Artificial Intelligence
AI-2002
Lecture 6

Mahzaib Younas
Lecturer Department of Computer Science
FAST NUCES CFD

Beyond Classical Search

Beyond Classical Search

- We have addressed a single category of problems: observable, deterministic, known environments
  - where the solution is a sequence of actions

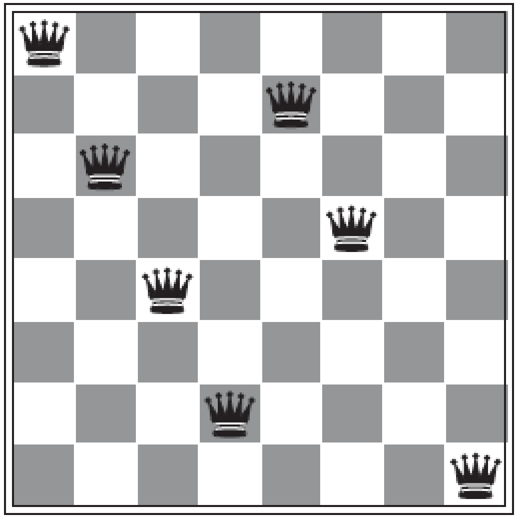
- The search algorithms that we have seen so far are designed to explore search spaces systematically.
  - When a goal is found, the ***path*** to that goal also constitutes a ***solution*** to the problem.

Beyond Classical Search

- In many problems, however, the **path to the goal is irrelevant**.

For example:

- 8-queens problem
  - *what matter, is the final configuration of queens, not the order*.
- The factory-floor layout problem
- The vehicle routing problem



Need Of Local Search

- The first issue is that
  - The algorithm tries to **explore the entire search space systematically**.
  - The algorithm visits the states in a **certain order** and it may **visit a lot of the states before finding a goal**.
  - There are some obvious problems with this behaviour. If the **search space is big, systematic exploration will take a long time**.
  - If the search space is **infinite, we cannot hope to visit all of states**
  - the search algorithm **remembers and returns a path from the initial state to the goal state**

Properties Of Local Search (Cont…)

- First, local search **does not attempt to explore the search space systematically**.
- Second, local search **only remembers the current state and does not keep track of a path to the goal** node

Local Search Algorithms

Local Search Algorithm

local search algorithms **give up on exploring the search space systematically**.

instead of attempting to visit all of the states, local search uses **strategies to find reasonably good states quickly on average**

good enough in practice especially when we are **solving a challenging problem under time constraint**.

Local Search Algorithm

A local search problem consists of:

- A state
    - a complete assignment to all of the variables.
- A neighbour relation
    - which states do I explore next?
- A cost function
    - how good is each state?

Local Search Algorithms

- Operate using a **single current node** and generally move only to neighbors of that node.
  - *No concern with paths* followed by the search
  - They are *NOT systematic*

- Local search algorithms have <u>**two key advantages**</u>:
  - they use very *little memory*
  - they can often find reasonable *solutions in large* or *infinite (continuous)* state spaces

Local Search Algorithms

- The local search algorithms are useful for solving pure **optimization problems**,

Optimization problems:

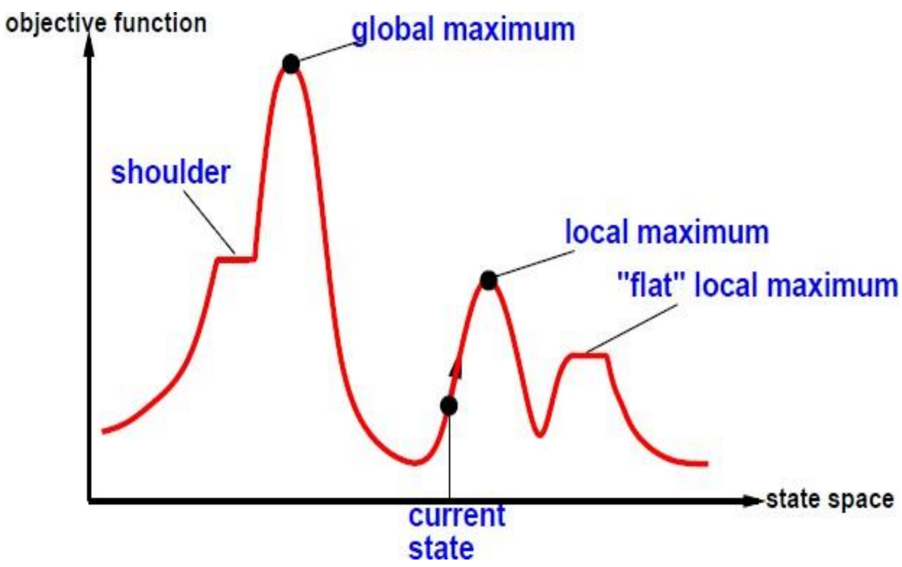
- the aim is **to find the best state** according to an *objective function*.
- The *standard search methods do not fit well* with optimization problems.

Local Search Algorithms

State-space landscape

A landscape has both

- "**location**"
  - defined by the state
- "**elevation**"
  - defined by the value of the **heuristic cost function** or **objective function**.



Local Search Algorithms

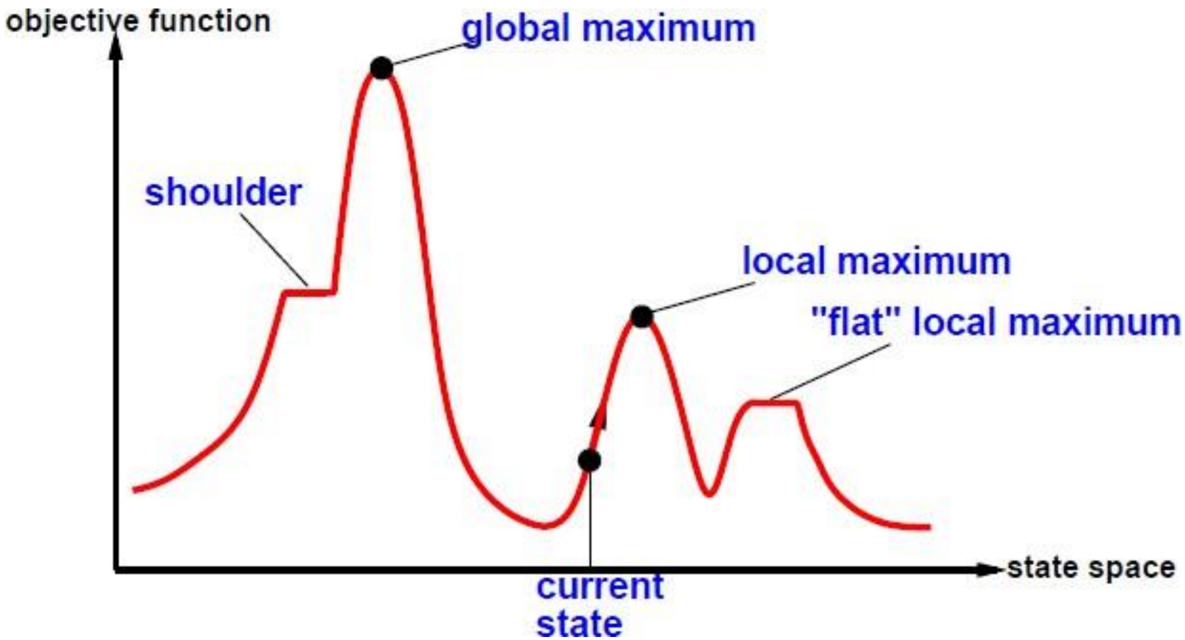
State-space landscape

- If elevation corresponds to **cost**, then the aim is to find the lowest valley—a **global minimum**;

- If elevation corresponds to an **objective function**, then the aim is to find the highest peak—a **global maximum**.

Local Search Algorithms

State-space landscape



A **one-dimensional state-space landscape** **in which** ***elevation corresponds to the objective function*****. The aim is to find the** **global maximum****.**

Local Search Algorithms

- A **complete local search** algorithm always finds a goal if one exists;
- An **optimal** algorithm always finds a global minimum/maximum.

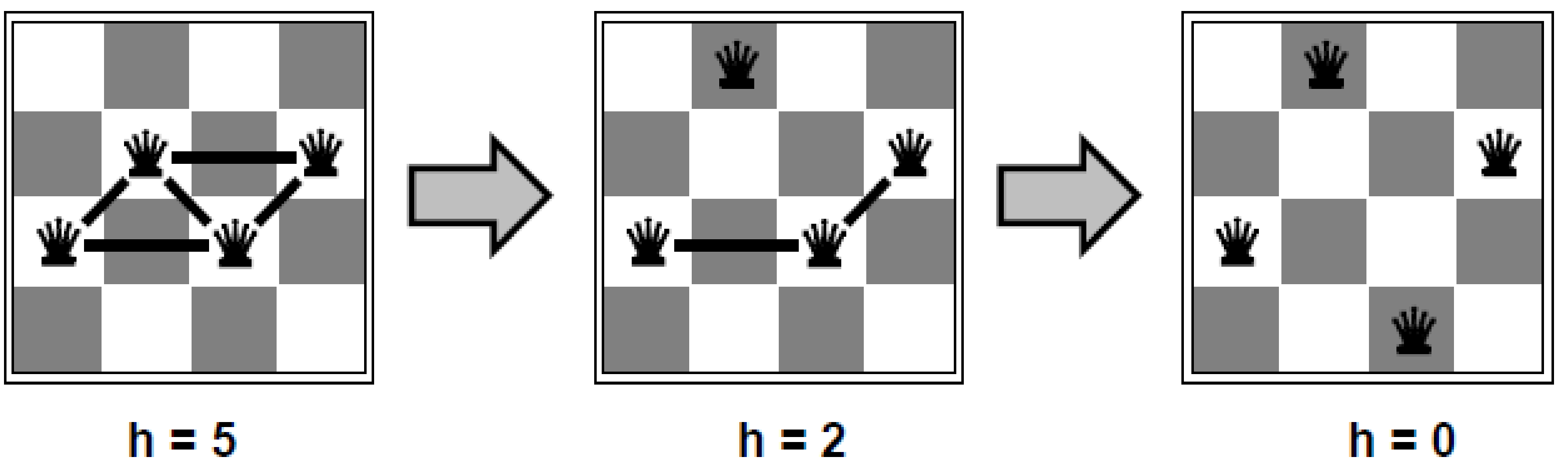
- Local search algorithms typically use a **complete-state formulation**,
  - In 8-queen problem, where *each state has 8 queens* on the board, one per column.

Terminology

Terminology	Purpose
Shoulder	It is a region having an edge upwards and it is also considered as one of the problems in hill climbing algorithms.
Global Maximum	a state that maximizes the objective function over the entire landscape.
Local Maximum	it is the state which is slightly better than the neighbor states but it is always lower than the highest state.
Flat Maximum	f the neighbor states all having same value, they can be represented by a flat space (as seen from the diagram) which are known as flat local maximums.
Current state	It is the state which contains the presence of an active agent.

Example: n-queens Problems

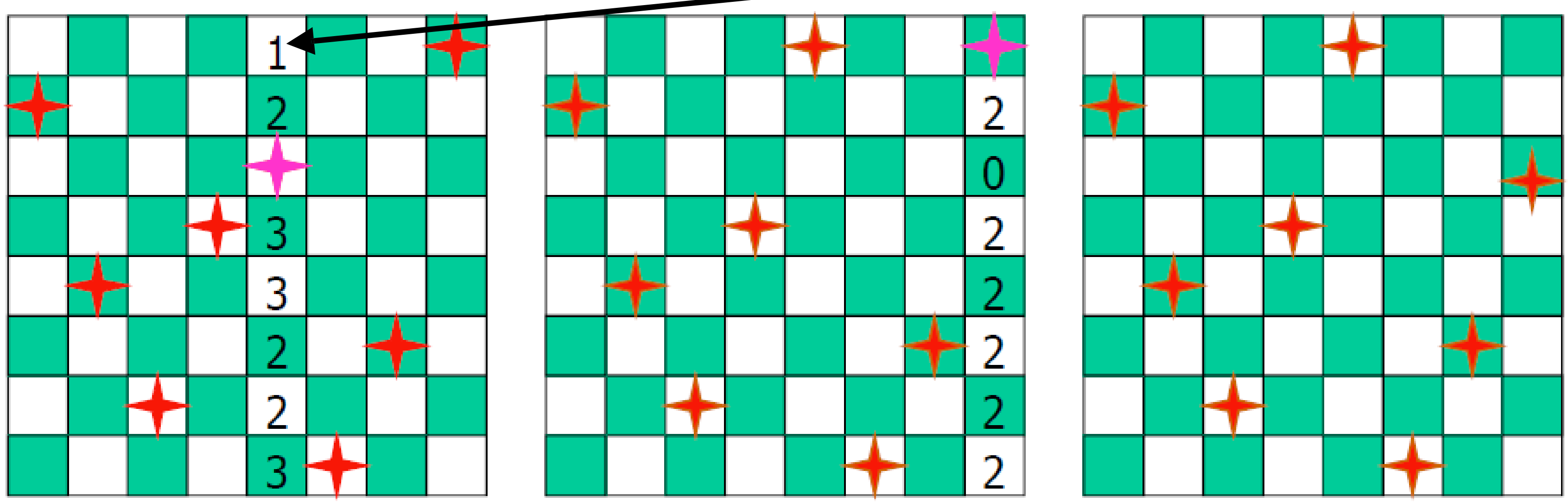
- Put $\boldsymbol{n}$ queens on an $\boldsymbol{n} \times \boldsymbol{n}$ board with no two queens on the same row, column, or diagonal
- Move a queen to reduce number of conflicts
- Heuristic $\boldsymbol{h}$: number of ‘attacks’



Example: n-queens Problems

- Put $\boldsymbol{n}$ queens on an $\boldsymbol{n} \times \boldsymbol{n}$ board with no two queens on the same row, column, or diagonal
- Move a queen to reduce number of conflicts
- Heuristic $\boldsymbol{h}$: number of ‘attacks’

If queen is placed here, there is only one attack then,



Hill-climbing Search

Hill-climbing Search

- It is simply a **loop** that continuously moves in the direction of increasing value—that is, uphill.
- It terminates when it reaches a "**peak**" where no neighbor has a higher value.
  - *does not maintain a* ***search tree***
  - need only

    record the state and

    the value of the *objective function*.
- Hill climbing only looks towards the immediate neighbors of the current state.

Hill-climbing Search

- The hill climbing **often gets stuck** for the following reasons:

Local maxima:

- A local maximum is a peak that is higher than each of its neighboring states but lower than the global maximum.

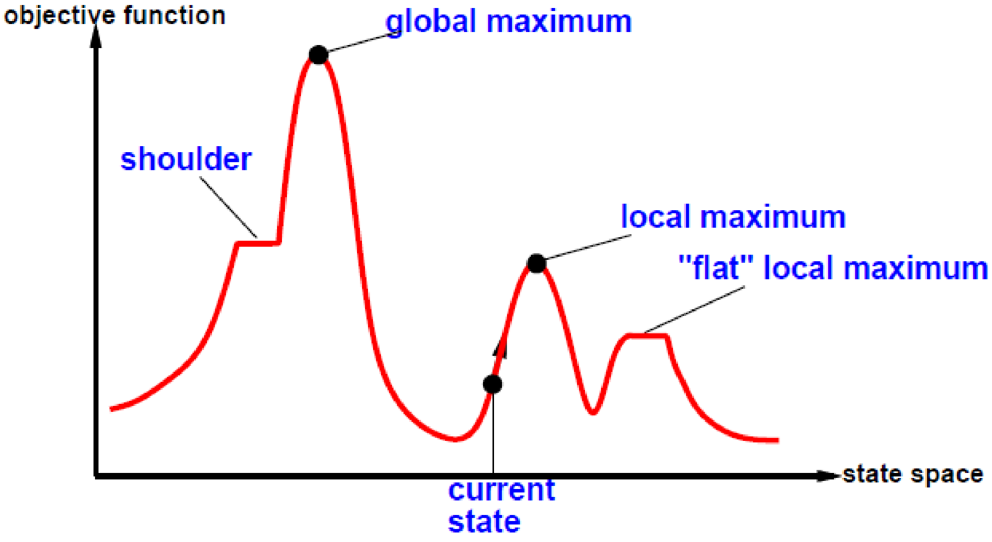
Hill-climbing Search

Plateaux:

- An area of the state space where the **evaluation function is flat**.
- It can be a **flat local maximum**, from which no uphill exit exists,

Shoulder:

- from which progress is possible

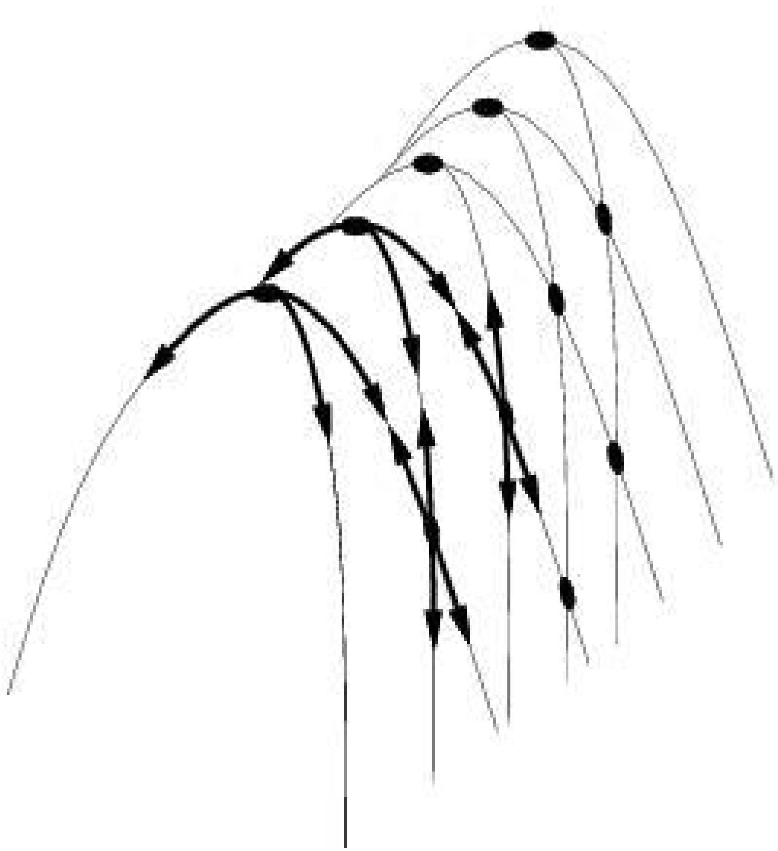


Hill-climbing Search

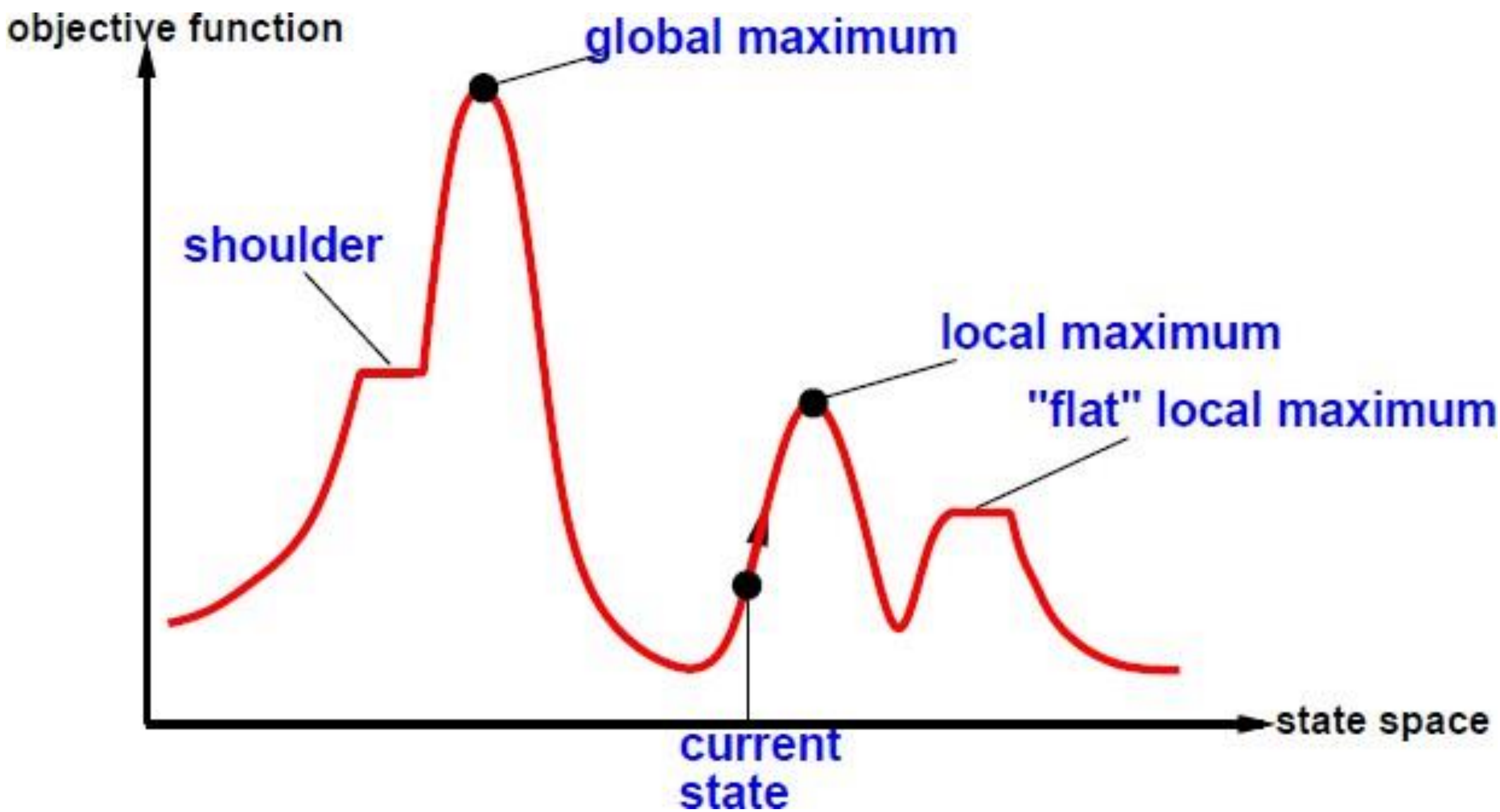
Ridges

:

- A sequence of local maxima
- *Its difficult for greedy algorithms to navigate*



Hill-climbing Search



Hill-climbing Search

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

current ← MAKE-NODE(*problem*.INITIAL-STATE)
loop do
  neighbor ← a highest-valued successor of *current*
  if neighbor.VALUE ≤ current.VALUE **then return** *current*.STATE
  current ← *neighbor*

- At *each step the current node is replaced by the best neighbor*; the neighbor with the highest VALUE,
- If a **heuristic cost estimate *h*** is used, we would find the neighbor with the ***lowest h***.

Hill Climb Search

- It is simply a **loop** that continuously moves in the direction of increasing value—that is, uphill.
- It terminates when it reaches a "**peak**" where no
- neighbor has a higher value.
  - **does not maintain a search tree**
  - need only
    - record the state and
    - the value of the **objective function.**
- Hill climbing only looks towards the immediate neighbors of the current state.

Hill Climb Search (Example)

- With randomly generated 8-queens starting states, the steepest-ascent hill climbing:
  - 14% of the time it solves the problem
  - 86% of the time it get stuck at a local minimum

Types/Variants of Hill Climb

- **Stochastic hill-climbing:**
  - Chooses randomly among potential successors
  - Sometimes better than steepest ascent
- **First-choice hill-climbing:**
  - Generates successors randomly and picks first
  - Good for many successors
- **Random restart hill-climbing:**
  - Restarts from randomly generated initial state when failed
  - Roughly 7 iterations with 8-queens problem

Hill Climb Search (Cont…)

- The success of hill-climbing search depends very much
  - on the shape of the state-space landscape
- If there are few local maxima and plateaux,
  - random-restart hill climbing will find a good solution very quickly.

Local Beam Search

Local Beam Search

Idea:

- Keep track of ***k*** states rather than just one
- Start with ***k*** randomly generated states
- At each iteration, all the successors of all *k* states are generated
- If any one is a goal state, stop;
- else select the ***k*** best successors from the complete list and repeat.

Local Beam Search

```
function BEAM-SEARCH(problem, k) returns a solution state
    start with k randomly generated states
    loop
        generate all successors of all k states
        if any of them is a solution then return it
        else select the k best successors
```

A local beam search with *k* states might seem to be nothing more than running ***k* random restarts in parallel** instead of in sequence.

Local Beam Search

Local beam search with *k* = 1

- We would randomly generate **1 start state**
- At each step we would generate all the successors, and retain the **1 best state**
- Equivalent to **HILL-CLIMBING**

Local Beam Search

Local beam search with $k = \infty$

1 initial state and no limit of the number of states retained

We start at initial state and **generate ALL successor** states (no limit how many)

If one of those is a goal, we stop

Otherwise, we generate all successors of those states (2 steps from the initial state), and continue

Equivalent to **BREADTH-FIRST SEARCH** except that

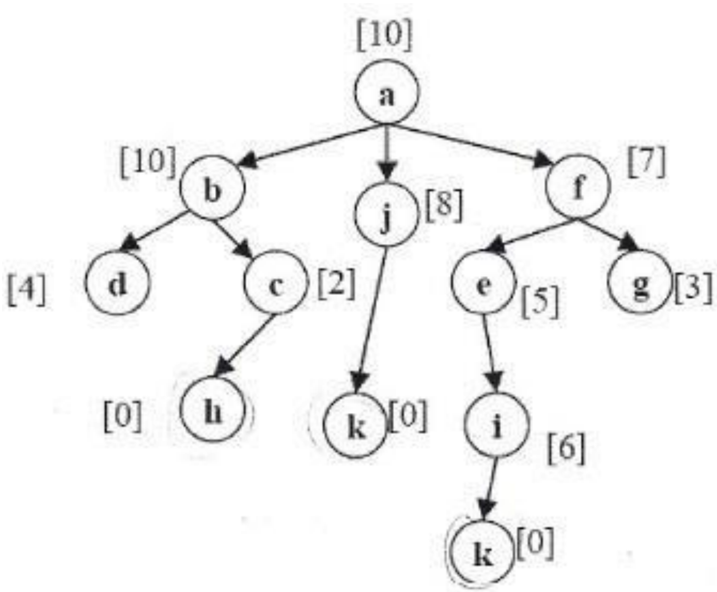
each layer is generated all at once.

Hill Climbing Vs. Beam Search

- Hill climbing just explores all nodes in one branch until goal found or not being able to explore more nodes.
- Beam search explores more than one path together. A factor **k** is used to determine the number of branches explored at a time.
- If k=2, then two branches are explored at a time. For k=4, four branches are explored simultaneously.
- The branches selected are the **best branches** based on the used **heuristic evaluation function**.

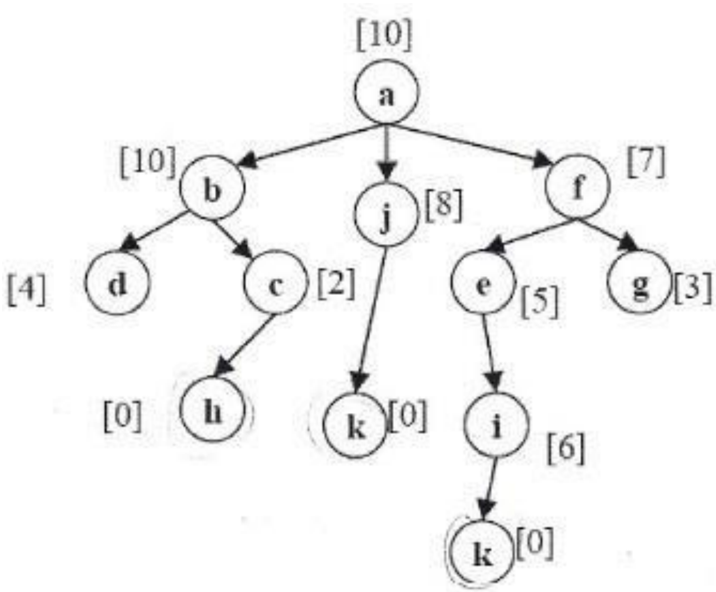
Beam Search, k=2
Goal – Node K

Current	Children
a	---



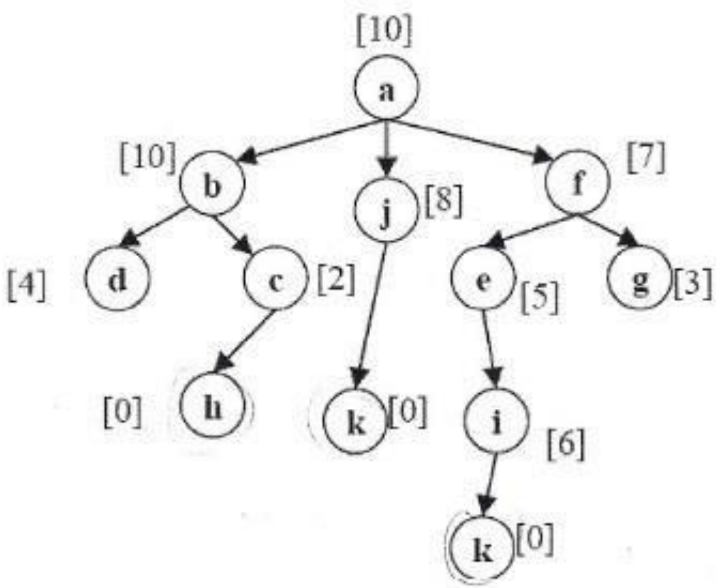
Beam Search
Goal – Node K

Current	Children
a	---



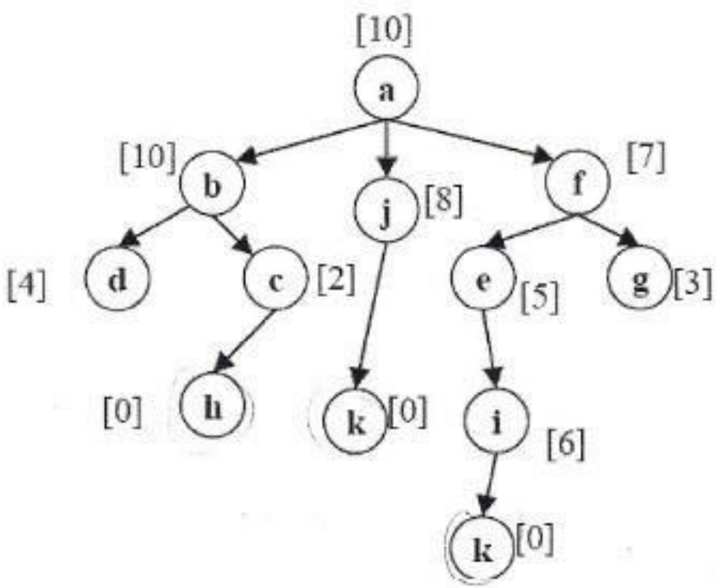
Beam Search
Goal – Node K

Current	Children
a	---
a	



Beam Search
Goal – Node K

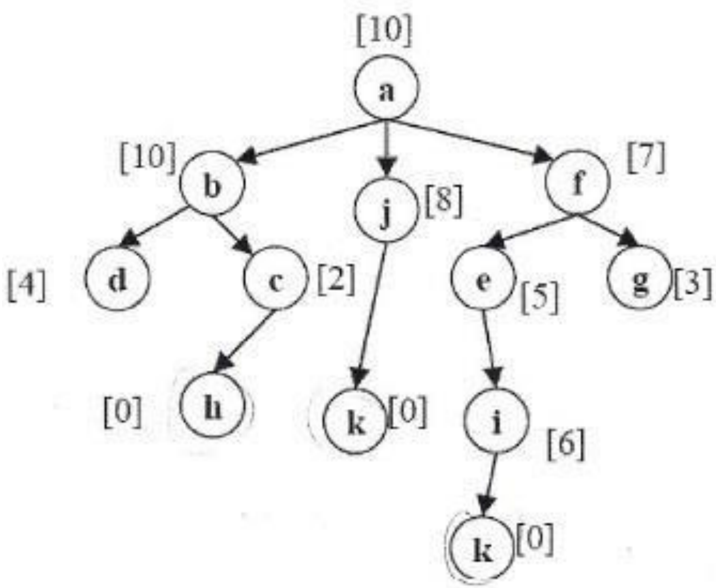
Current	Children
a	---
a	$f7, j8, b10$



Beam Search

Goal – **Node K**

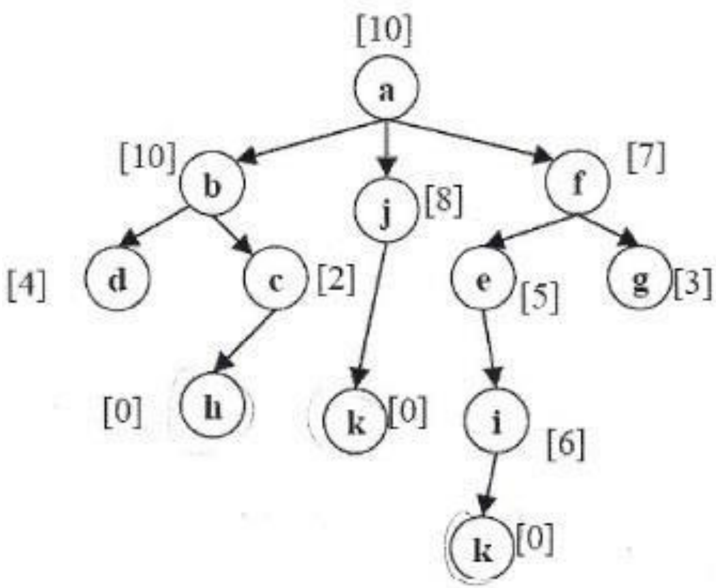
Best k Successors	Current	Children
	a	---
	a	$f7, j8, b10$



Beam Search

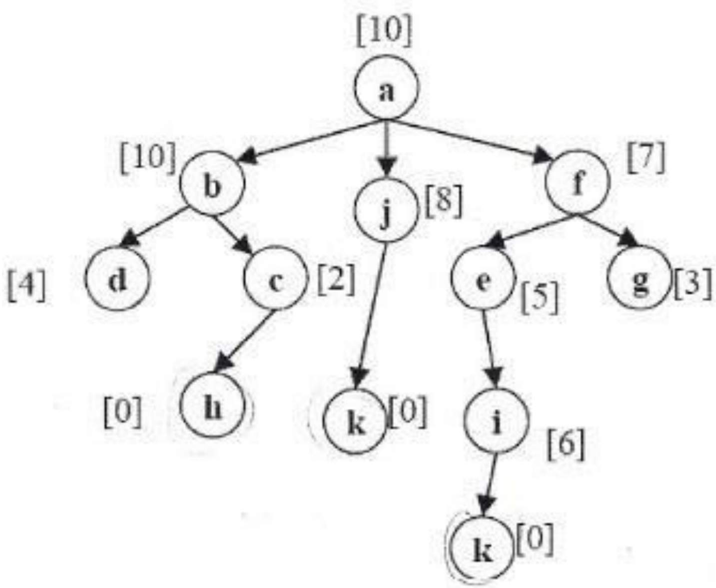
Goal – **Node K**

Best k Successors	Current	Children
	a	---
	a	$f7, j8$, $b10$

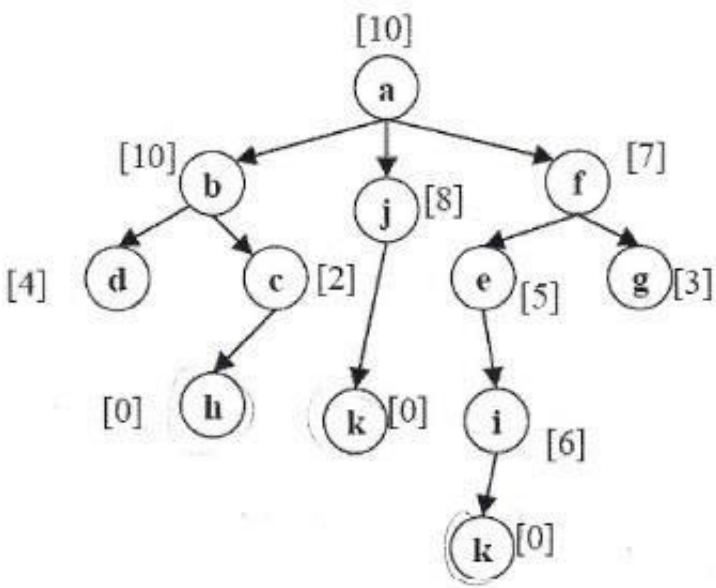


Beam Search
Goal – Node K

Current	Children
a	---
a	f7, j8,b10
f j	

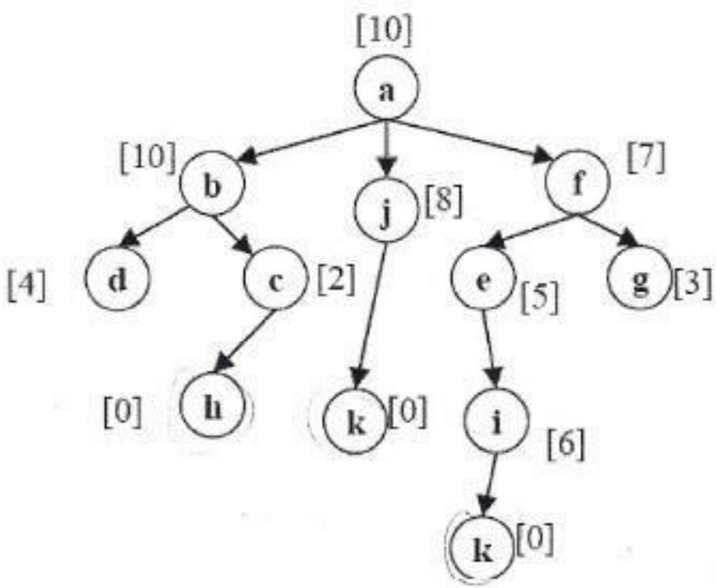


Beam Search
Goal – Node K



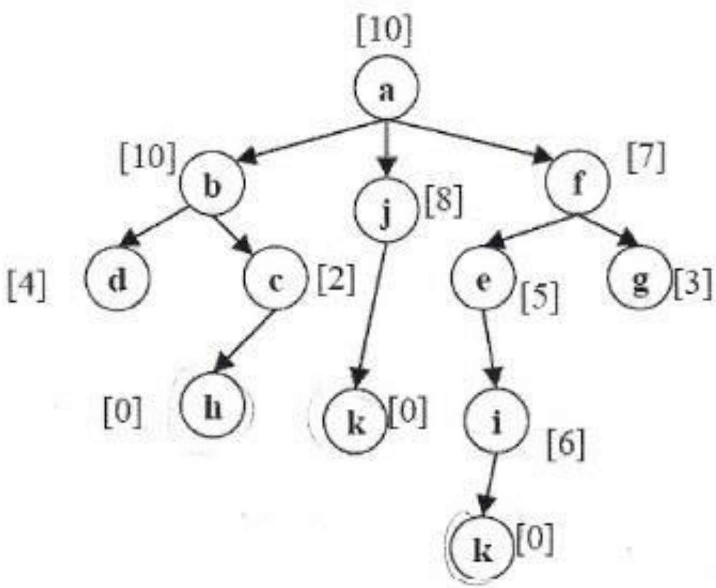
Current	Children
a	---
a	f7, j8,b10
f j	

Beam Search
Goal – Node K



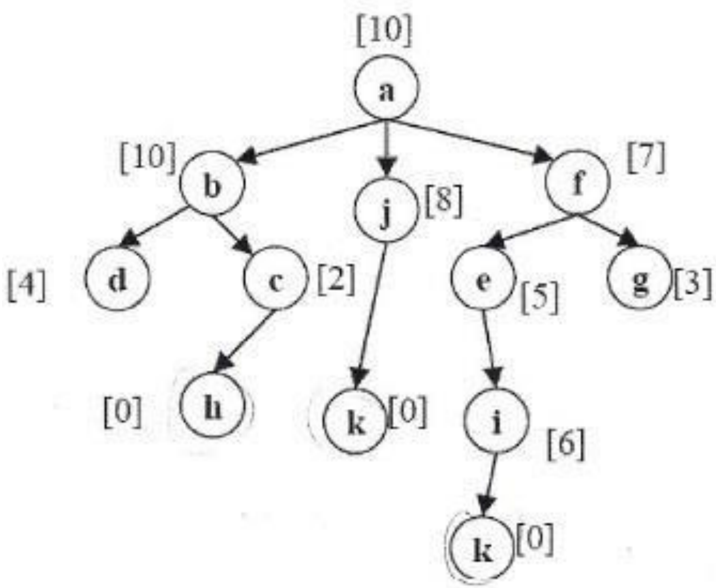
Current	Children
a	---
a	f7, *j*8,*b*10
f j	

Beam Search
Goal – Node K



Current		Children
a		---
a		f7, j8, b10
f	j	
g3, e 5	K0	

Beam Search
Goal – Node K



Current	Children
a	---
a	f7, *j*8,*b*10

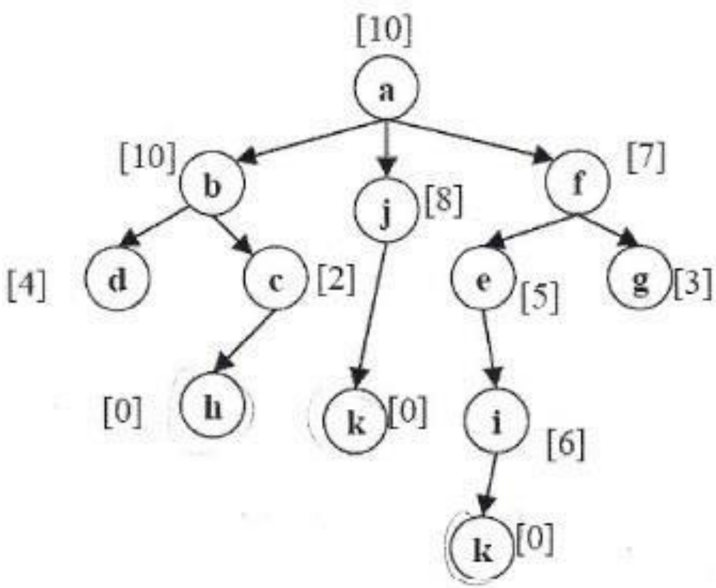
f j

g3, *e* 5 K0

k0, *g*3, *e*5

Beam Search

Goal – **Node K**

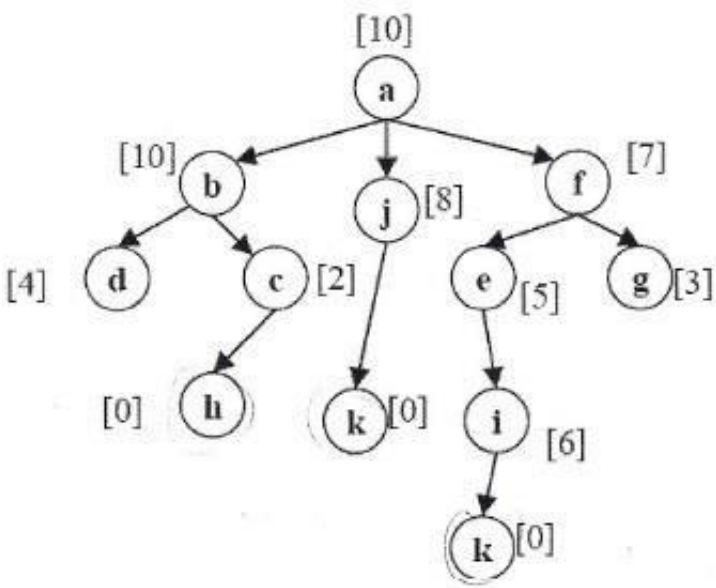


Best k Successors	Current	Children
	a	---
	a	f7, $j8$,$b10$
	f j	
	$g3$, e 5	K0
	k0, $g3$, $e5$	

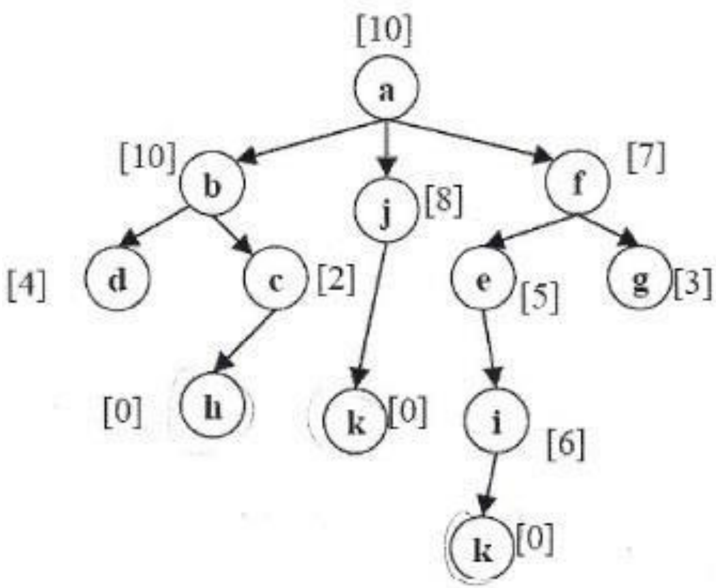
Beam Search
Goal – **Node K**

Best k Successors

Current	Children
a	---
a	f7, *j*8,*b*10
f j	
g3, *e* 5 K0	
*k*0, *g*3, *e*5	

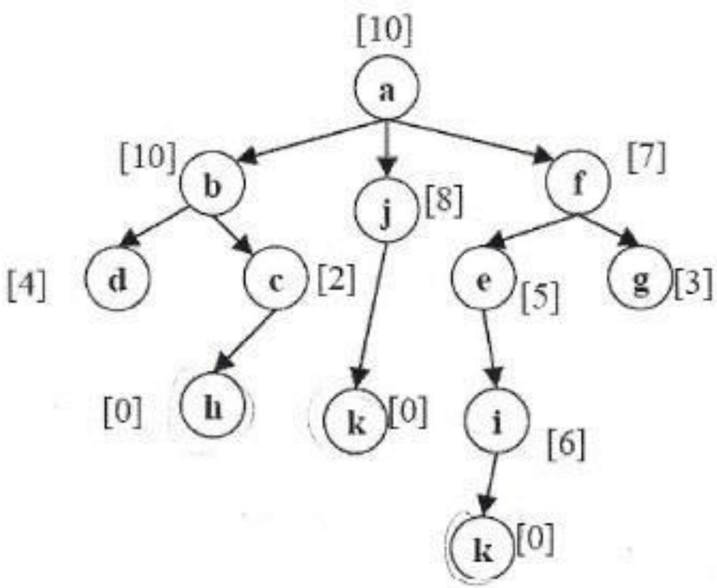


Beam Search
Goal – Node K



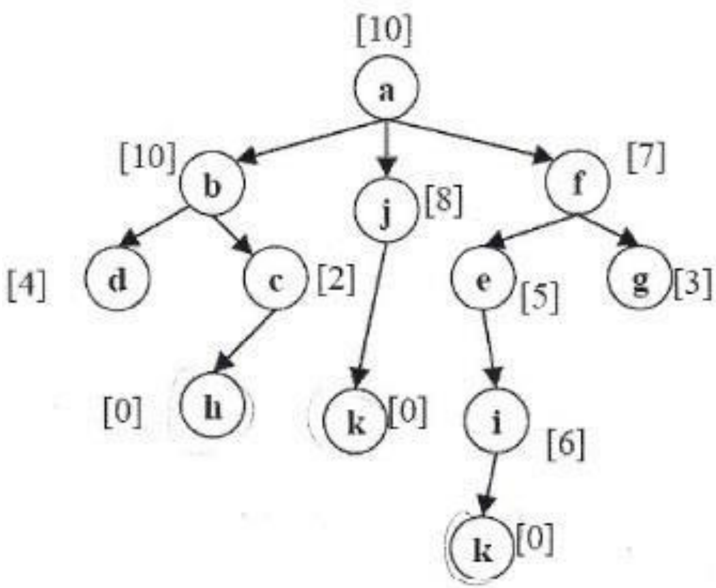
Current	Children
a	---
a	f7, j8, b10
f	j
g3, e 5	K0
K0, g3, e5	
k	g

Beam Search
Goal – Node K



Current	Children
a	---
a	f7, j8, b10
f j	
g3, e 5 K0	
K0, g3, e5	
K0 g3	

Beam Search
Goal – Node K



Current	Children
a	---
a	f7, $j8$, $b10$
f j	
g3, e 5	K0
K0, $g3$, $e5$	
K0	g3

GOAL

Simulated Annealing

Simulated Anneling

- Idea:
  - escape local maxima by allowing some “bad” moves but gradually decrease the size and frequency of the bad moves,
- In thermodynamics, the probability to go from a state with energy $E1$ to a state of energy $E2$ is given by:

$$p = e^{\frac{(E2 - E1)}{kT}} = e^{\frac{-(E1 - E2)}{kT}}$$

Simulated Anneling (cont…)

- e is Euler’s number
- T is a “temperature" controlling the probability of downward steps
- k is Boltzmann’s constant
  - (relating energy and temperature; with appropriate choice of units, it will be equal to 1).

Simulated Annealing

$$p = e^{\frac{-(E_1 - E_2)}{kT}}$$

Where,

- *e* is Euler’s number
- *T* is a “temperature" controlling the probability of downward steps
- *k* is Boltzmann’s constant
  - (relating energy and temperature; with appropriate choice of units, it will be equal to 1).

Simulated Annealing

$$p = e^{\frac{(E_2 - E_1)}{kT}} = e^{\frac{-(E_1 - E_2)}{kT}}$$

The idea is that probability decreases exponentially with $E_2 - E_1$ increasing,

The probability gets lower as temperature decreases

If the ***schedule*** lowers T slowly enough, the algorithm will find a global optimum with probability approaching 1.

Simulated Annealing

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"
  local variables: *current*, a node
    next, a node
    T, a "temperature" controlling prob. of downward steps

  current ← MAKE-NODE(INITIAL-STATE[*problem*])
  for $t \leftarrow 1$ **to** ∞ **do**
    $T \leftarrow schedule[t]$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE[*next*] – VALUE[*current*]
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

Similar to hill climbing, but a random move instead of best move

case of improvement, make the move

Otherwise, choose the move with probability that decreases exponentially with the "badness" of the move.

Simulated Annealing…Example

Consider there are 3 moves available, with changes in the objective function of

$$\Delta E_1 = -0.1, \Delta E_2 = 0.5, \Delta E_3 = -3$$

Suppose $T = 1$

Pick a move randomly:

- if ΔE_2 is picked, move there.
- if ΔE_1 or ΔE_3 are picked, probability of move = $e^{\frac{\Delta E}{T}}$
  - move 1: prob1 = $e^{-0.1} = 0.9$,
    i.e., 90% of the time we will accept this move
  - move 3: prob3 = $e^{-3} = 0.0497$
    i.e., 5% of the time we will accept this move

Simulated Annealing

T = "temperature" parameter

If *T* is high => the probability of "locally bad" move is higher

If *T* is low => the probability of "locally bad" move is lower

typically, *T* is decreased as the algorithm runs longer

- i.e., there is a "temperature schedule"

Simulated Annealing

Convergence:

With exponential schedule, will provably converge to global optimum

- If *T* decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1.

Few more precise convergence rate

Recent work on **rapidly mixing Markov chains**.

Surprisingly, deep foundations.

Simulated Annealing

method proposed in 1983 by IBM researchers for solving VLSI layout problems.

- theoretically will always **find the global optimum** (the best solution)

Useful for some problems, but can be very **slow**

- slowness comes about because *T* must be decreased very gradually to retain optimality

In practice *how to decide the rate* at which to decrease

T? (this is a practical problem with this method)