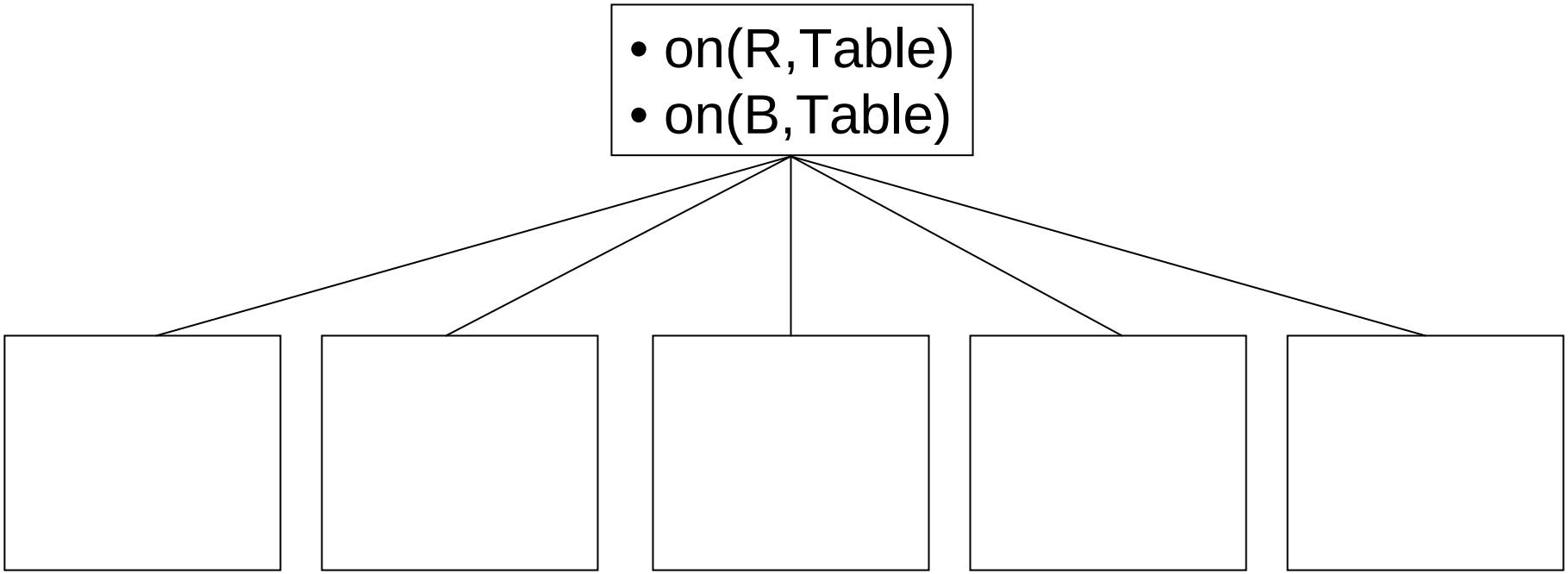


Goal State
- on(R,B)
- on(B,G)
- on(G,Table)

Moves?
Cost Function?
Heuristic Function?

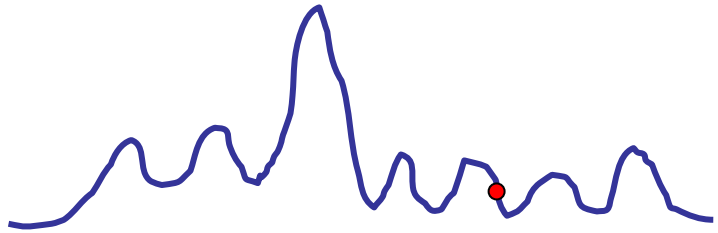
What moves can we make?
Which one is best?

Hill Climbing Search

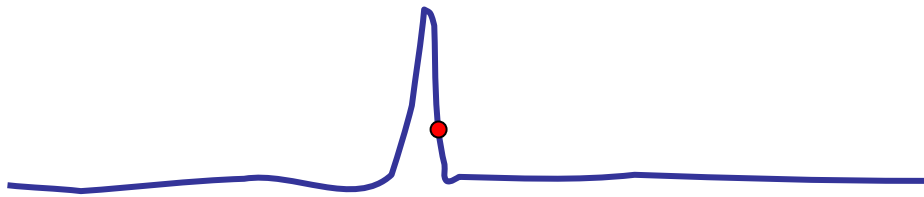


Hill Climbing Problems

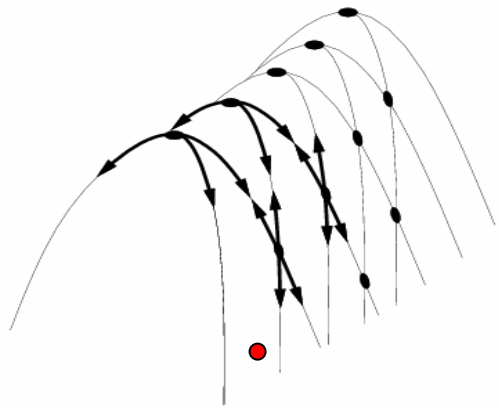
Local maxima



Plateaus



Diagonal ridges



Does it have any advantages?

Solving the Problems

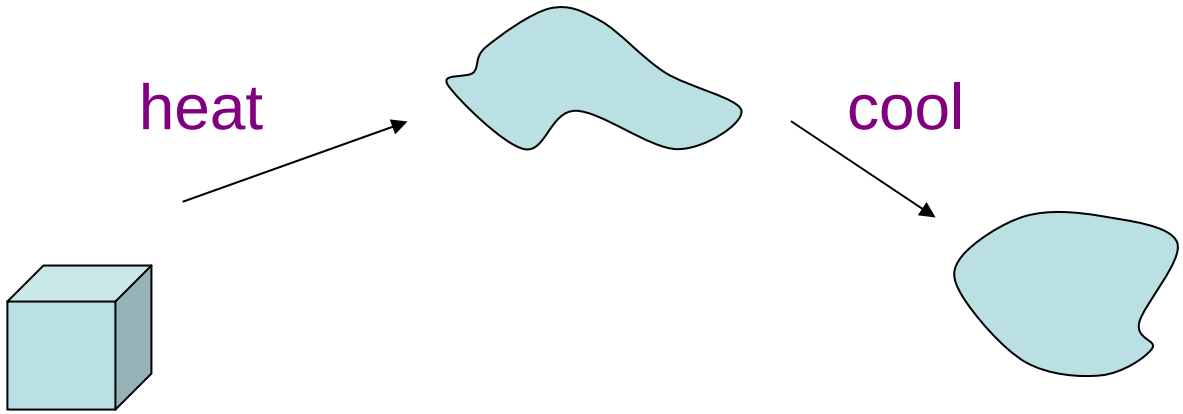
- Allow backtracking (What happens to complexity?)
- Stochastic hill climbing: choose at random from uphill moves, using steepness for a probability
- Random restarts: “If at first you don’t succeed, try, try again.”
- Several moves in each of several directions, then test
- Jump to a different part of the search space

Simulated Annealing

- Variant of hill climbing (so up is good)
- Tries to explore enough of the search space early on, so that the final solution is less sensitive to the start state
- May make some downhill moves before finding a good way to move uphill.

Simulated Annealing

- Comes from the physical process of annealing in which substances are raised to high energy levels (melted) and then cooled to solid state.



- The probability of moving to a higher energy state, instead of lower is $p = e^{(-\Delta E/kT)}$ where ΔE is the positive change in energy level, T is the temperature, and k is Bolzmann's constant.

Simulated Annealing

- At the beginning, the temperature is high.
- As the temperature becomes lower
  - kT becomes lower
  - $\Delta E/kT$ gets bigger
  - $(-\Delta E/kT)$ gets smaller
  - $e^{(-\Delta E/kT)}$ gets smaller
- As the process continues, the probability of a downhill move gets smaller and smaller.

For Simulated Annealing

- ΔE represents the change in the value of the objective function.
- Since the physical relationships no longer apply, drop k. So p = e^(-ΔE/T)
- We need an annealing schedule, which is a sequence of values of T: T_0, T_1, T_2, ...

Simulated Annealing Algorithm

- current <- start state; if it's a goal, return it
- for each T on the schedule /* need a schedule */
  - next <- randomly selected successor of current
  - evaluate next; it it's a goal, return it
  - ΔE <- value(next) – value(current) /* already negated */
  - if $\Delta E > 0$
    - then current <- next /* better than current */
    - else current <- next with probability $e^{(\Delta E/T)}$

How would you do this probabilistic selection?

Simulated Annealing Properties

- At a fixed “temperature” T, state occupation probability reaches the Boltzman distribution

$$p(x) = \alpha e^{(E(x)/kT)}$$

- If T is decreased slowly enough (very slowly), the procedure will reach the best state.

- Slowly enough has proven too slow for some researchers who have developed alternate schedules.

Local Beam Search

- Keeps more previous states in memory
  - Simulated annealing just kept one previous state in memory.
  - This search keeps k states in memory.

    - randomly generate k initial states
    - if any state is a goal, terminate
    - else, generate all successors and select best k
    - repeat

What does your book say is good about this?

Genetic Algorithms

- Start with random population of states
  - Representation serialized (ie. strings of characters or bits)
  - States are ranked with “fitness function”
- Produce new generation
  - Select random pair(s) using probability:
    - probability ~ fitness
  - Randomly choose “crossover point”
    - Offspring mix halves
  - Randomly mutate bits