Review

- Best-first search uses an evaluation function $f(n)$ to select the next node for expansion.

- Greedy best-first search uses $f(n) = h(n)$.

- Greedy best first search is not optimal, not complete, and has complexity $O(b^m)$.

Review

- A* search uses $f(n) = g(n) + h(n)$.
- A* search is complete.
- A* is optimal if h(n) is admissable

  $h(n) <= h^*(n)$ for true cost h*

  – with tree search
  – with graph search, if it discards the more expensive of any 2 paths to the same node or
  – if h(n) is consistent $h(n) <= c(n,a,n') + h(n')$.

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

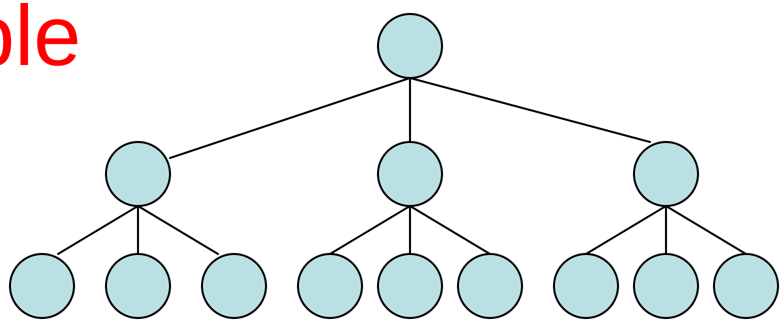


Table of Effective Branching Factors

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?

Complexity of A*

- Time complexity is exponential in the length of the solution path unless

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

  which we can't guarantee.

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

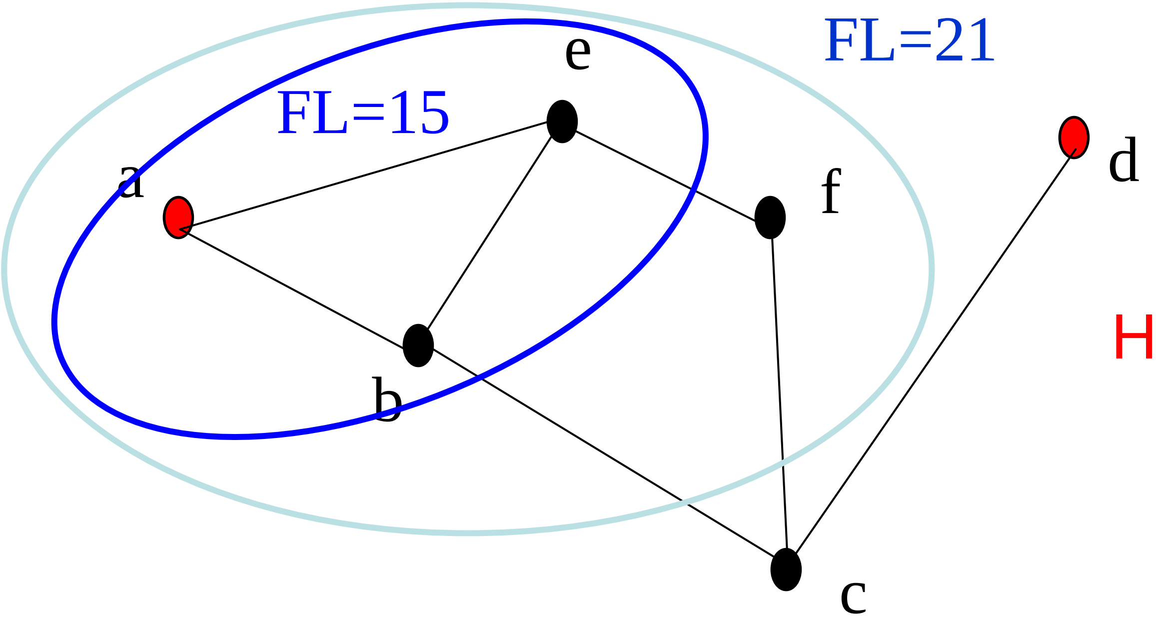
- Space complexity is also exponential, because it keeps all generated nodes in memory.

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

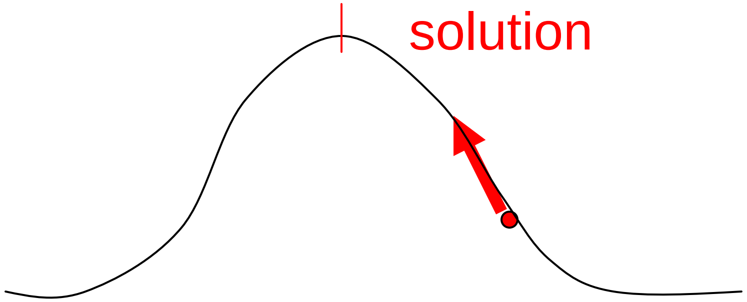


How would this work?

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.

Often used for numerical optimization problems.

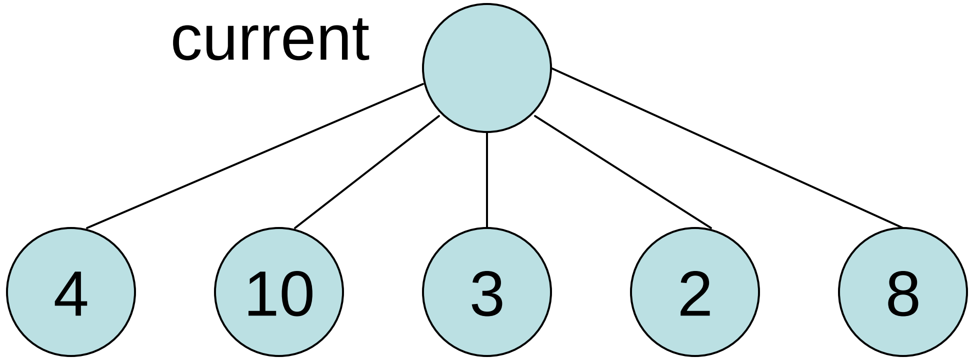
How does it work?

AI Hill Climbing

Steepest-Ascent Hill Climbing

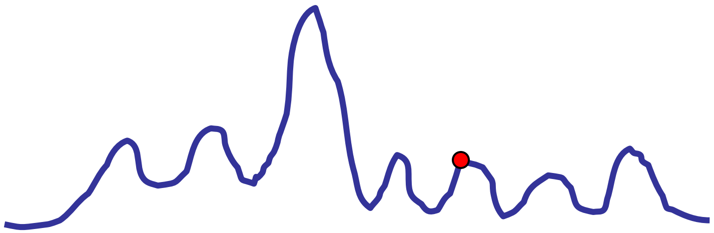
- current <- start state; if it’s a goal return it.
- loop
  - initialize best_successsor
  - for each operator
  - apply operator to current to get next
    - if next is a goal, return it and quit
    - if next is better than best_successor, best_successor <- next
  - if best-successor is better than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

Hill Climbing Search

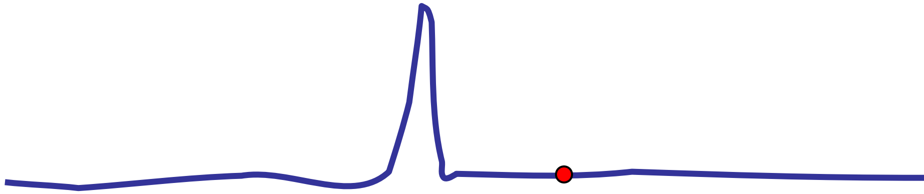


Hill Climbing Problems

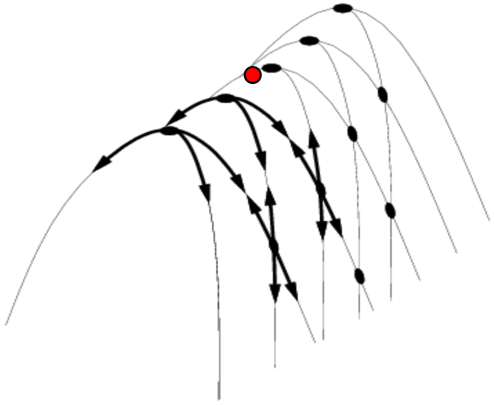
Local maxima



Plateaus



Diagonal ridges



What is it sensitive to?
Does it have any advantages?

Solving the Problems

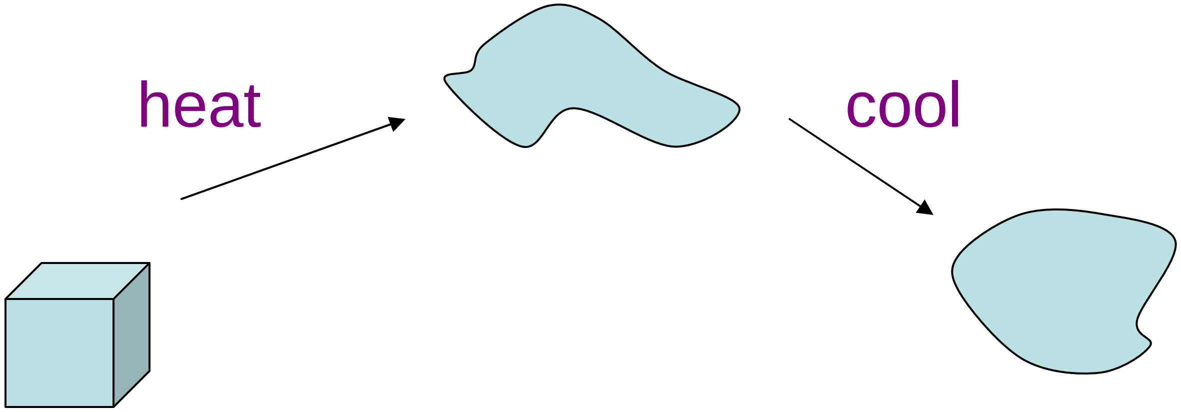
- Allow backtracking (What happens to complexity?)
- Stochastic hill climbing: choose at random from uphill moves, using steepness for a probability
- Random restarts: “If at first you don’t succeed, try, try again.”
- Several moves in each of several directions, then test
- Jump to a different part 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
  - ΔE/kT gets bigger
  - (-ΔE/kT) gets smaller
  - e^(-ΔE/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 = $e^{(-\Delta E/T)}$
- We need an annealing schedule, which is a sequence of values of T: $T_0, T_1, T_2, \ldots$

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

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

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

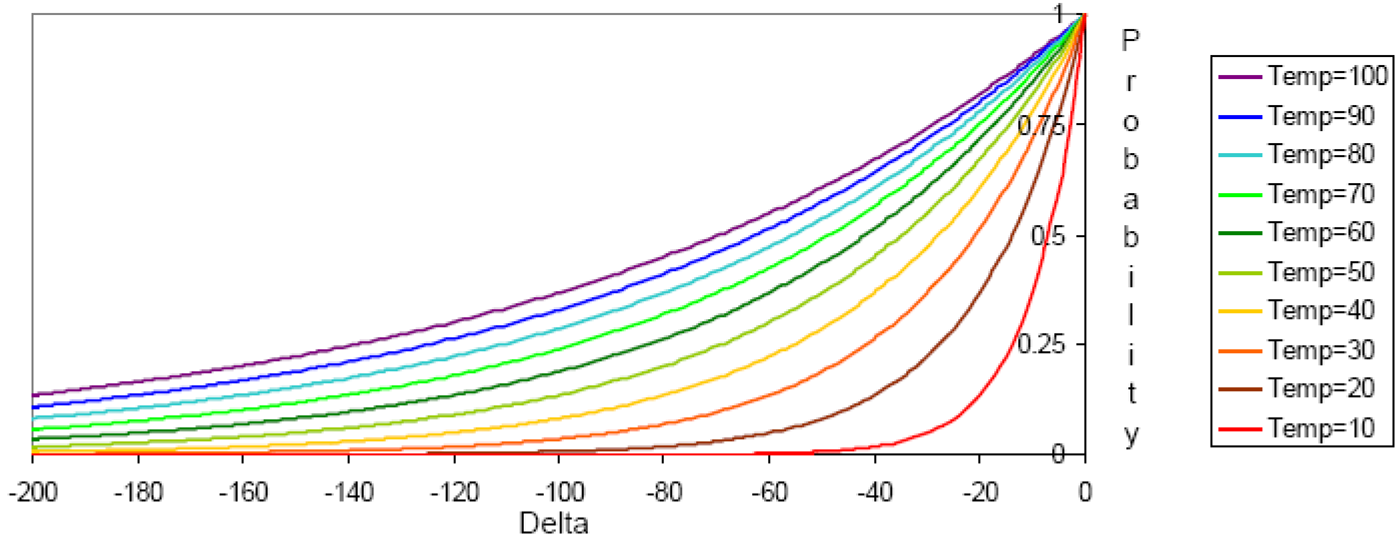
- Slowly enough has proven too slow for some researchers who have developed alternate schedules.

Simulated Annealing Schedules

- Acceptance criterion and cooling schedule

if (delta>=0) accept

else if ($random < e^{delta/Temp}$) accept, else reject /* 0<=random<=1 */



Initially temperature is very high (most bad moves accepted)

Temp slowly goes to 0, with multiple moves attempted at each temperature

Final runs with temp=0 (always reject bad moves) greedily “quench” the system

Simulated Annealing Applications

- Basic Problems
  - Traveling salesman
  - Graph partitioning
  - Matching problems
  - Graph coloring
  - Scheduling
- Engineering
  - VLSI design
    - Placement
    - Routing
    - Array logic minimization
    - Layout
  - Facilities layout
  - Image processing
  - Code design in information theory

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