Local Search Algorithms

Dr. Sonali Patil

Search Types

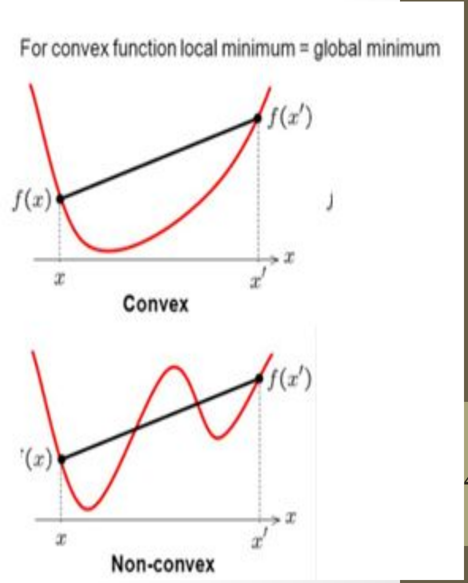
- Backtracking state-space search
- Local Search and Optimization
  - Hill Climbing
  - Escaping local Maxima: Simulated Annealing
  - Genetic Algorithms
- Constraint satisfaction search
- Adversarial search

Local Search and Optimization

- Previous searches: Builds a search tree, keep paths in memory, and remember alternatives so search can backtrack. Solution is a path to a goal.
- Path may be irrelevant, if only the final configuration is needed (8-queens, 8-puzzle, IC design, network optimization, …)

Local Search and Optimization

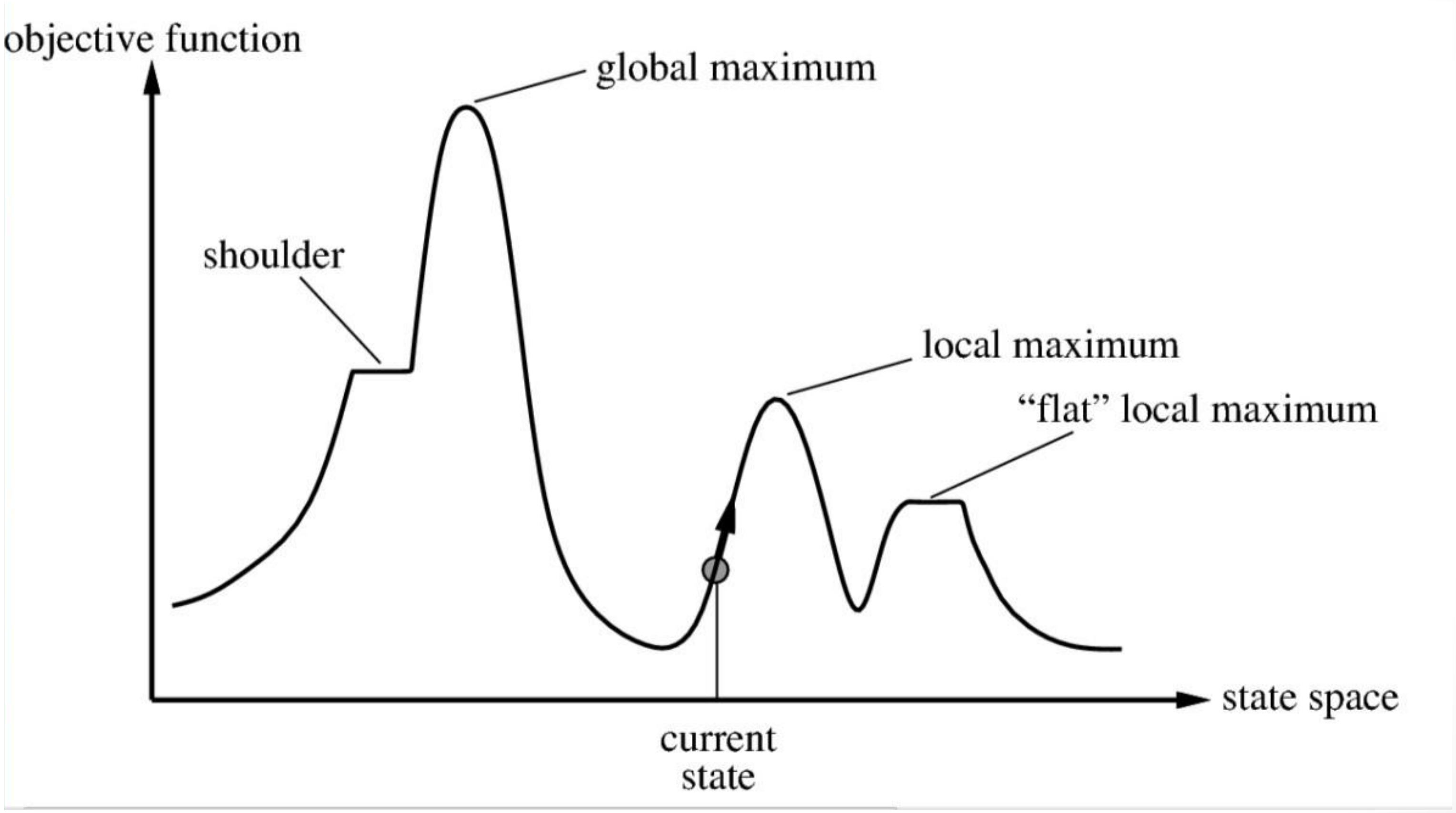
- **Local search:**
  - Use single current state and move to neighbouring states.
- **Idea: start with an initial guess at a solution and incrementally improve it until it is one**
- **Advantages:**
  - Use very little memory
  - Find often *reasonable* solutions in large or infinite state spaces.
- **Useful for pure optimization problems.**
  - Find or approximate best state according to some *objective function*
  - *Optimal if the space to be searched is convex*



Optimization

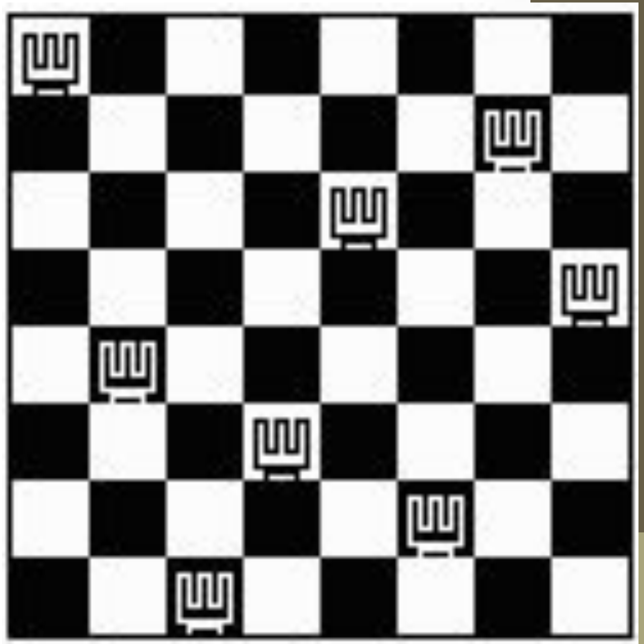
- Local search is often suitable for optimization problems. Search for best state by optimizing an objective function.
- F(x) where often x is a vector of continuous or discrete values
- Begin with a complete configuration
- A successor of state S is S with a single element changed
- Move from the current state to a successor state
- Low memory requirements, because the search tree or graph is not maintained in memory (paths are not saved)

State Space Features



Examples

- 8 queens: find an arrangement of 8 queens on a chess board such that no two queens are attacking each other
- Start with some arrangement of the queens, one per column
- X[j]: row of queen in column j
- Successors of a state: move one queen
- F: # pairs attacking each other



Examples

- Traveling salesman problem: visit each city exactly once
- Start with some ordering of the cities
- State representation – order of the cities visited (for eg)
- Successor state: a change to the current ordering
- F: length of the route

Examples

- Flight Travel problem
- See file on schedule. We will look at the details later, but the general problem is for all members of the Glass family to travel to the same place.
- A state consists of flights for each member of the family.
- Successor states: All schedules that have one person on the next later or the next earlier departing or returning flight.
- F: sum of different types of costs ($$, time, etc)

Examples

- Cryptosystems: resistance to types of attacks is often discussed in terms of Boolean functions used in them
- Much work on constructing Boolean functions with desired cryptographic properties (balancedness, high nonlinearity, etc.)
- One approach: local search, where states are represented with truth tables (that's a simplification); a successor results from a change to the truth table; objective functions have been devised to assess (estimate) relevant qualities of the Boolean functions

Examples

- Racing yacht hull design
- Design representation has multiple components, including a vector of B-Spline surfaces.
- Successors: modification of a B-Spline surface (e.g.).
- Objective function estimates the time the yacht would take to traverse a course given certain wind conditions (e.g.)

Comparison to tree/graphsearch framework

- Chapter 3: start state is typically not a complete configuration. Chapter 4: all states are
- Chapter 3: binary goal test; Chapter 4: no binary goal test, unless one can define one in terms of the objective function for the problem (e.g., no attacking pairs in 8-queens)
- h: estimate of the distance to the nearest goal
- objective function: preference/quality measure – how good is this state?
- Chapter 3: saving paths
- Chapter 4: start with a complete configuration and make modifications to improve it

Visualization

- States are laid out in a landscape
- Height corresponds to the objective function value
- Move around the landscape to find the highest (or lowest) peak
- Only keep track of the current states and immediate neighbors

Local Search Alogorithms

- Two strategies for choosing the state to visit next
  - Hill climbing
  - Simulated annealing
- Then, an extension to multiple current states:
  - Genetic algorithms

Hillclimbing (Greedy Local Search)

- Generate nearby successor states to the current state
- Pick the best and replace the current state with that one.
- Loop

```
Inputs: Problem, problem
Local variables: Current, a node
Next, a node
Current = MAKE-NODE(INITIAL-STATE[Problem])
Loop do
    Next = a highest-valued successor of Current
    If VALUE[Next] < VALUE[Current] then return Current
    Current = Next
End
```

Hillclimbing (Steepest Ascent Version)

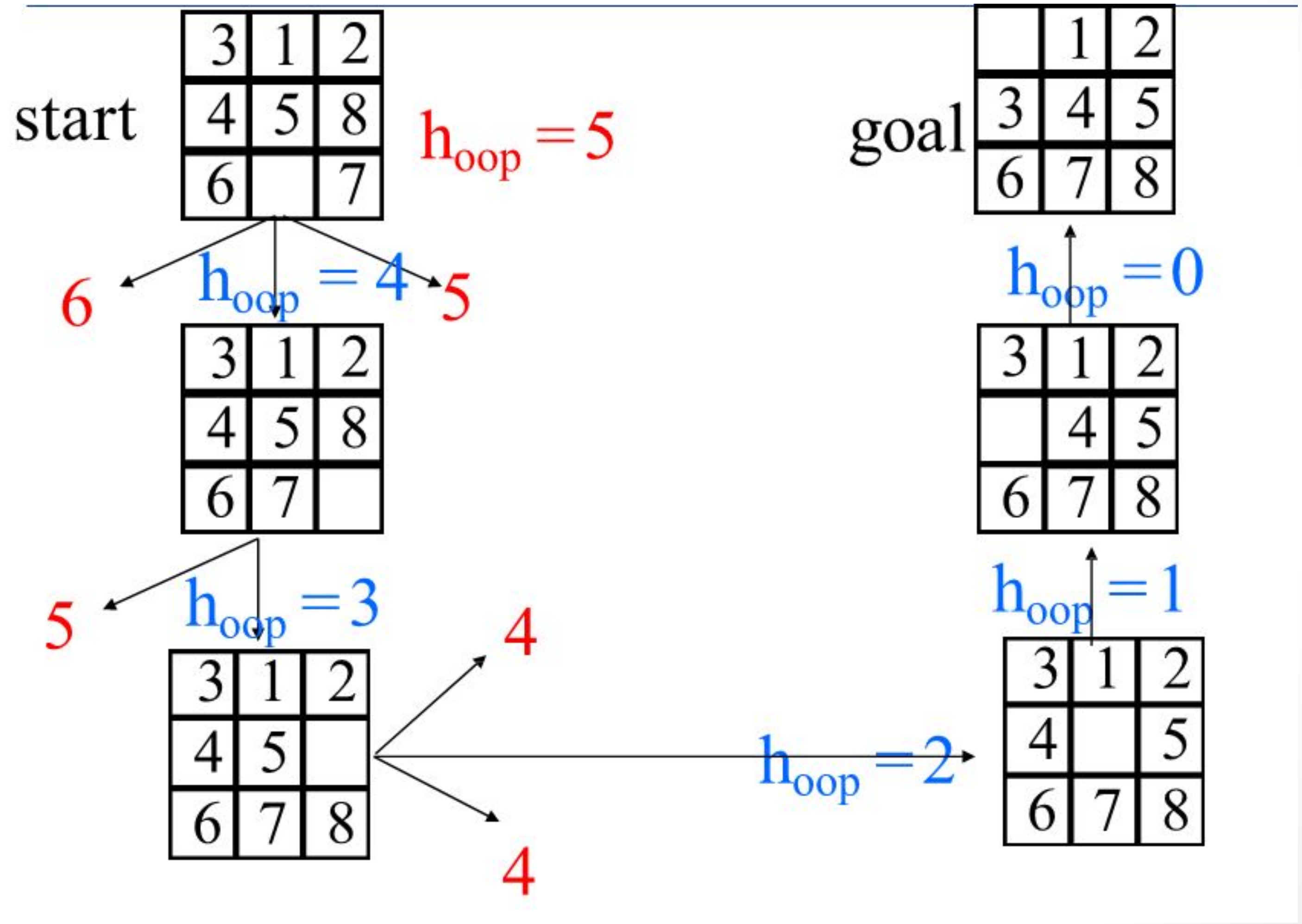
I. **While (∃ uphill points):**
   - Move in the direction of increasing evaluation function ***f***

II. **Let $s_{next} = \max f(s)$, s a successor state to the current state n**
   - If $f(n) < f(s)$ then move to s
   - Otherwise halt at n

- **Properties:**
  I. Terminates when a peak is reached.
  II. Does not look ahead of the immediate neighbors of the current state.
  III. Chooses randomly among the set of best successors, if there is more than one.
  IV. Doesn't *backtrack*, since it doesn't remember where it's been

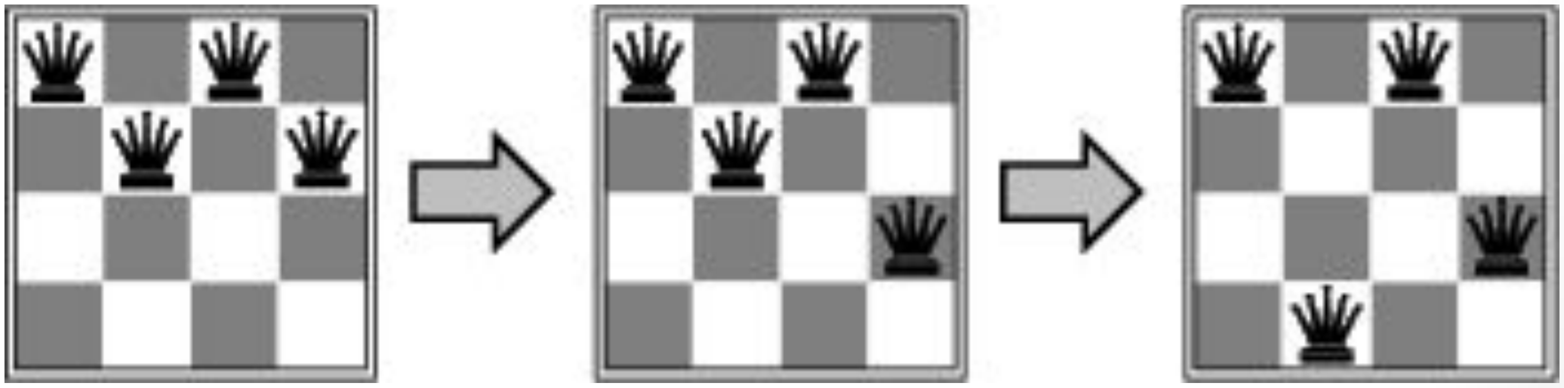
I. **a.k.a. *greedy local search*** as it only looks to its good immediate neighbor state and not beyond that
   - ***"Like climbing Everest in thick fog with amnesia"***

Hill Climbing Example (minimize h)



Hill Climbing Example: n-queens

- ***n*-queens problem: Put *n* queens on an *n* * *n* board with no two queens on the same row, column, or diagonal**
- ***Good heuristic: h* = number of pairs of queens that are attacking each other**

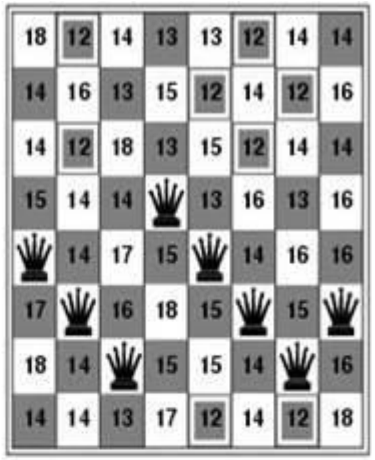


h=5

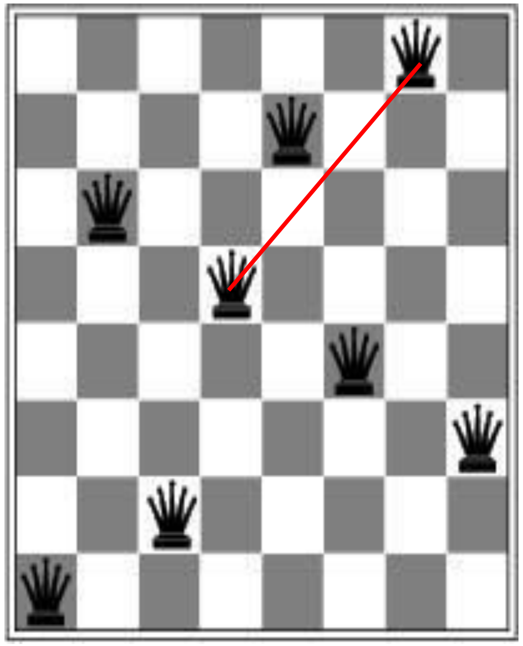
h=3
(for illustration)

h=1

Hill Climbing Example: 8-queens



A state with *h*=17 and the *h*-value for each possible successor



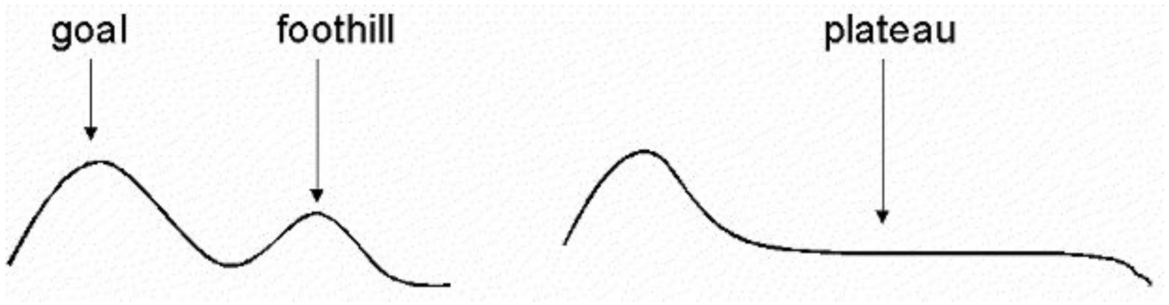
A local minimum of *h* in the 8-queens state space (*h*=1).

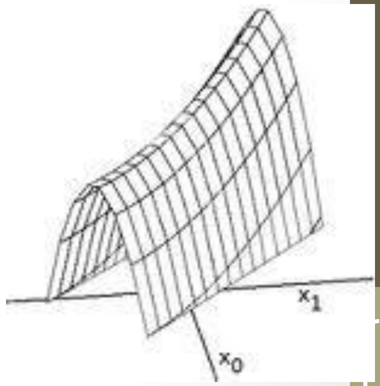
h **= number of pairs of queens that are attacking each other**

Hill-climbing search problems

(this slide assumes maximization rather than minimization)

- *Local maximum*: a peak that is lower than the highest peak, so a suboptimal solution is returned
- *Plateau*: the evaluation function is flat, resulting in a random walk. Give4s no direction to search algorithm)
- *Ridges*: slopes very gently toward a peak, so the search may oscillate from side to side. dropoffs to the sides; steps to the North, East, South and West may go down, but a step to the NW may go up



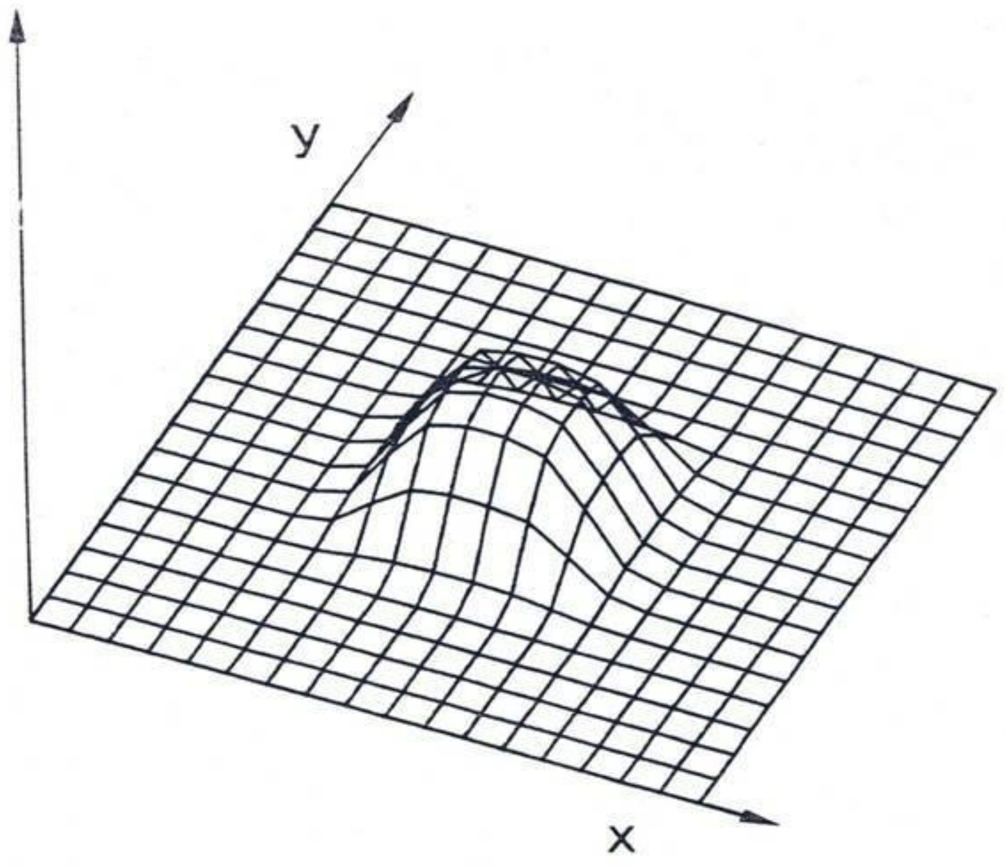


Local maximum

Plateau

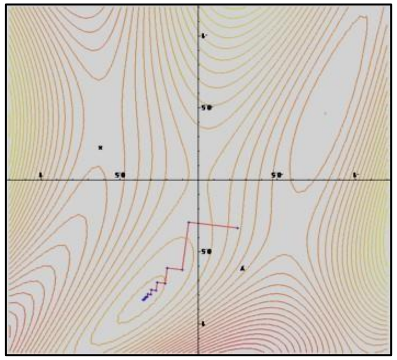
Ridge

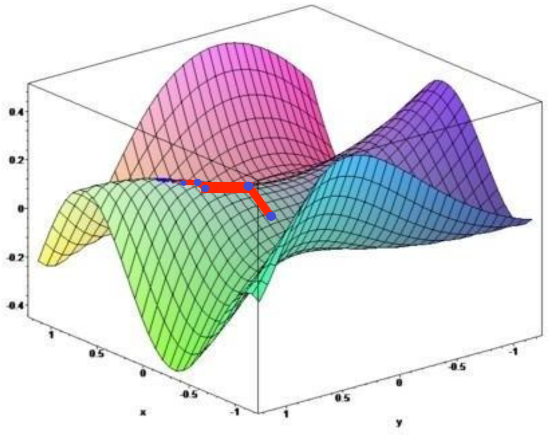
The Shape of an Easy Problem



This and next several slides from Goldberg '89

Hill Climbing Variant: Gradient Ascent/Descent

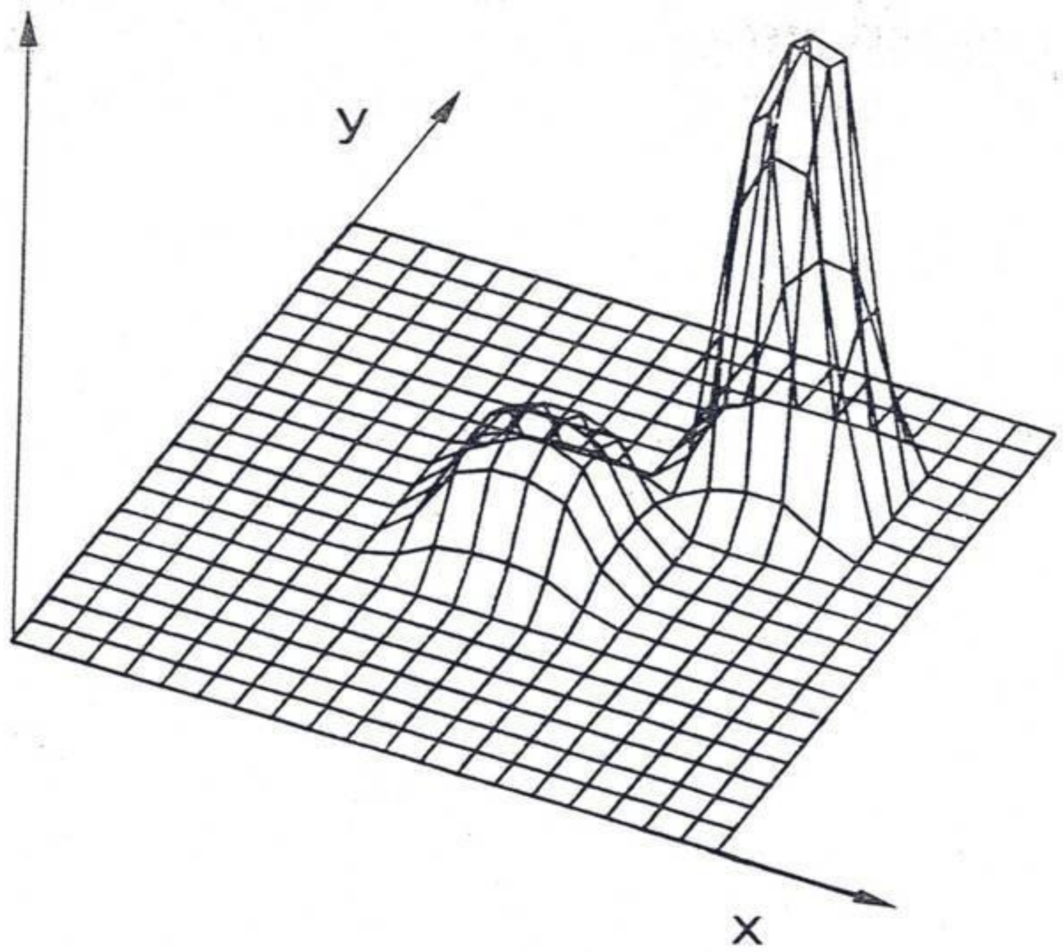




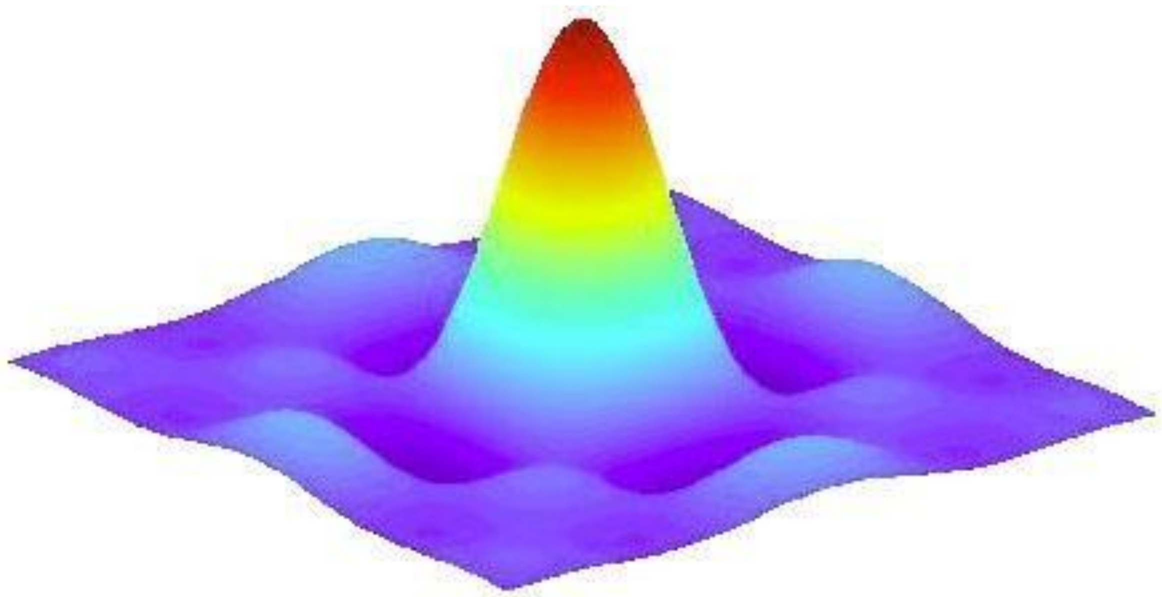
Images from http://en.wikipedia.org/wiki/Gradient_descent

- Gradient Ascent to maximize a function and Gradient Descent to minimize a function
- Gradient descent procedure for finding the $arg_x\ min\ \ f(x)$
  - choose initial x_0 randomly
  - repeat
    - $x_{i+1} \leftarrow x_i - \eta f'(x_i)$
  - until the sequence $x_0, x_1, \ldots, x_i, x_{i+1}$ converges
- Step size η (eta) is small (perhaps 0.1 or 0.05)

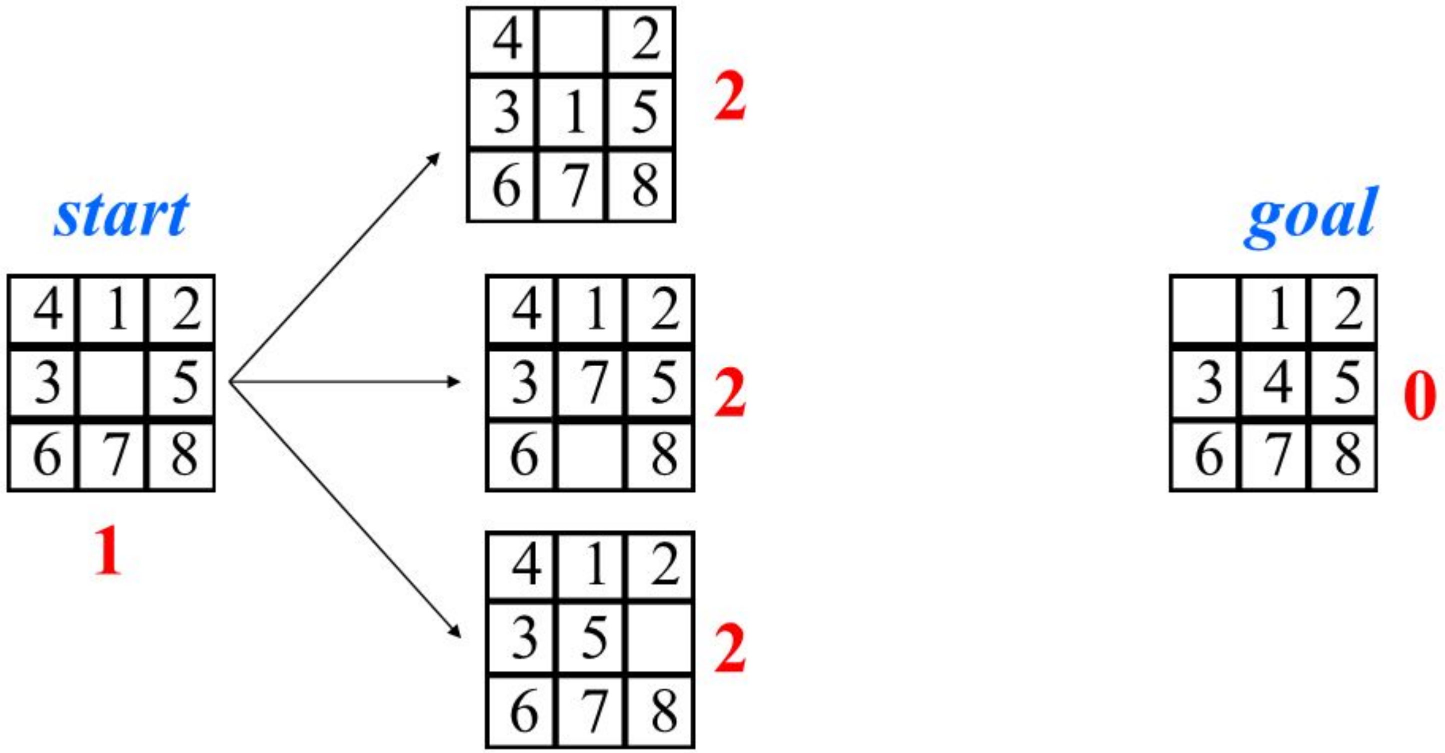
The Shape of a harder Problem



The Shape of a Yet harder Problem



Hill-climbing search problems Local maxima

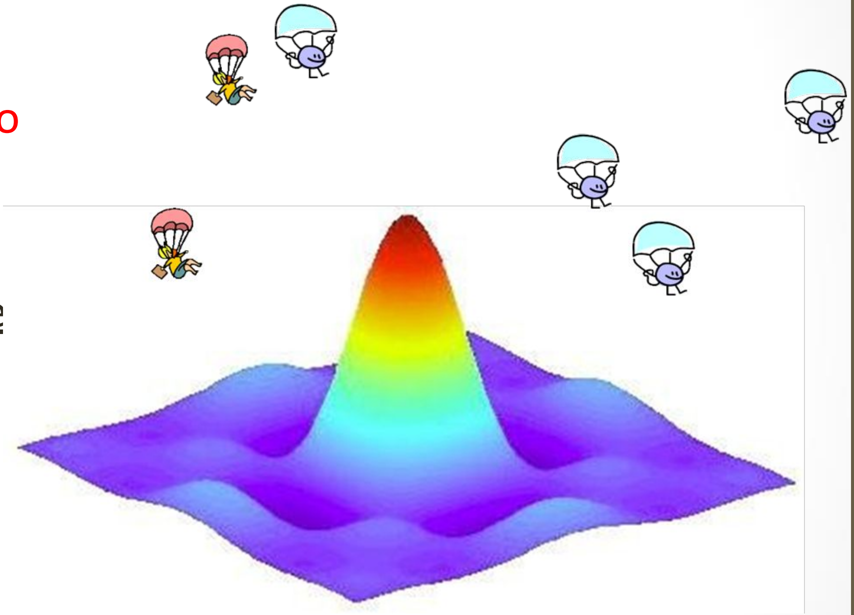


Hill Climbing: Variants of hill climbing

- Stochastic Hill Climbing:
  - Stochastic hill climbing does not examine for all its neighbor before moving. Rather, this search algorithm selects one neighbor node at random and decides whether to choose it as a current state or examine another state.
  - Usually converges more slowly than steepest ascent, finds better solution in some state land
- First Choice Hill Climbing:
  - Implements stochastic hill climbing by generating successors randomly until one is generated that is better than the current state
  - Good strategy when a state has many successors (thousands)

Remedy to Drawbacks of Hill Climbing: Random-restart hill-climbing

- Start different hill-climbing searches from random starting positions stopping when a goal is found
- Save the best result from any search so far
- If all states have equal probability of being generated, it is complete with probability approaching 1 (a goal state will eventually be generated).
- Finding an optimal solution becomes the question of sufficient number of restarts
- Surprisingly effective, if there aren't too many local maxima or plateaux



Simulated Annealing

- Why Simulated Annealing?
  - Hill Climbing: incomplete
  - Random Walk: complete but inefficient
  - Need an approach which can combine these benefits and yield an algorithm which is complete and efficient
  - Simulated Annealing is one such approach

Simulated Annealing (SA)

- It's a stochastic optimization technique
- Developed by Kirpatrick, Gelatt and Vecchi, 1983
- It is offshoot of the Metropolis Hasting algorithm, a very powerful sampling technique in statistics
- SA is a global optimization procedure
- Works very well for discrete optimization problems as well as for exceeding complex problems

Simulated Annealing

- Basic Philosophy: Consider cooling process of molten metals through annealing
- At high temperatures, the atoms in the molten state can move freely with respect to one another
- However, as the temperature is reduced, the movements are restricted
- The atoms begin to get ordered and finally form crystals with the minimum potential energy
- *The crucial parameter here is cooling rate*
- *If cooling rate is high, the crystalline state may not be achieved. Gradual cooling is the key*

Simulated Annealing

- We mimic the process of annealing in optimization. Hence the name Simulated Annealing
- Based on a metallurgical metaphor
  - Start with a temperature set very high and slowly reduce it.
  - Run hill climbing with the twist that you can occasionally replace the current state with a worse state based on the current temperature and how much worse the new state is.
- More formally…
  - Generate a random new neighbor from current state.
  - If it’s better take it.
  - If it’s worse then take it with some probability proportional to the temperature and the delta between the new and old states.

Simulated Annealing Algorithm

- SA is a point method
- We begin with an initial point and high temperature T
- A second point is created in the vicinity of the initial point and ΔE is calculated.
    - If ΔE is negative (next < current), the new point is accepted (for minimization problems)
    - Else, the point is accepted with a probability of $e^{\Delta E/kT}$ (k=1)*
    - This completes one iteration
- In the next generation/iteration, again a new point is chosen, but now the temperature is reduced according to cooling schedule
- If T is too large, slow convergence
- If T too small, may lead to premature convergence(local minima/maxima)

Simulated Annealing Algorithm

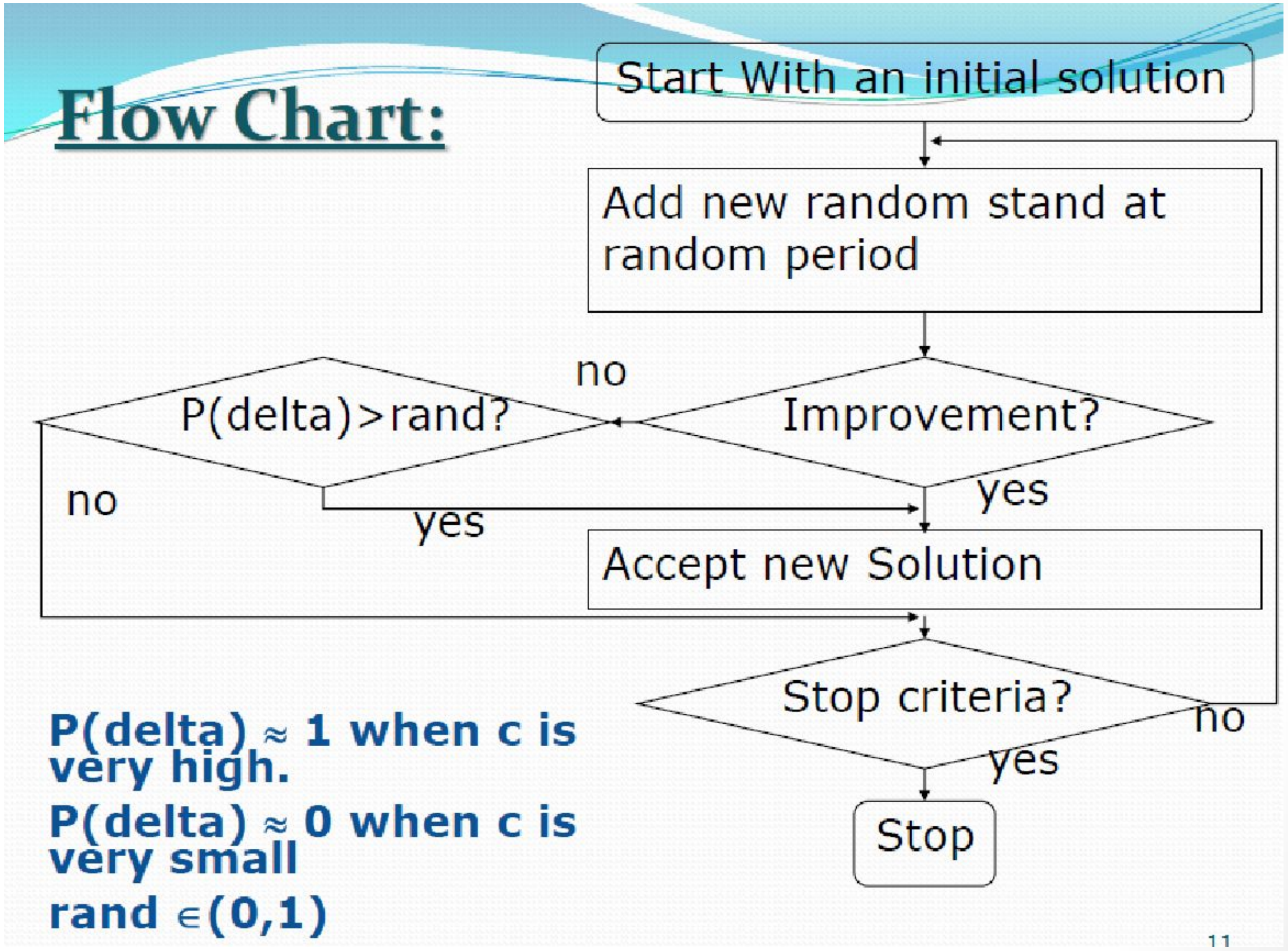
```
Function SIMULATED-ANNEALING( problem, schedule) returns a solution state
input: problem, a problem
schedule, a mapping from time to “temperature”
current <- MAKE-NODE(problem.INITIAL-STATE)
for t <- 1 to ∞ do
T <- schedule(t)
ifT = 0 then return current
Next <- a randomly selected successor of current
ΔE <- next.VALUE–current.VALUE
ifΔE > 0 then current <- next
Else current<-next only with probability $e^{\Delta E/T}$
```

Stopping conditions

- Stopping condition can be many:
- till T becomes zero,
- choose a v small threshold and compare Enext-Ecurr and stop if its < threshold etc
- A given minimum value of the temperature has been
- reached.
- A certain number of iterations (or temperatures) has
- passed without acceptance of a new solution.
- A specified number of total iterations has been
- executed

- How will be decide whether the worse point in the neighbourhood will be selected as next state or not???
- choose a random number r in range [0-1]. If the r is <= probability of acceptance which is calculated (e∆E/kT), accept the next as current and proceed.
- How to choose initial value: if we know the lower and upper bound of the variable, ini value can be choosen as (lower+upper)/2
- To calculate initial T draw a few random initial values(x) and calculate the average f(x) and use it as initial T???

Simulated Annealing Algorithm



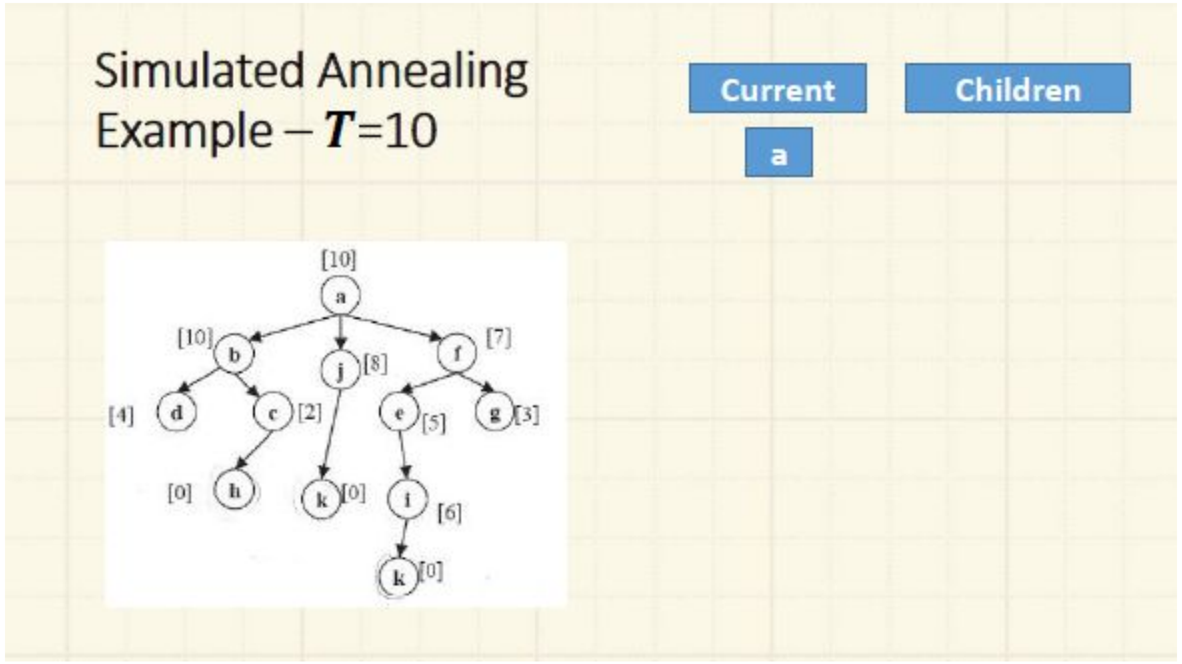
Flow Chart:

P(delta) ≈ 1 when c is very high.

P(delta) ≈ 0 when c is very small

rand ∈(0,1)

Simulated Annealing Problem

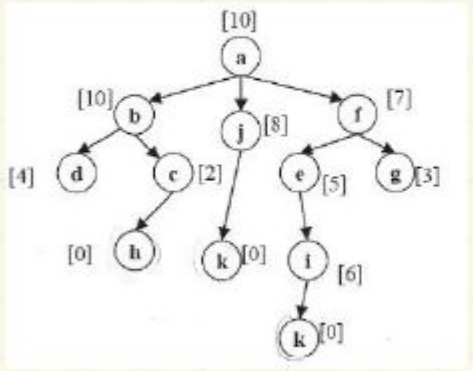


Simulated Annealing
Example – ***T***=10

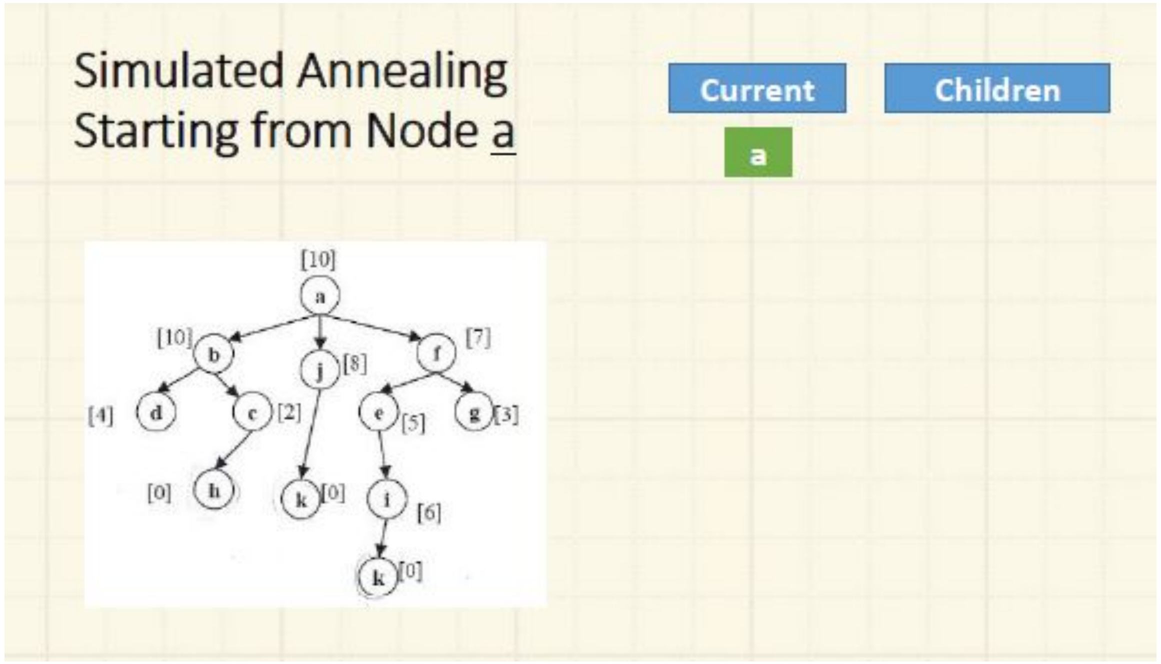
Simulated Annealing Problem

Simulated Annealing
Starting from Node a

Current	Children
a	



Simulated Annealing Problem



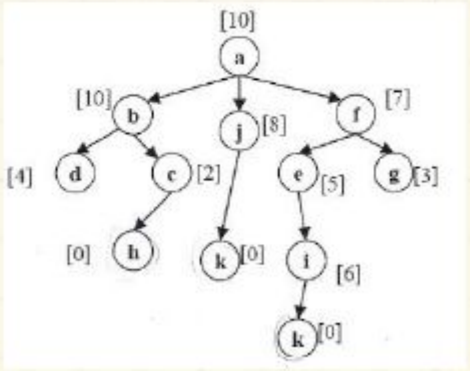
Simulated Annealing
Starting from Node a

Current	Children
a	

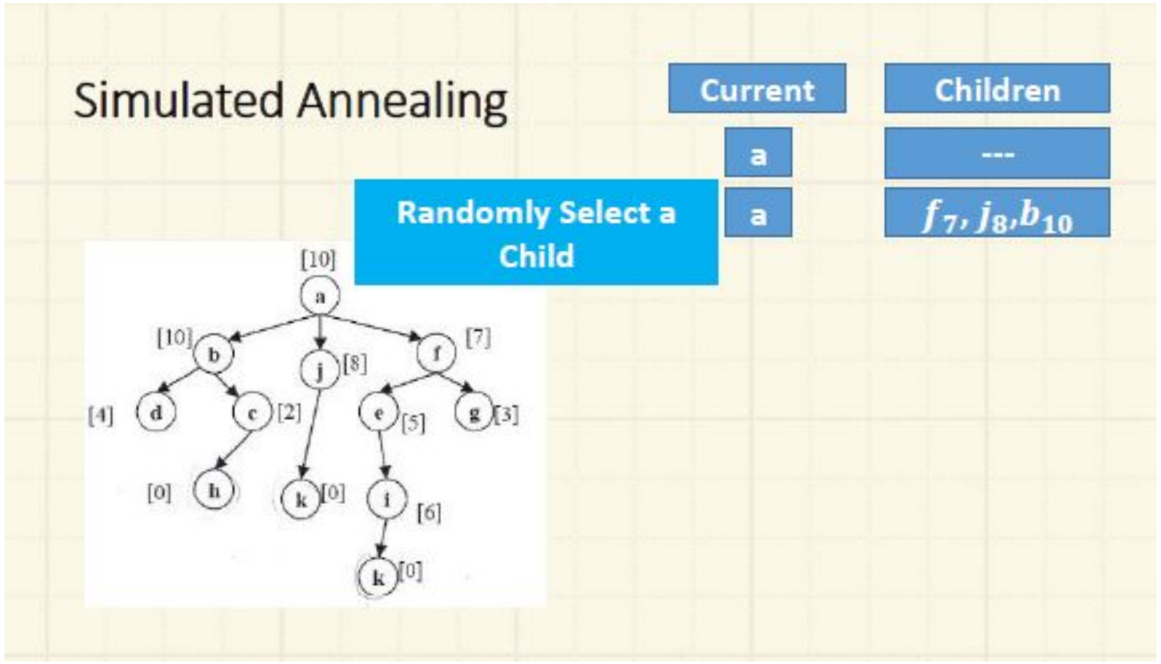
Simulated Annealing Problem

Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}



Simulated Annealing Problem

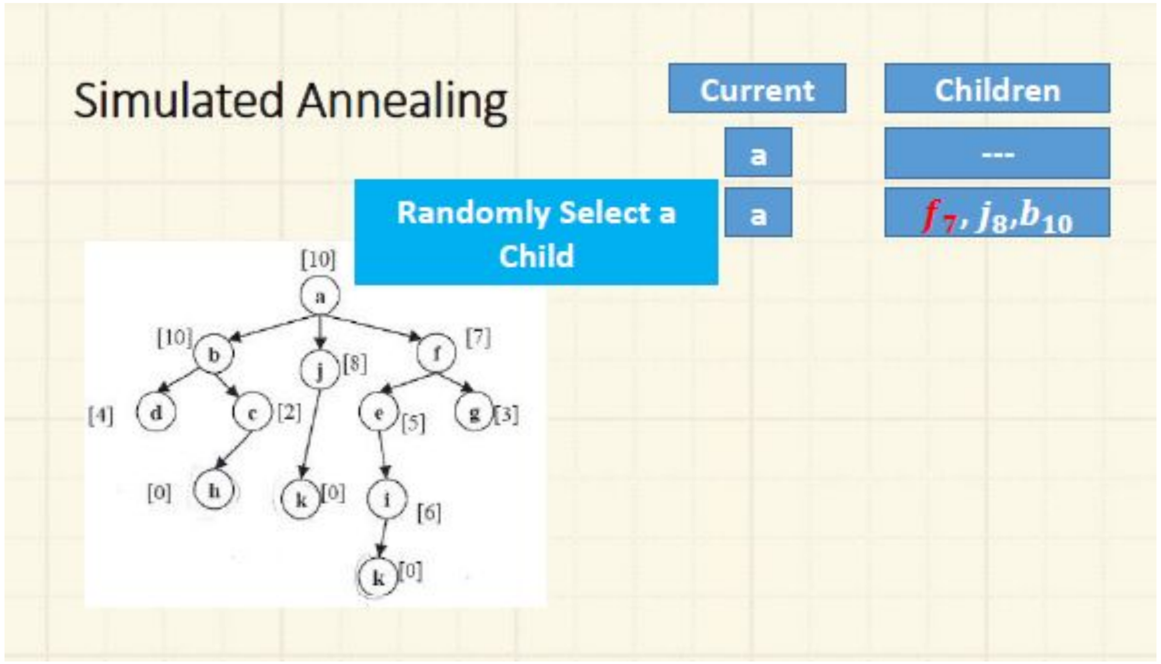


Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}

Randomly Select a Child

Simulated Annealing Problem



Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}

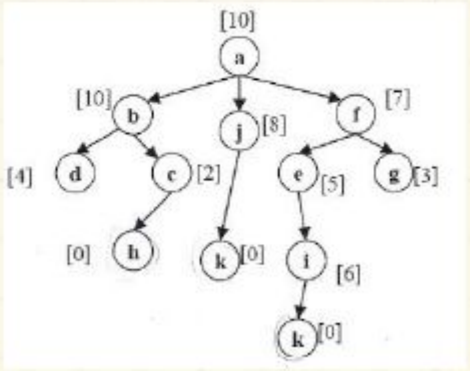
Randomly Select a Child

Simulated Annealing Problem

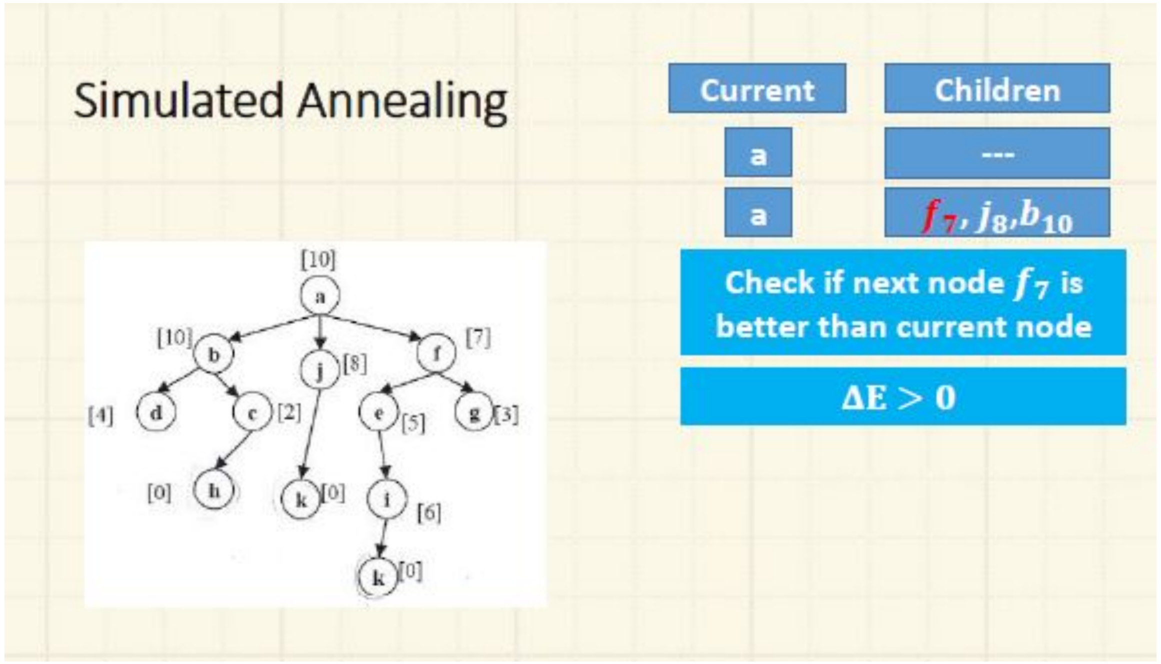
Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}

Check if next node f_7 is better than current node



Simulated Annealing Problem



Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}

Check if next node f_7 is better than current node

$\Delta E > 0$

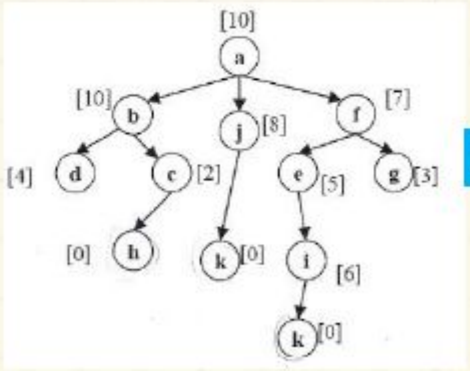
Simulated Annealing Problem

Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}

Check if next node f_7 is better than current node

$\Delta E = \text{value(next)} - \text{value(current)}$



Simulated Annealing Problem

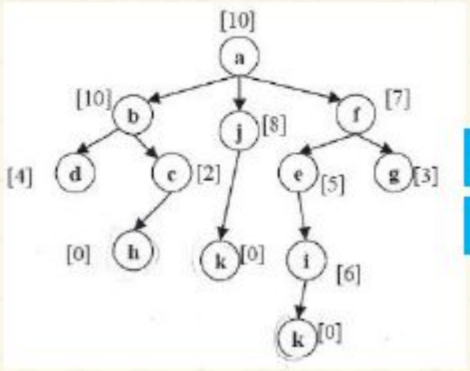
Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}

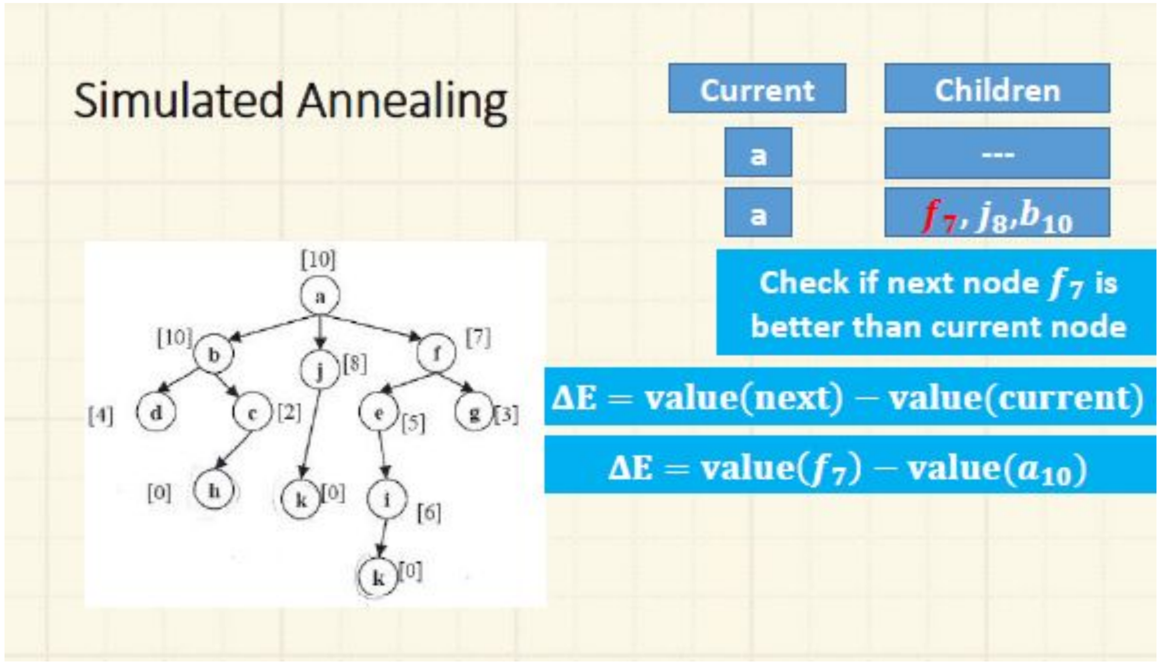
Check if next node f_7 is better than current node

$$\Delta E = \text{value(next)} - \text{value(current)}$$

$$\Delta E = \text{value}(f_7) - \text{value}(a_{10})$$



Simulated Annealing Problem



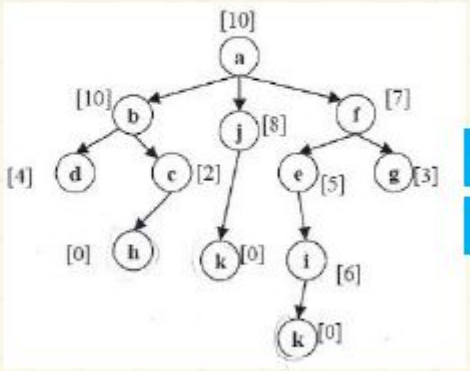
Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}

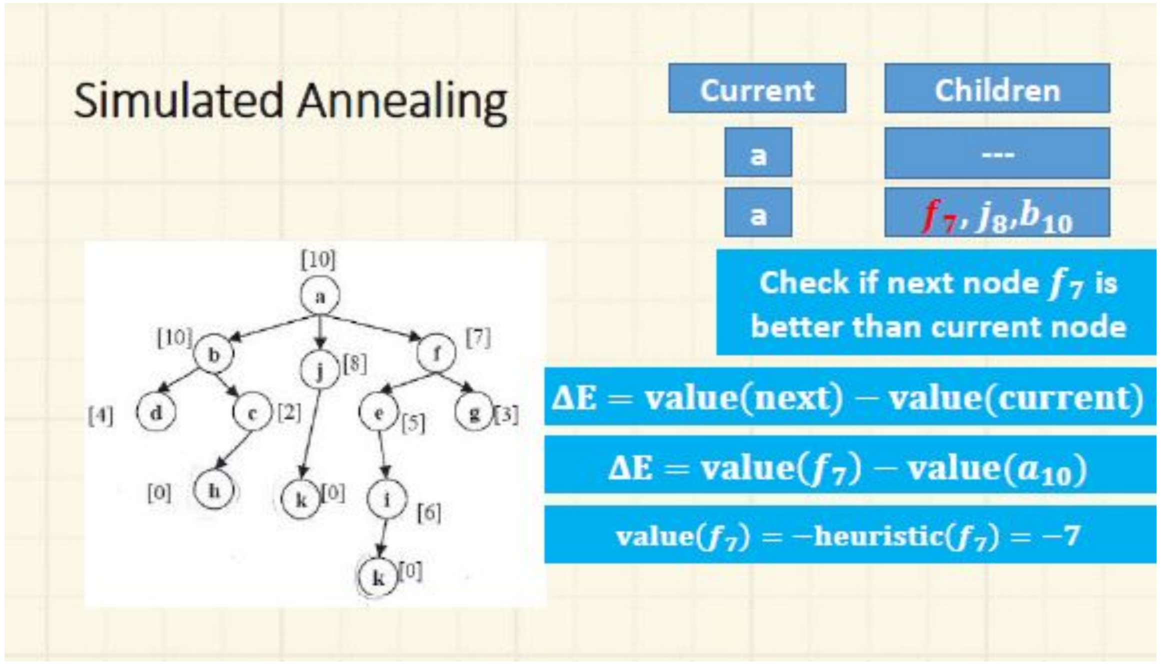
Check if next node f_7 is better than current node

$\Delta E = \text{value(next)} - \text{value(current)}$

$\Delta E = \text{value}(f_7) - \text{value}(a_{10})$



Simulated Annealing Problem



Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}

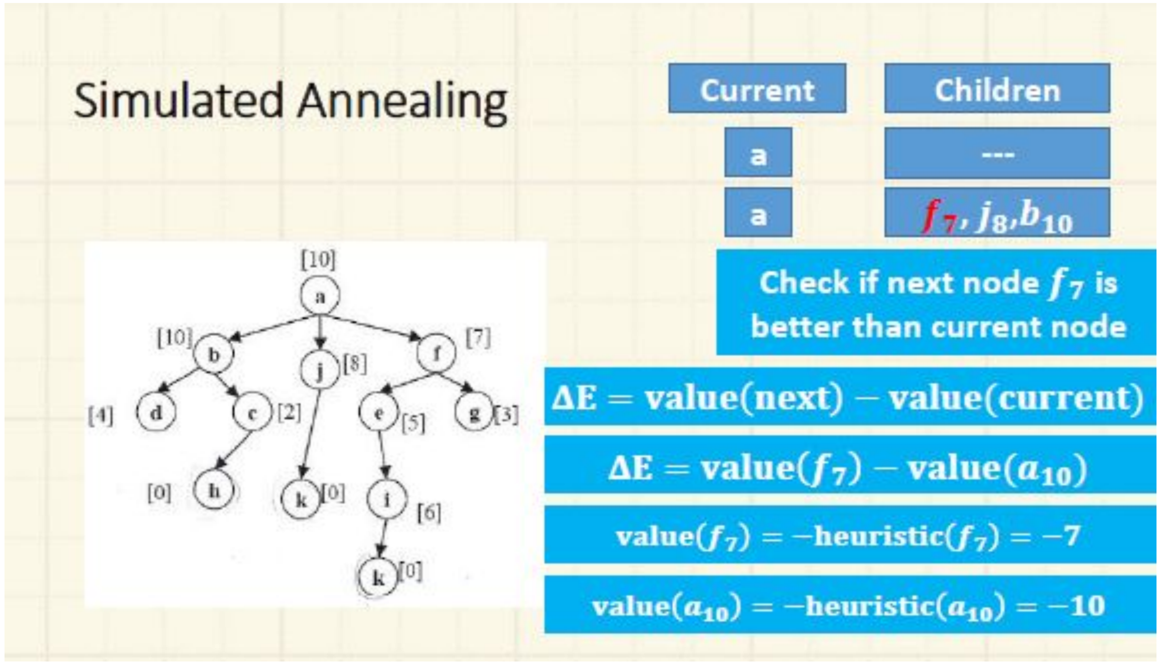
Check if next node f_7 is better than current node

$\Delta E = \text{value(next)} - \text{value(current)}$

$\Delta E = \text{value}(f_7) - \text{value}(a_{10})$

$\text{value}(f_7) = -\text{heuristic}(f_7) = -7$

Simulated Annealing Problem



Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}

Check if next node f_7 is better than current node

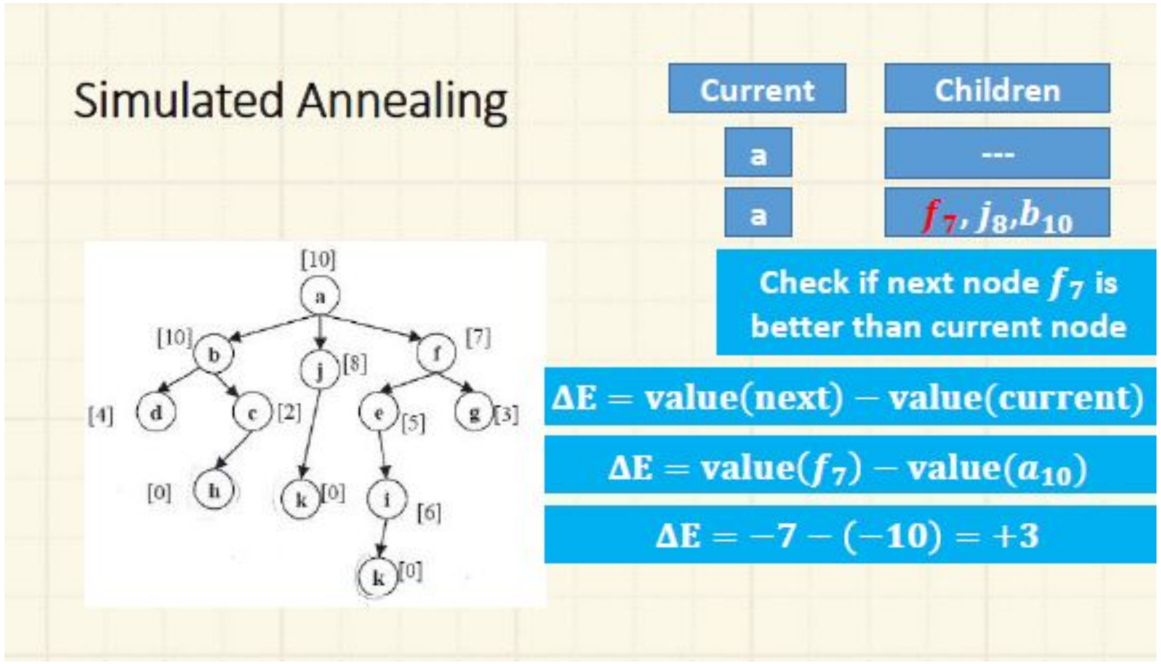
$\Delta E = \text{value}(\text{next}) - \text{value}(\text{current})$

$\Delta E = \text{value}(f_7) - \text{value}(a_{10})$

$\text{value}(f_7) = -\text{heuristic}(f_7) = -7$

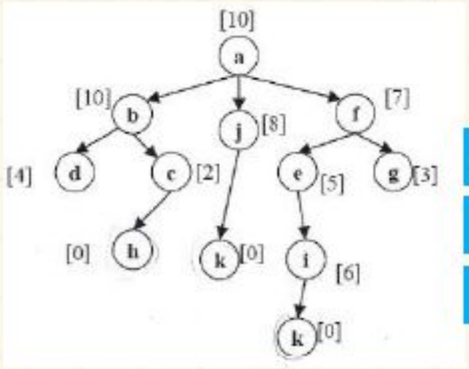
$\text{value}(a_{10}) = -\text{heuristic}(a_{10}) = -10$

Simulated Annealing Problem



Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}



Check if next node f_7 is better than current node

$$\Delta E = \text{value(next)} - \text{value(current)}$$

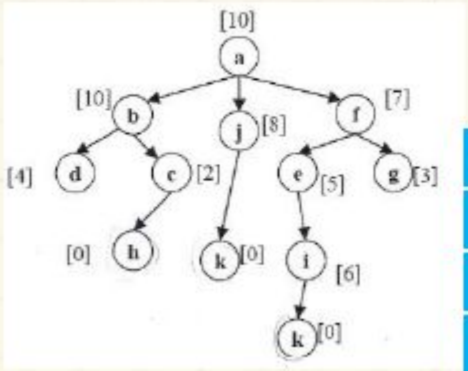
$$\Delta E = \text{value}(f_7) - \text{value}(a_{10})$$

$$\Delta E = -7 - (-10) = +3$$

Simulated Annealing Problem

Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}



Check if next node f_7 is better than current node

$$\Delta E = \text{value(next)} - \text{value(current)}$$

$$\Delta E = \text{value}(f_7) - \text{value}(a_{10})$$

$$\Delta E = -7 - (-10) = +3$$

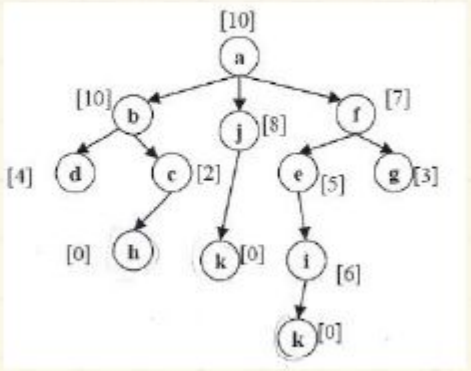
$$\because \Delta E > 0$$

$\therefore f_7$ will be selected with probability 1

Simulated Annealing Problem

Simulated Annealing

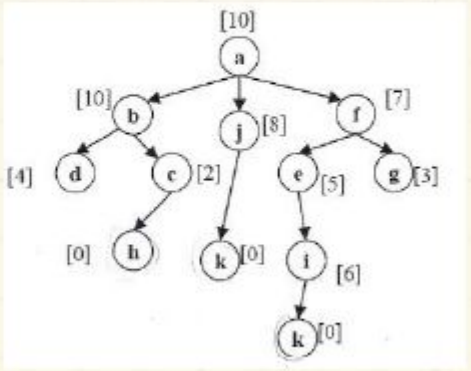
Current	Children
a	---
a	f_7, j_8, b_{10}
f	



Simulated Annealing Problem

Simulated Annealing

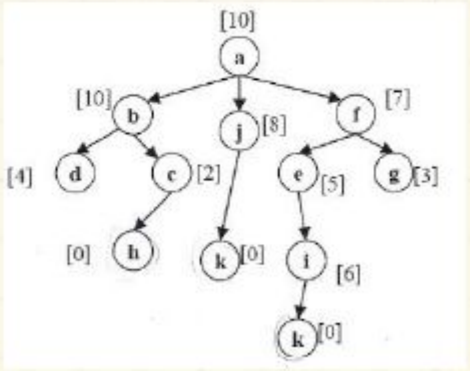
Current	Children
a	---
a	f_7, j_8, b_{10}
f	



Simulated Annealing Problem

Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3

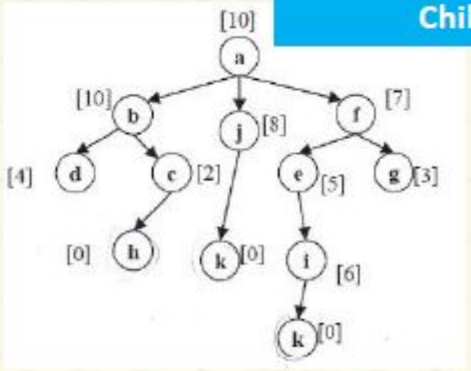


Simulated Annealing Problem

Simulated Annealing

Randomly Select a Child

Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3

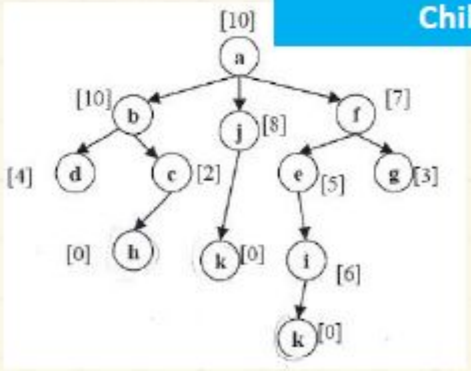


Simulated Annealing Problem

Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3

Randomly Select a Child

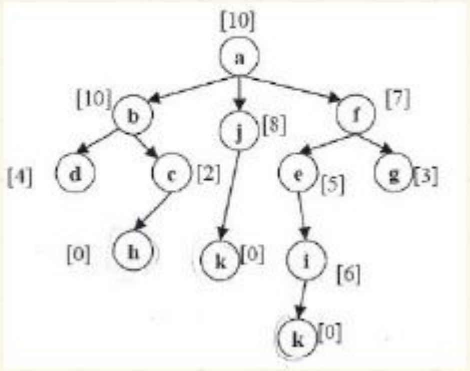


Simulated Annealing Problem

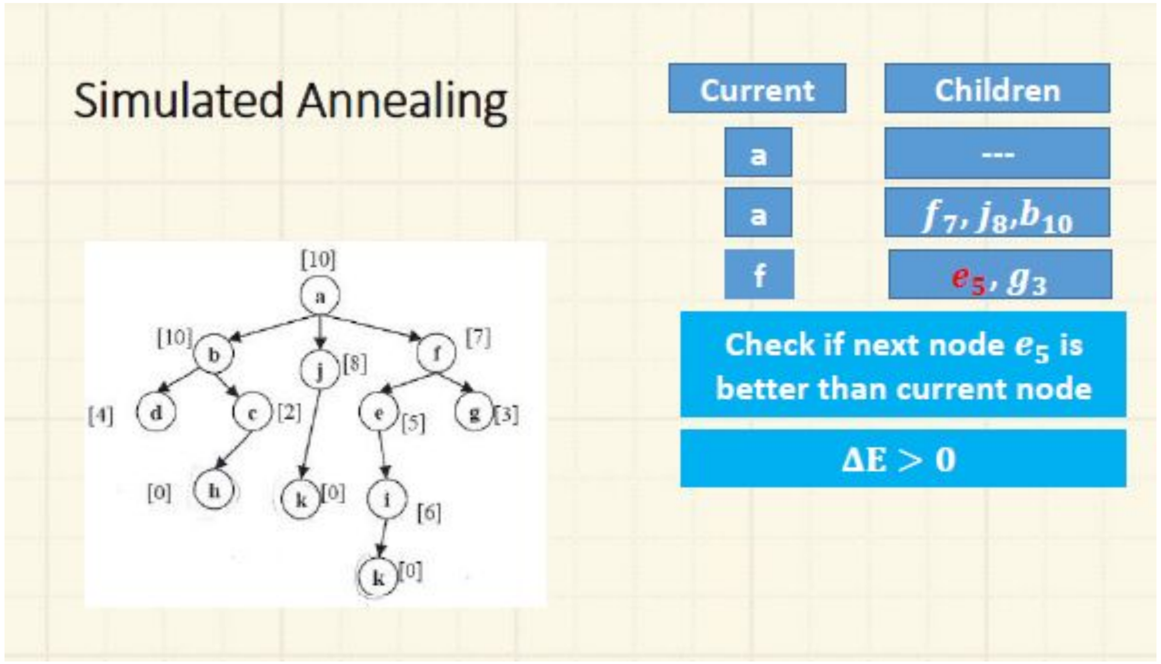
Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3

Check if next node e_5 is better than current node



Simulated Annealing Problem



Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3

Check if next node e_5 is better than current node

$\Delta E > 0$

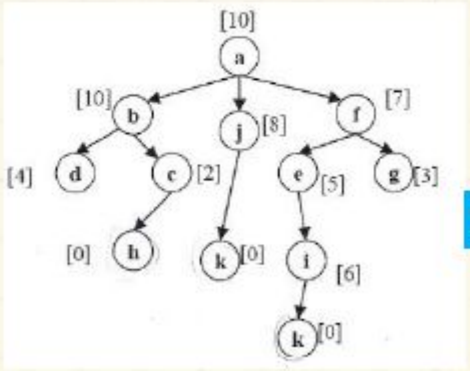
Simulated Annealing Problem

Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3

Check if next node e_5 is better than current node

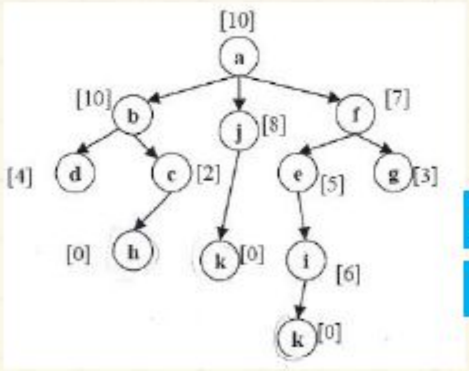
$\Delta E = \text{value(next)} - \text{value(current)}$



Simulated Annealing Problem

Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3



Check if next node e_5 is better than current node

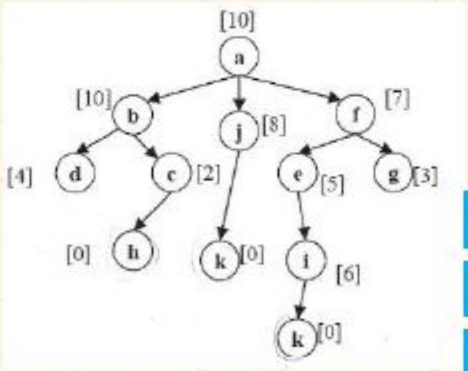
$$\Delta E = \text{value(next)} - \text{value(current)}$$

$$\Delta E = \text{value}(e_5) - \text{value}(f_7)$$

Simulated Annealing Problem

Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3



Check if next node e_5 is better than current node

$$\Delta E = \text{value(next)} - \text{value(current)}$$

$$\Delta E = \text{value}(e_5) - \text{value}(f_7)$$

$$\text{value}(e_5) = -\text{heuristic}(e_5) = -5$$

Simulated Annealing Problem

Simulated Annealing

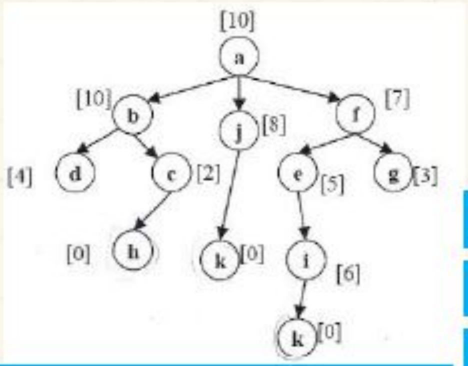
Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3

Check if next node e_5 is better than current node

$\Delta E = \text{value(next)} - \text{value(current)}$

$\Delta E = \text{value}(e_5) - \text{value}(f_7)$

$\text{value}(e_5) = -\text{heuristic}(e_5) = -5$

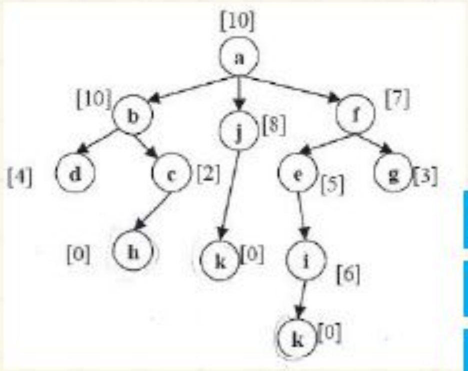


$\text{value}(f_7) = -\text{heuristic}(f_7) = -7$

Simulated Annealing Problem

Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3



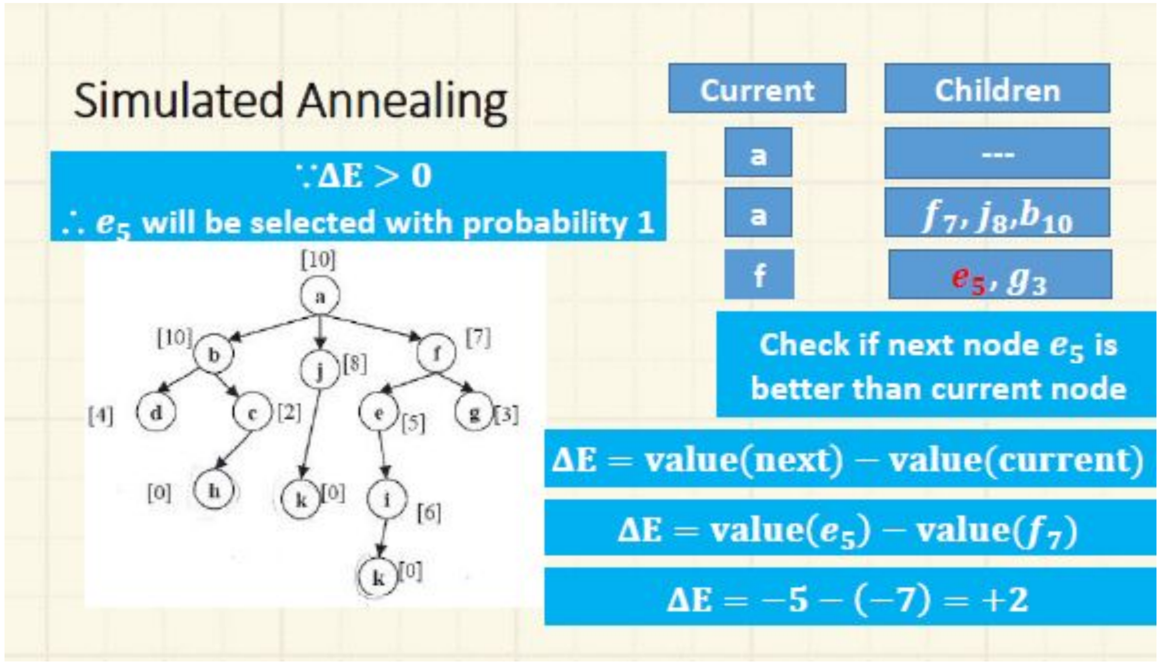
Check if next node e_5 is better than current node

$$\Delta E = \text{value(next)} - \text{value(current)}$$

$$\Delta E = \text{value}(e_5) - \text{value}(f_7)$$

$$\Delta E = -5 - (-7) = +2$$

Simulated Annealing Problem



Simulated Annealing

$\because \Delta E > 0$

$\therefore$ e_5 will be selected with probability 1

Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3

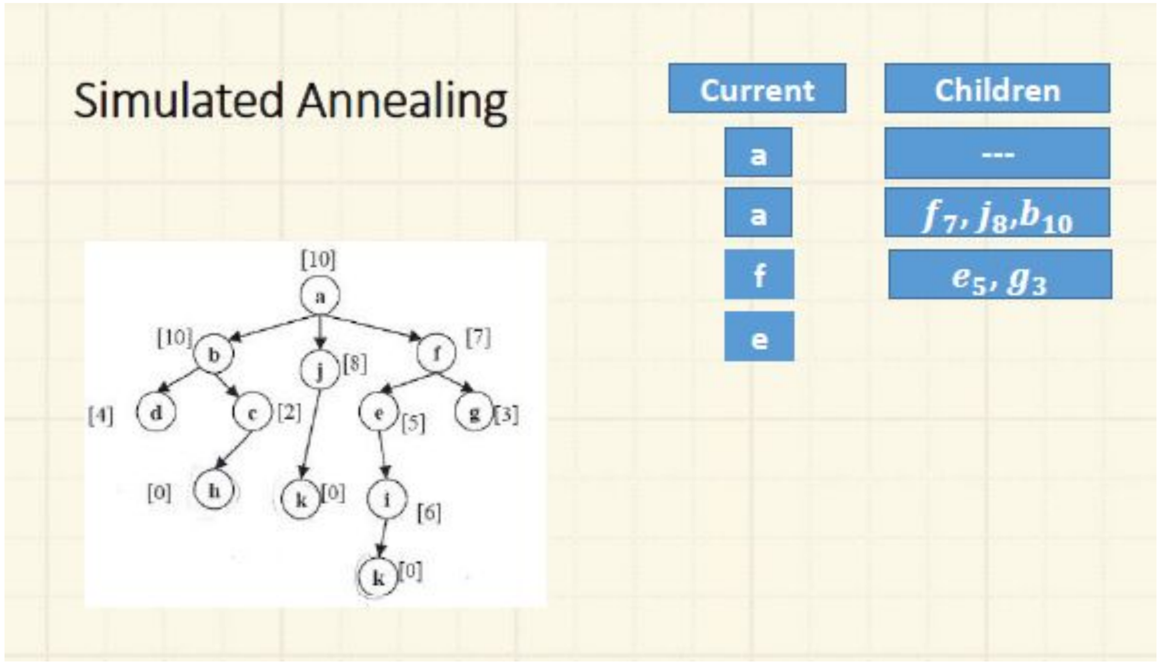
Check if next node e_5 is better than current node

$\Delta E = \text{value(next)} - \text{value(current)}$

$\Delta E = \text{value}(e_5) - \text{value}(f_7)$

$\Delta E = -5 - (-7) = +2$

Simulated Annealing Problem

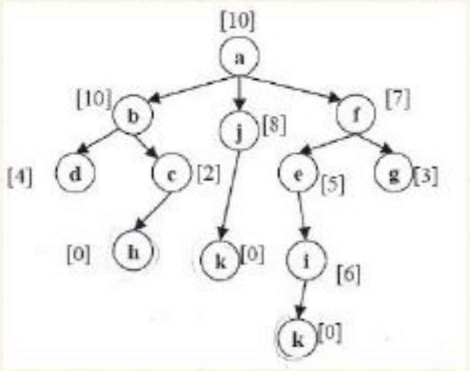


Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	

Simulated Annealing Problem

Simulated Annealing

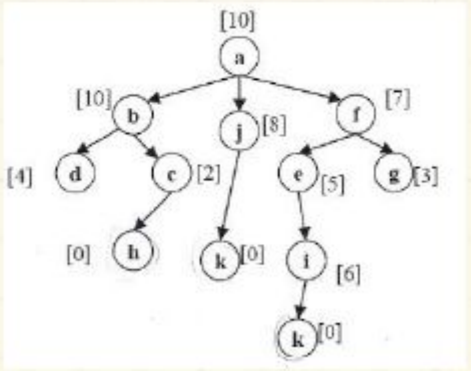


Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	

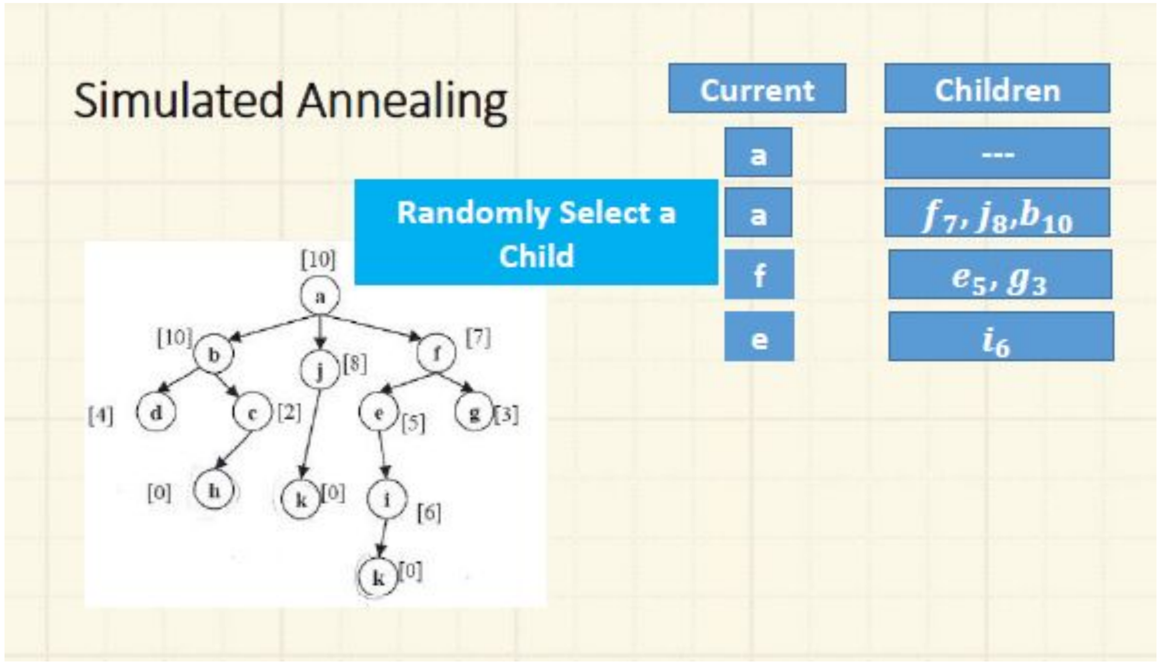
Simulated Annealing Problem

Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6



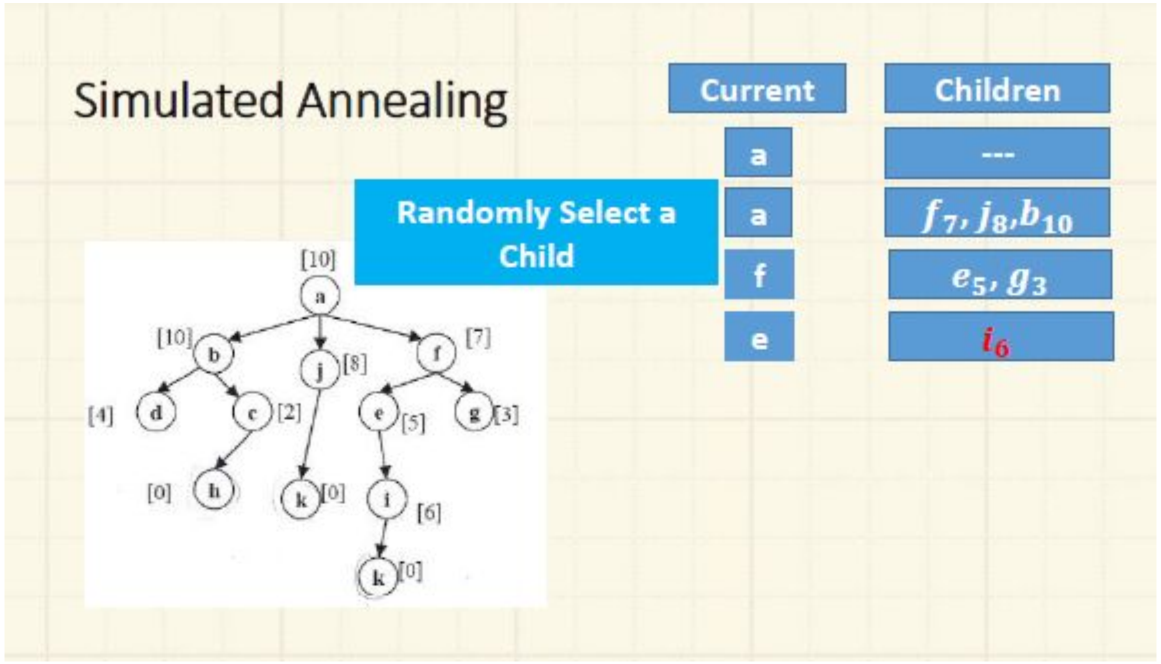
Simulated Annealing Problem


Simulated Annealing

Randomly Select a Child

Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6

Simulated Annealing Problem



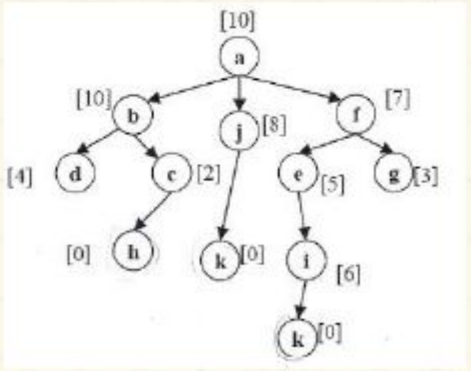
Simulated Annealing

Randomly Select a Child

Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6

Simulated Annealing Problem

Simulated Annealing

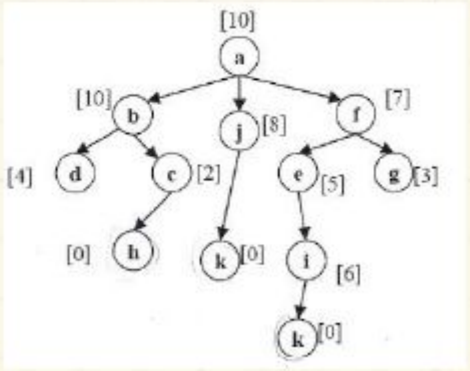


Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6

Check if next node i_6 is better than current node

Simulated Annealing Problem

Simulated Annealing

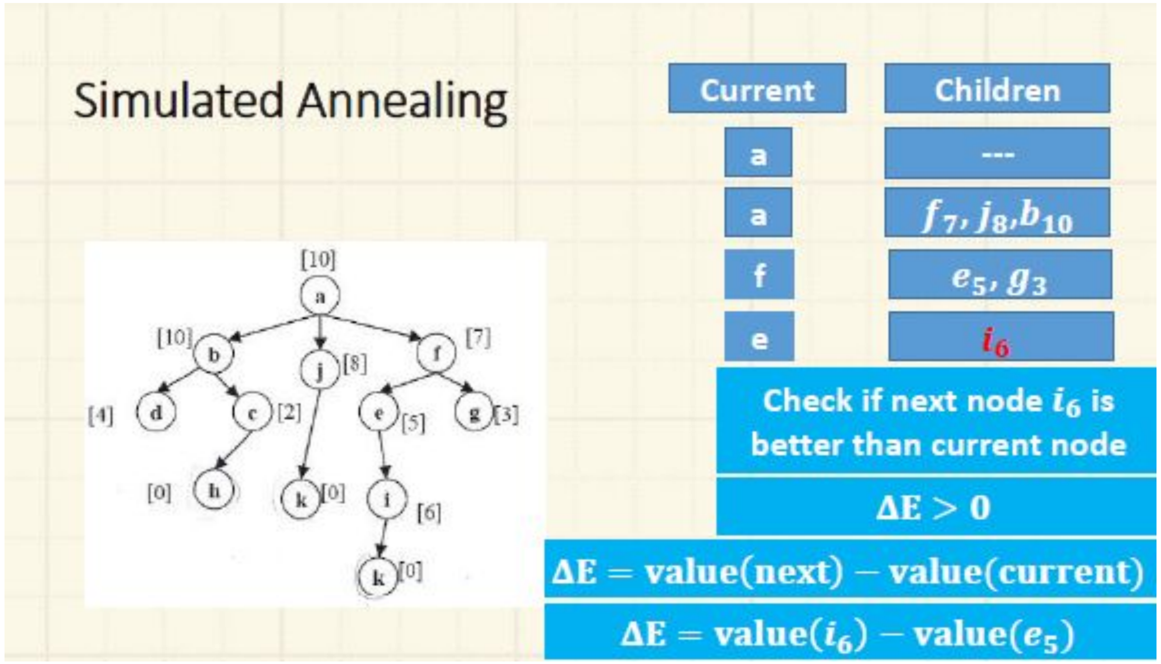


Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6

Check if next node i_6 is better than current node

$\Delta E > 0$

Simulated Annealing Problem



Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6

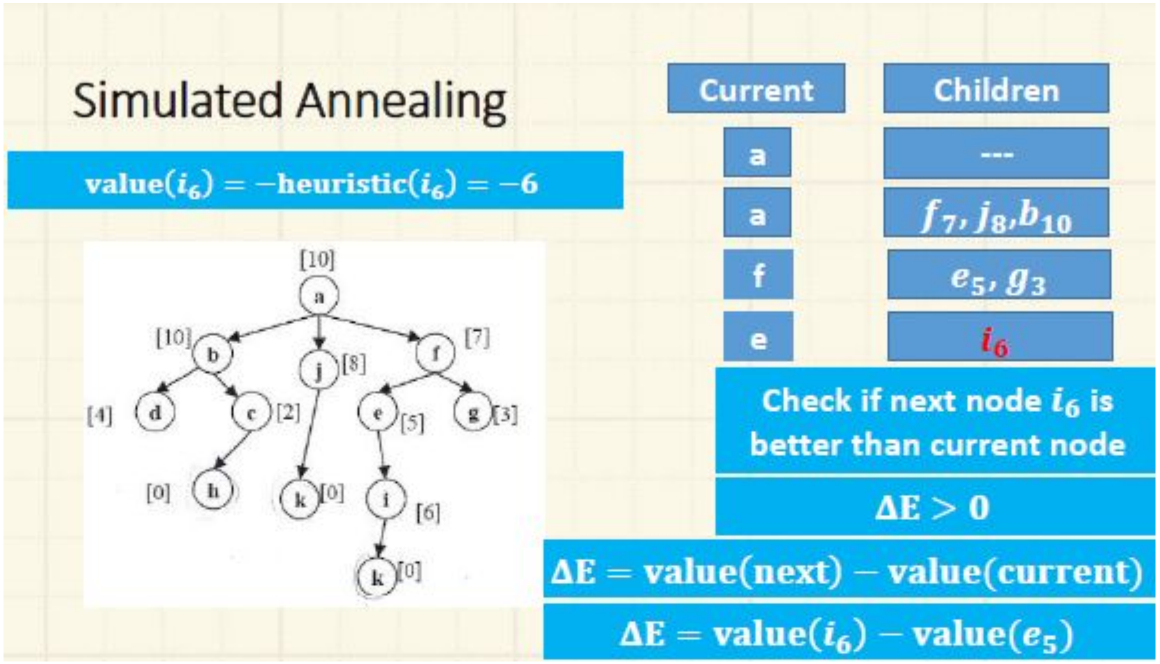
Check if next node i_6 is better than current node

$\Delta E > 0$

$\Delta E = \text{value(next)} - \text{value(current)}$

$\Delta E = \text{value}(i_6) - \text{value}(e_5)$

Simulated Annealing Problem



Simulated Annealing

$\text{value}(i_6) = -\text{heuristic}(i_6) = -6$

Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6

Check if next node i_6 is better than current node

$\Delta E > 0$

$\Delta E = \text{value(next)} - \text{value(current)}$

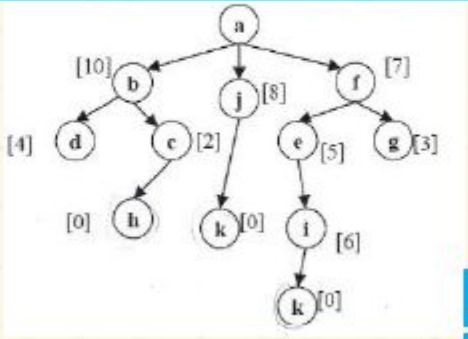
$\Delta E = \text{value}(i_6) - \text{value}(e_5)$

Simulated Annealing Problem

Simulated Annealing

$\textbf{value}(i_6) = -\textbf{heuristic}(i_6) = -6$

$\textbf{value}(e_5) = -\textbf{heuristic}(e_5) = -5$



Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6

Check if next node i_6 is better than current node

$\Delta E > 0$

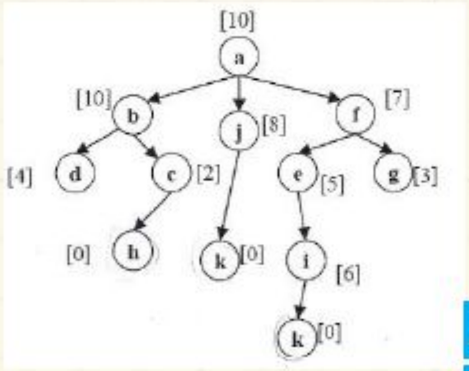
$\Delta E = \textbf{value}(\textbf{next}) - \textbf{value}(\textbf{current})$

$\Delta E = \textbf{value}(i_6) - \textbf{value}(e_5)$

Simulated Annealing Problem

Simulated Annealing

$\Delta E = -6 - (-5) = -1$



Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6

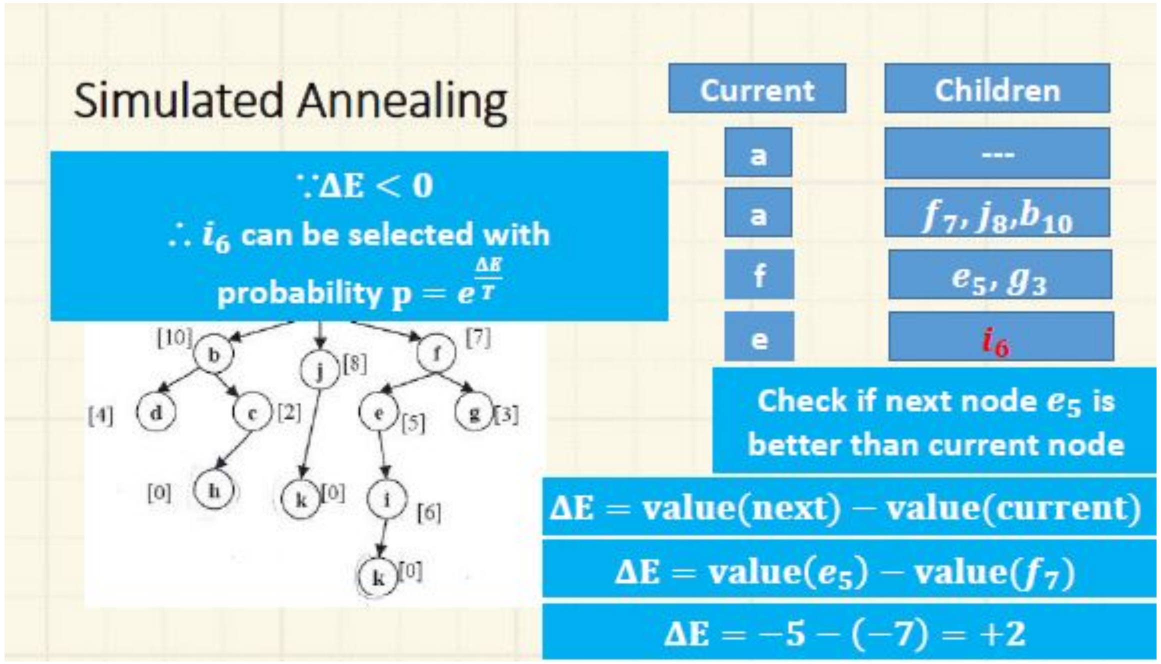
Check if next node i_6 is better than current node

$\Delta E > 0$

$\Delta E = \text{value(next)} - \text{value(current)}$

$\Delta E = \text{value}(i_6) - \text{value}(e_5)$

Simulated Annealing Problem



Simulated Annealing

$\because \Delta E < 0$

$\therefore$ i_6 can be selected with probability $p = e^{\frac{\Delta E}{T}}$

Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6

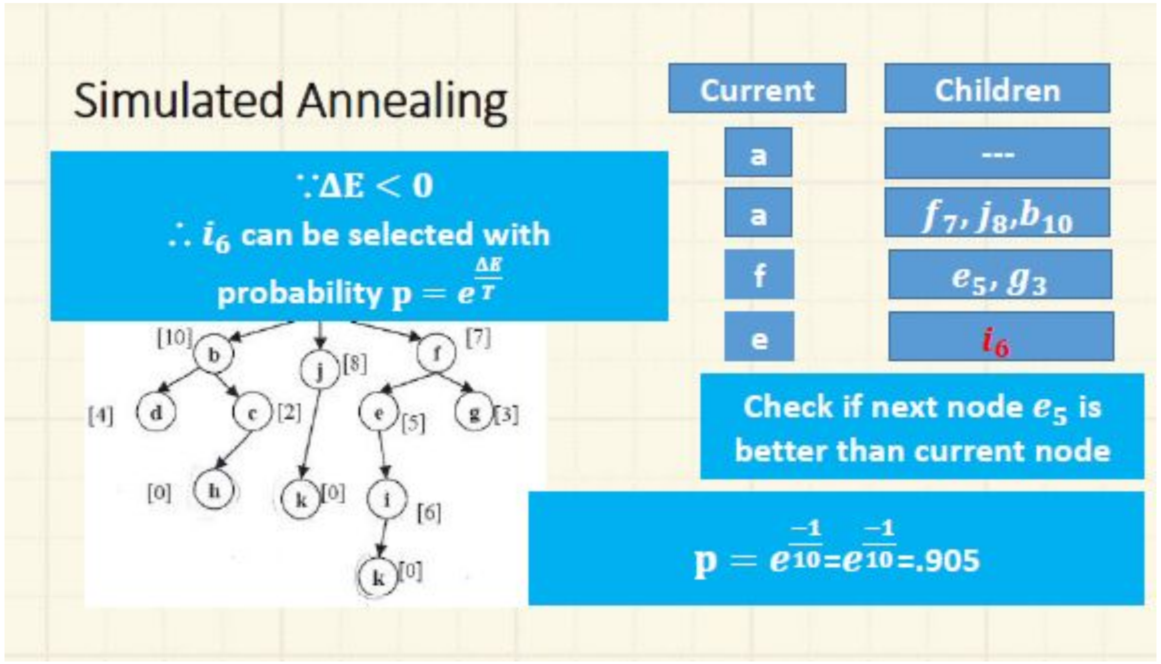
Check if next node e_5 is better than current node

$$\Delta E = \text{value(next)} - \text{value(current)}$$

$$\Delta E = \text{value}(e_5) - \text{value}(f_7)$$

$$\Delta E = -5 - (-7) = +2$$

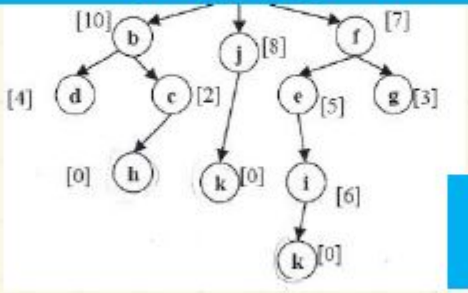
Simulated Annealing Problem



Simulated Annealing

$\because \Delta E < 0$

$\therefore i_6$ can be selected with probability $p = e^{\frac{\Delta E}{T}}$



Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6

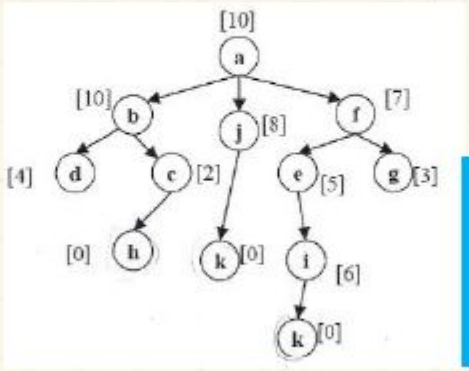
Check if next node e_5 is better than current node

$p = e^{\frac{-1}{10}} = e^{\frac{-1}{10}} = .905$

Simulated Annealing Problem

Simulated Annealing

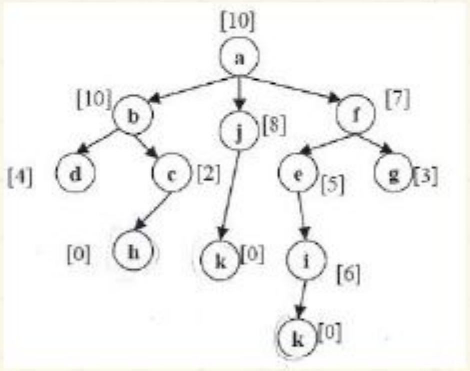
Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6



Because the only child of e_5 is i_6 then it will be selected even if its probability is not 1.

Simulated Annealing Problem

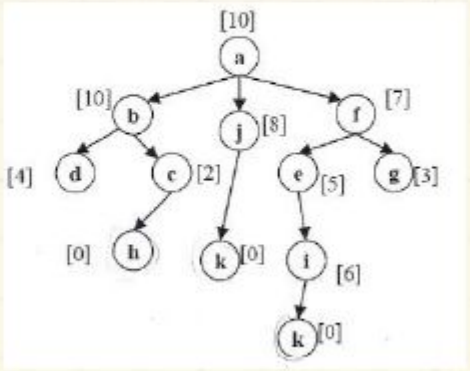
Simulated Annealing



Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6
i	

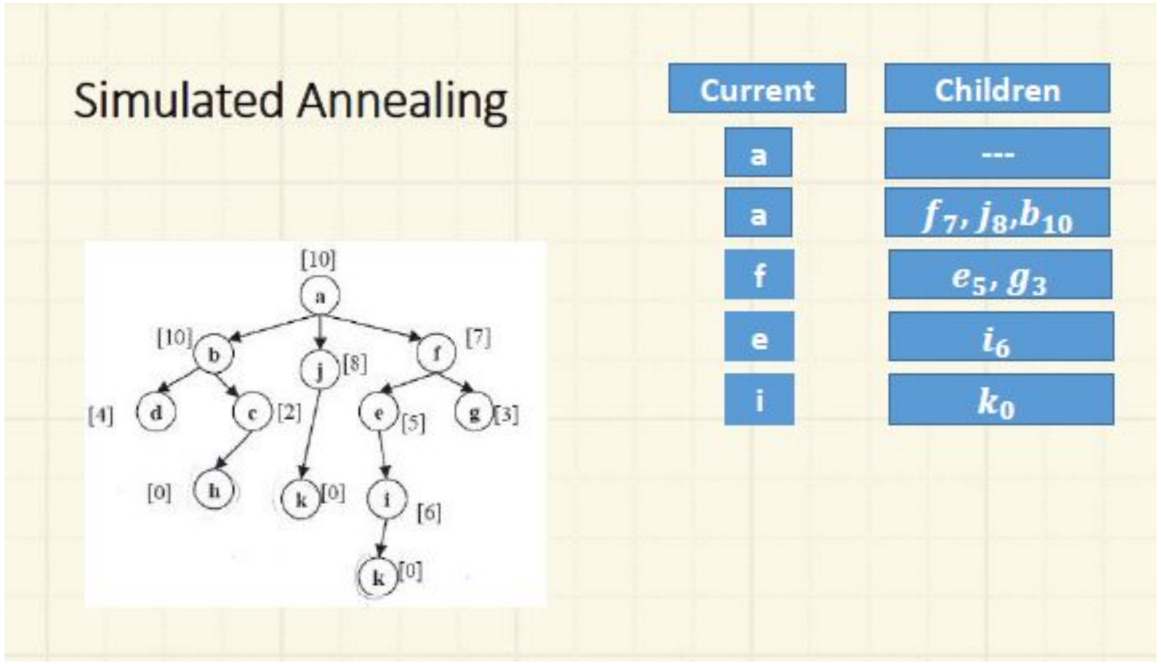
Simulated Annealing Problem

Simulated Annealing



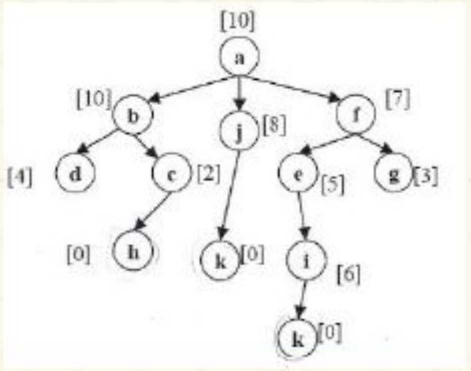
Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6
i	

Simulated Annealing Problem



Simulated Annealing

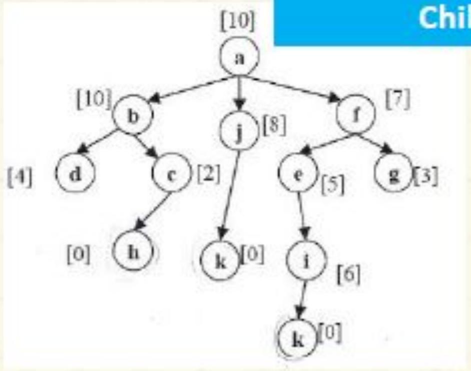
Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6
i	k_0



Simulated Annealing Problem

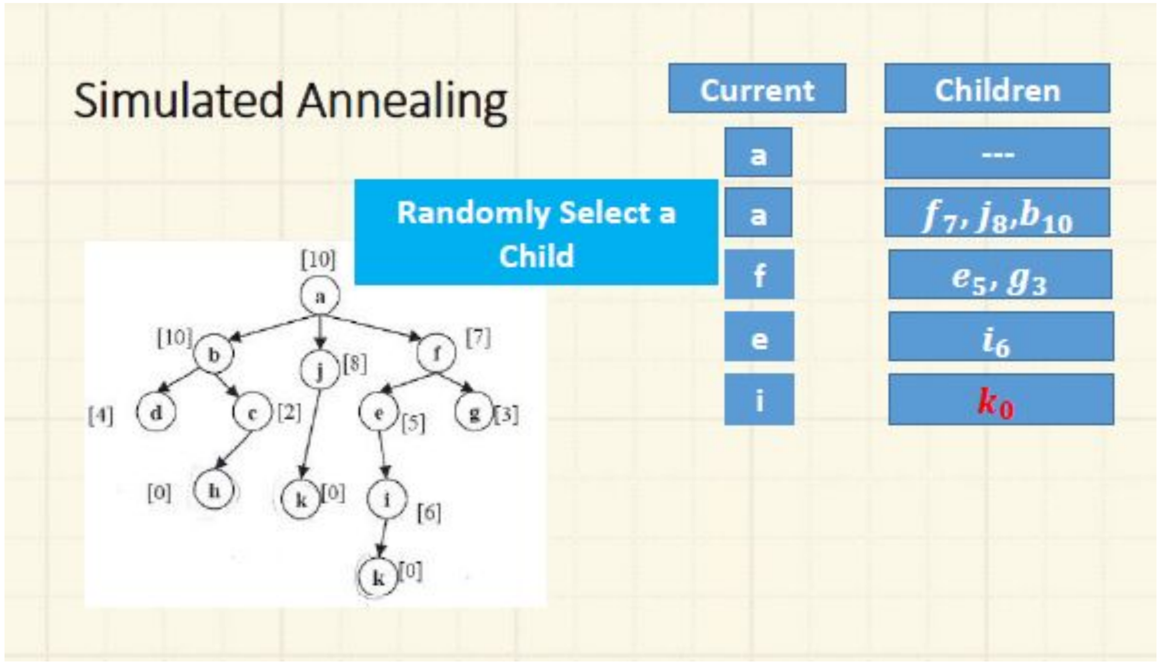
Simulated Annealing

Randomly Select a Child



Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6
i	k_0

Simulated Annealing Problem



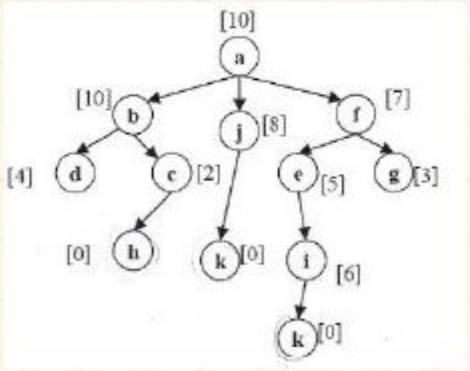
Simulated Annealing

Randomly Select a Child

Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6
i	k_0

Simulated Annealing Problem

Simulated Annealing

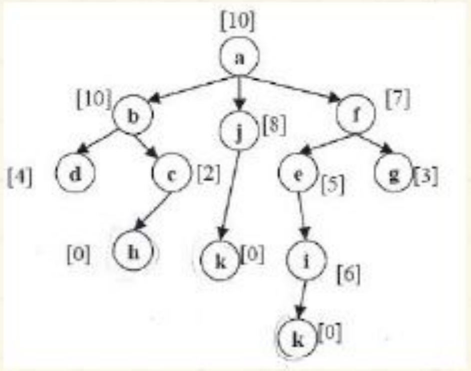


Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6
i	k_0

Check if next node k_0 is better than current node

Simulated Annealing Problem

Simulated Annealing



Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6
i	k_0

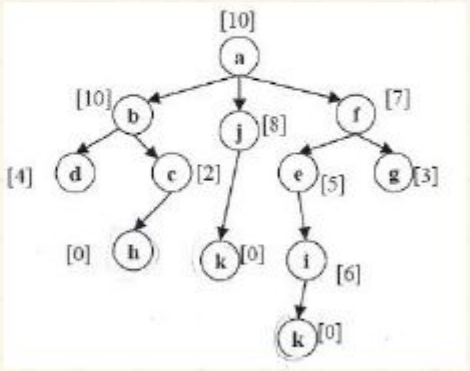
Check if next node k_0 is better than current node

$\Delta E > 0$

Simulated Annealing Problem

Simulated Annealing

$\Delta E = \text{value(next)} - \text{value(current)}$



Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6
i	k_0

Check if next node k_0 is better than current node

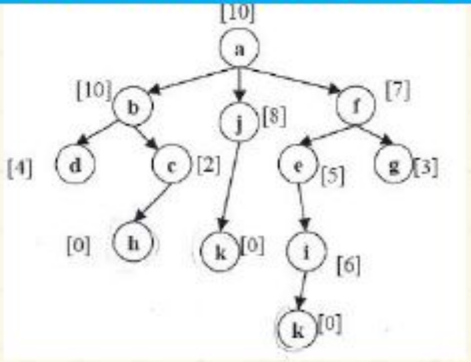
$\Delta E > 0$

Simulated Annealing Problem

Simulated Annealing

$\Delta E = \text{value(next)} - \text{value(current)}$

$\Delta E = \text{value}(k_0) - \text{value}(i_6)$

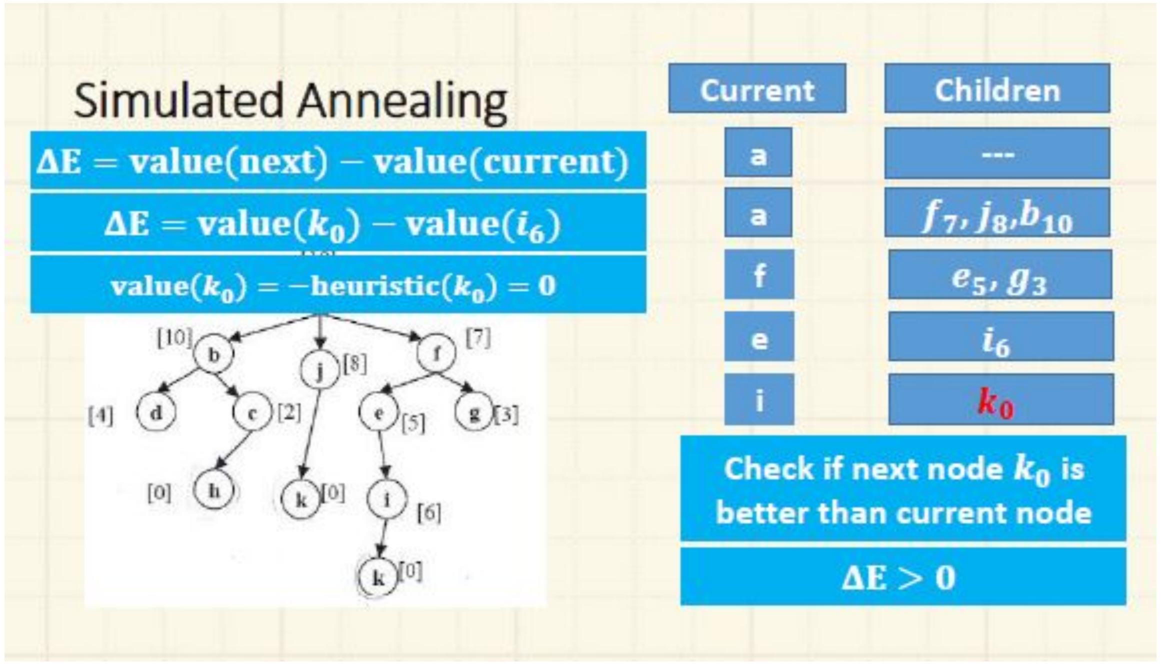


Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6
i	k_0

Check if next node k_0 is better than current node

$\Delta E > 0$

Simulated Annealing Problem



Simulated Annealing

$\Delta E = \text{value}(\text{next}) - \text{value}(\text{current})$

$\Delta E = \text{value}(k_0) - \text{value}(i_6)$

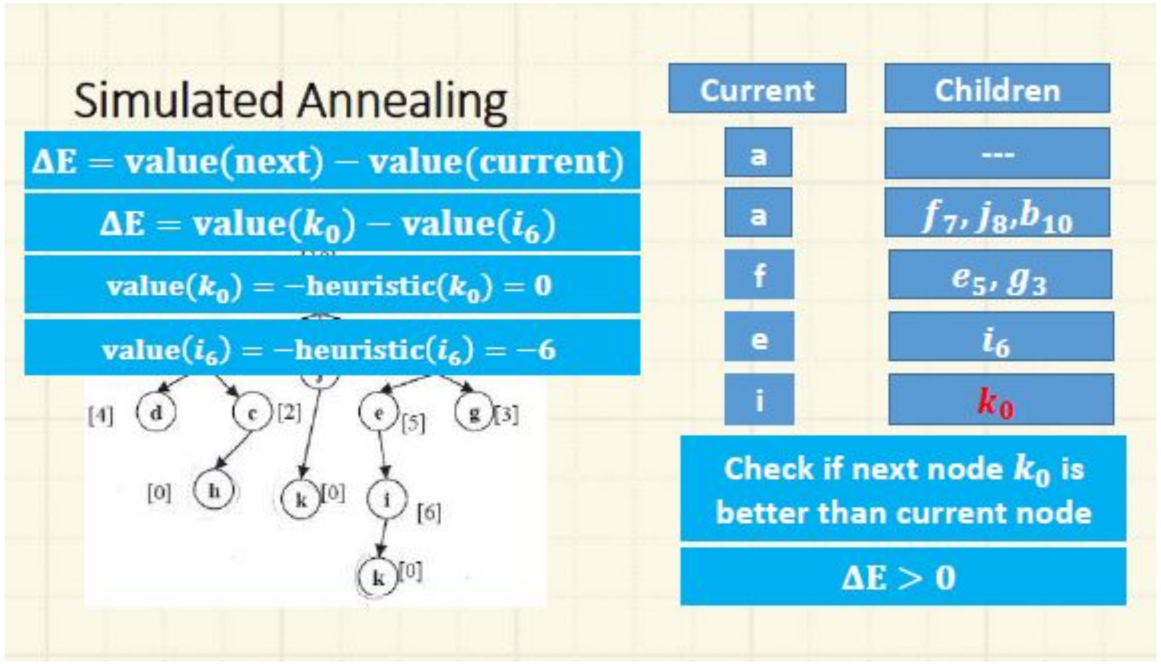
$\text{value}(k_0) = -\text{heuristic}(k_0) = 0$

Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6
i	k_0

Check if next node k_0 is better than current node

$\Delta E > 0$

Simulated Annealing Problem



Simulated Annealing

$\Delta E = \text{value(next)} - \text{value(current)}$

$\Delta E = \text{value}(k_0) - \text{value}(i_6)$

$\text{value}(k_0) = -\text{heuristic}(k_0) = 0$

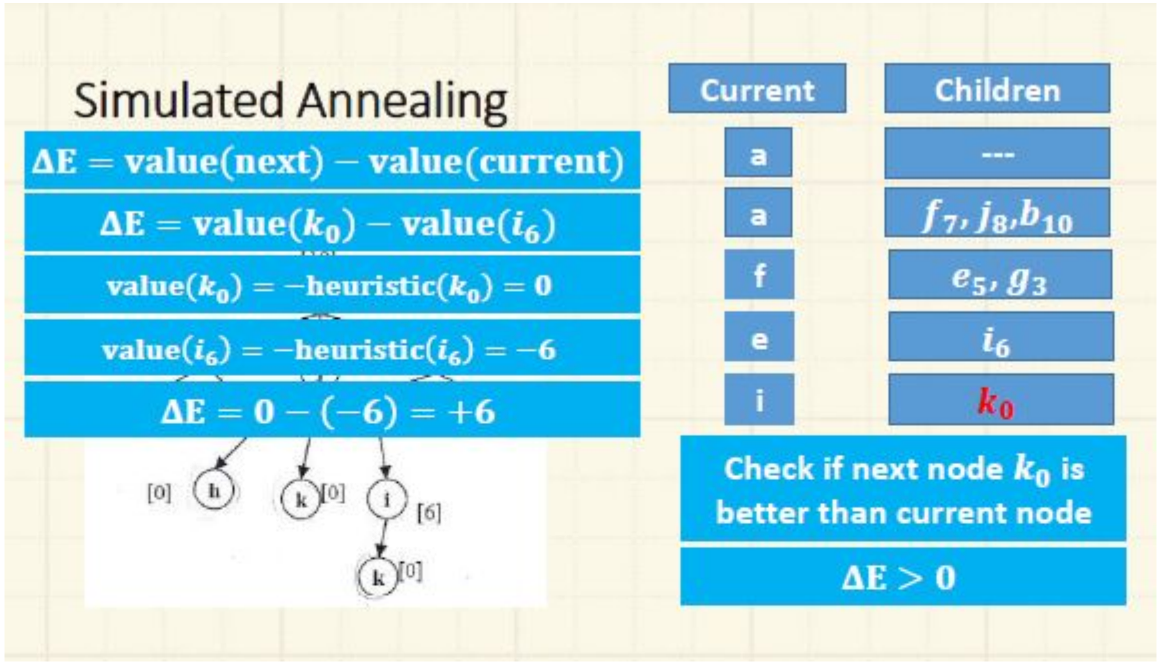
$\text{value}(i_6) = -\text{heuristic}(i_6) = -6$

Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6
i	k_0

Check if next node k_0 is better than current node

$\Delta E > 0$

Simulated Annealing Problem



Simulated Annealing

$\Delta E = \text{value(next)} - \text{value(current)}$

$\Delta E = \text{value}(k_0) - \text{value}(i_6)$

$\text{value}(k_0) = -\text{heuristic}(k_0) = 0$

$\text{value}(i_6) = -\text{heuristic}(i_6) = -6$

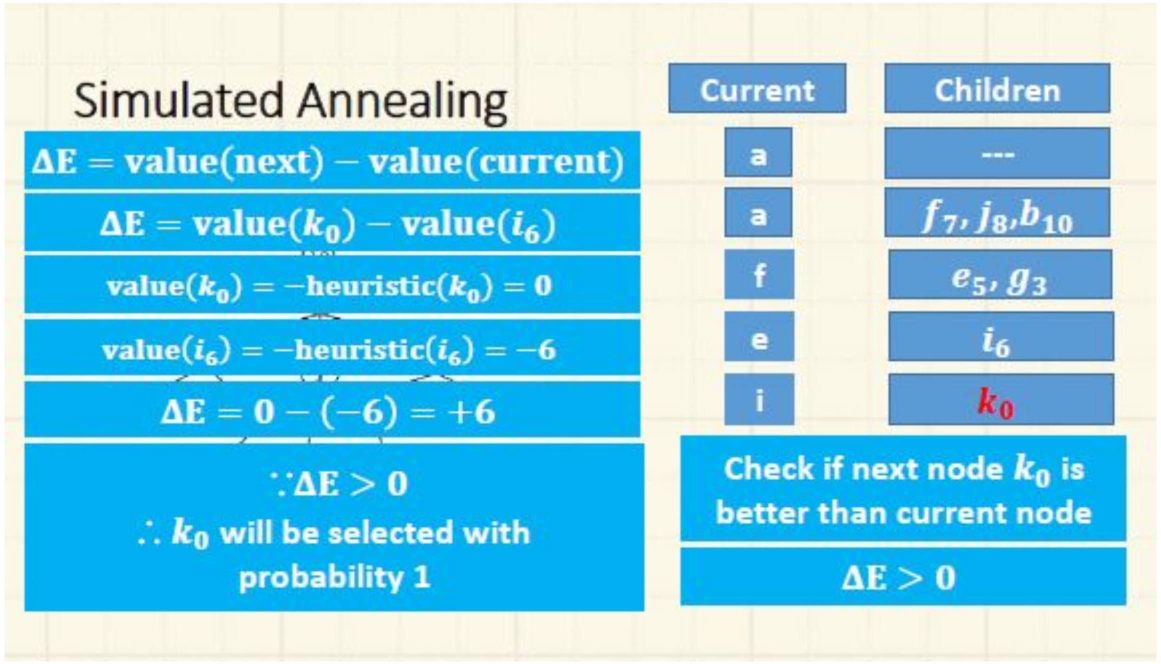
$\Delta E = 0 - (-6) = +6$

Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6
i	k_0

Check if next node k_0 is better than current node

$\Delta E > 0$

Simulated Annealing Problem



Simulated Annealing

$\Delta E = \text{value(next)} - \text{value(current)}$

$\Delta E = \text{value}(k_0) - \text{value}(i_6)$

$\text{value}(k_0) = -\text{heuristic}(k_0) = 0$

$\text{value}(i_6) = -\text{heuristic}(i_6) = -6$

$\Delta E = 0 - (-6) = +6$

$\because \Delta E > 0$

$\therefore k_0$ will be selected with probability 1

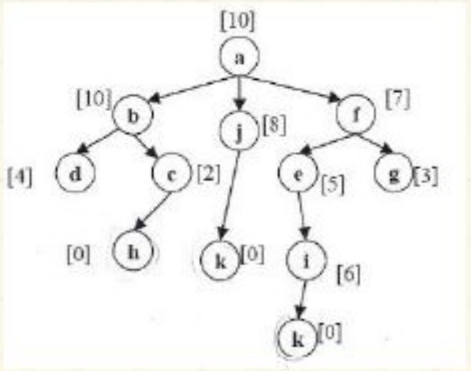
Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6
i	k_0

Check if next node k_0 is better than current node

$\Delta E > 0$

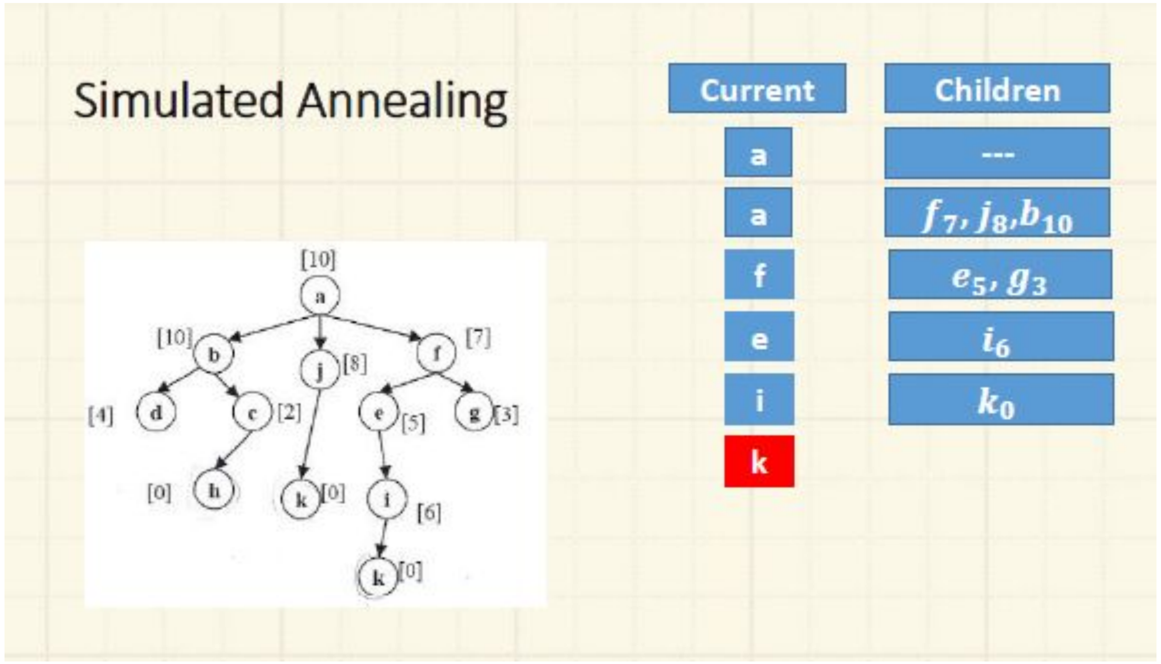
Simulated Annealing Problem

Simulated Annealing



Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6
i	k_0
k	

Simulated Annealing Problem

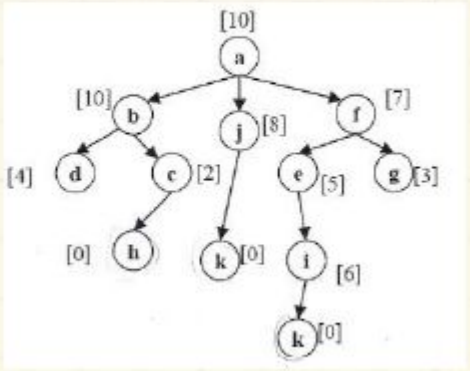


Simulated Annealing

Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6
i	k_0
k	

Simulated Annealing Problem

Simulated Annealing



Current	Children
a	---
a	f_7, j_8, b_{10}
f	e_5, g_3
e	i_6
i	k_0
k 🟥	

GOAL

Intuitions

- Hill-climbing is incomplete
- Pure random walk, keeping track of the best state found so far, is complete but very inefficient
- Combine the ideas: add some randomness to hill-climbing to allow the possibility of escape from a local optimum

Intuitions

- the algorithm wanders around during the early parts of the search, hopefully toward a good general region of the state space
- Toward the end, the algorithm does a more focused search, making few bad moves

Theoretical Completeness

- There is a proof that if the schedule lowers T slowly enough, simulated annealing will find a global optimum with probability approaching 1
- In practice, that may be way too many iterations
- In practice, though, SA can be effective at finding good solutions

Local Beam Search

- Keep track of k states rather than just one, as in hill climbing
- In comparison to beam search we saw earlier, this algorithm is state-based rather than node-based.

Local Beam Search

- Begins with k randomly generated states
- At each step, all successors of all k states are generated
- If any one is a goal, alg halts
- Otherwise, selects best k successors from the complete list, and repeats

Local Beam Search

- Successors can become concentrated in a small part of state space
- Stochastic beam search: choose k successors, with probability of choosing a given successor increasing with value
- Like natural selection: successors (offspring) of a state (organism) populate the next generation according to its value (fitness)

Genetic Algorithms (GA)

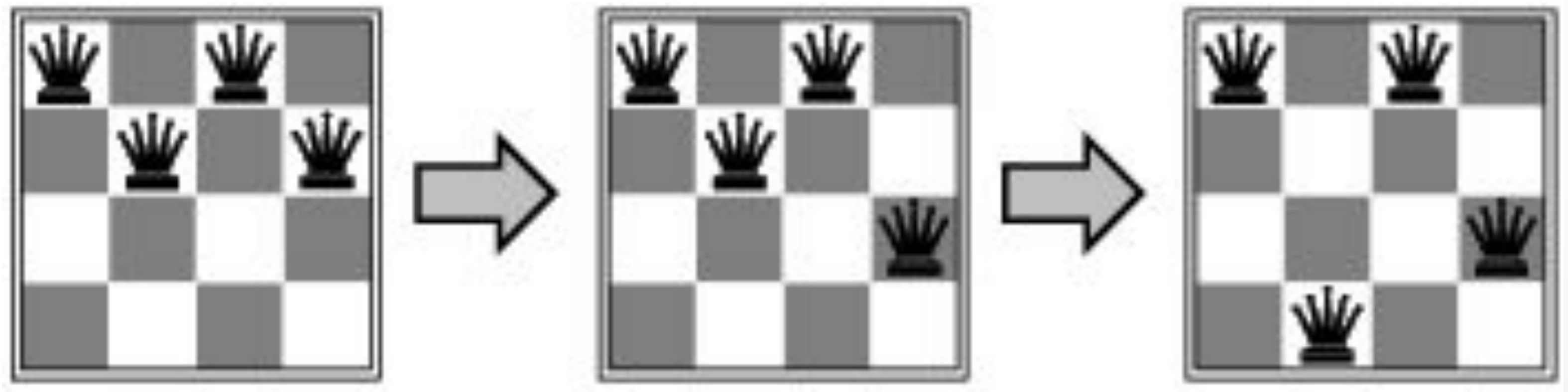
- Genetic Algorithms are the heuristic search and optimization techniques that mimic the process of natural evolution to find true or approximate solutions to optimization and search problems.
- Genetic algorithms implement the optimization strategies by simulating evolution of species through natural selection
- Based on “Survival of fittest

```
function GA (pop, fitness-fn)
Repeat
  new-pop = {}
  for i from 1 to size(pop):
    x = rand-sel(pop,fitness-fn)
    y = rand-sel(pop,fitness-fn)
    child = reproduce(x,y)
    if (small rand prob): child □mutate(child)
    add child to new-pop
  pop = new-pop
Until an indiv is fit enough, or out of time
Return best indiv in pop, according to fitness-fn
```

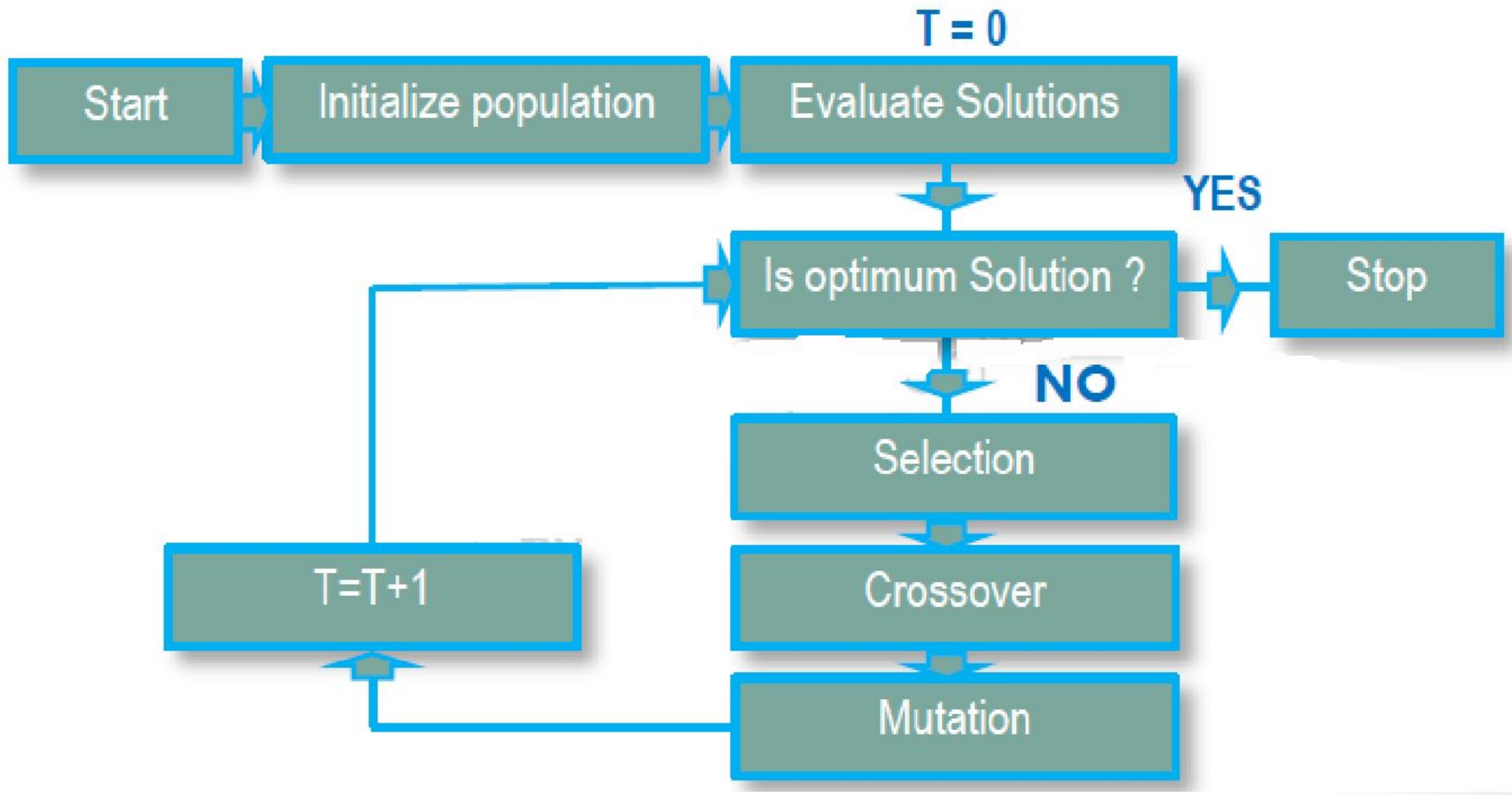
```
function reproduce(x,y)
   n = len(x)
   c = random num from 1 to n
   return:
     append(substr(x,1,c),substr(y,c+1,n)
```

Example: *n*-queens

- Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal



Genetic Algorithms



Genetic Algorithms

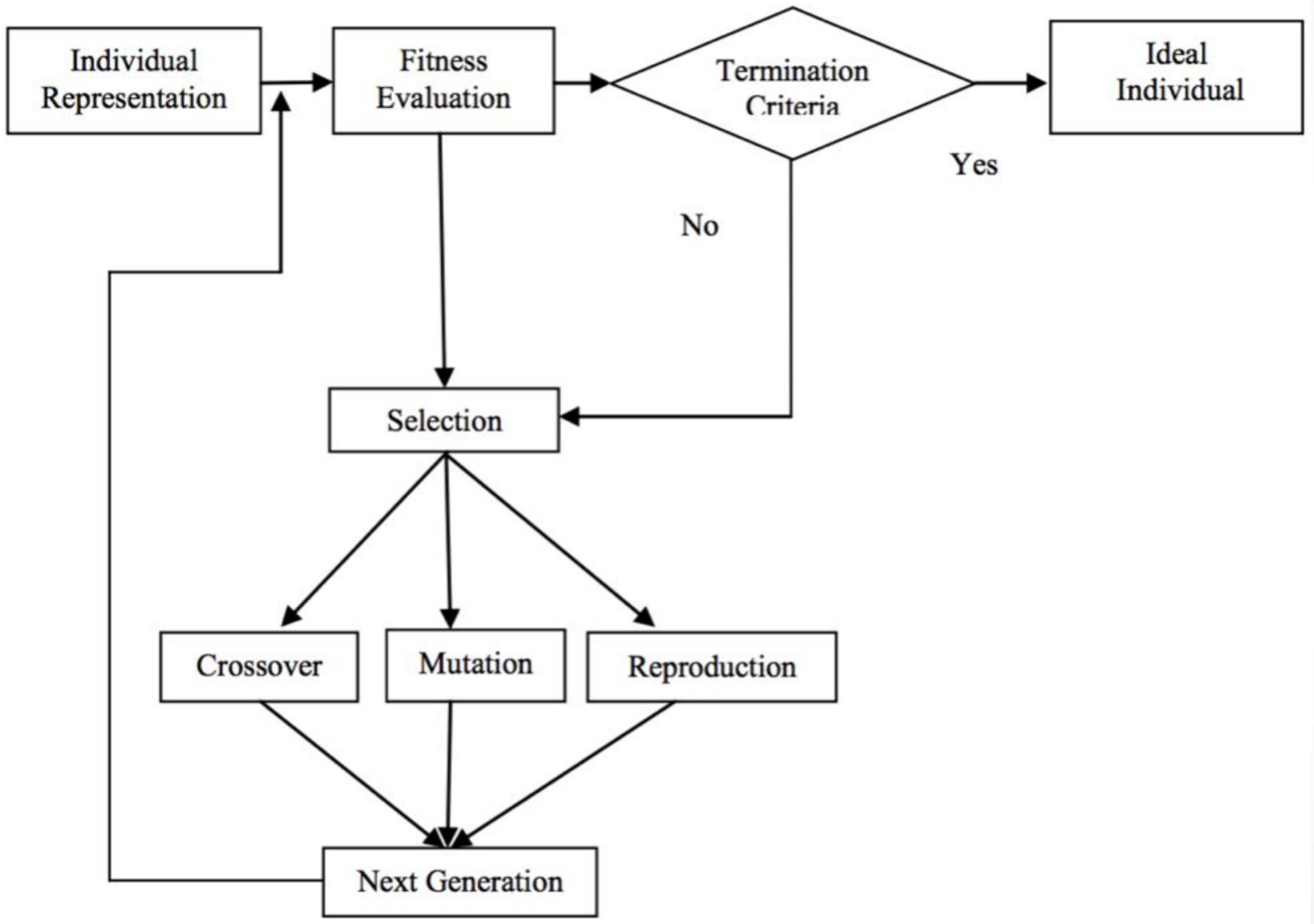
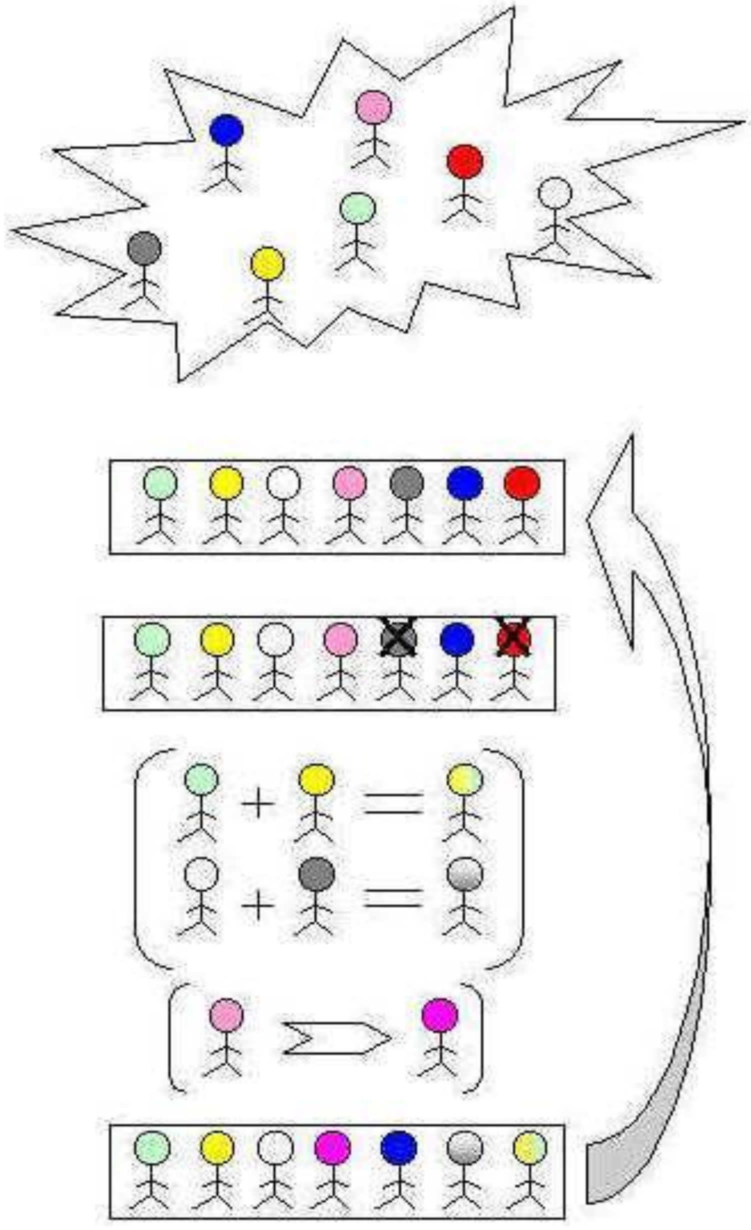
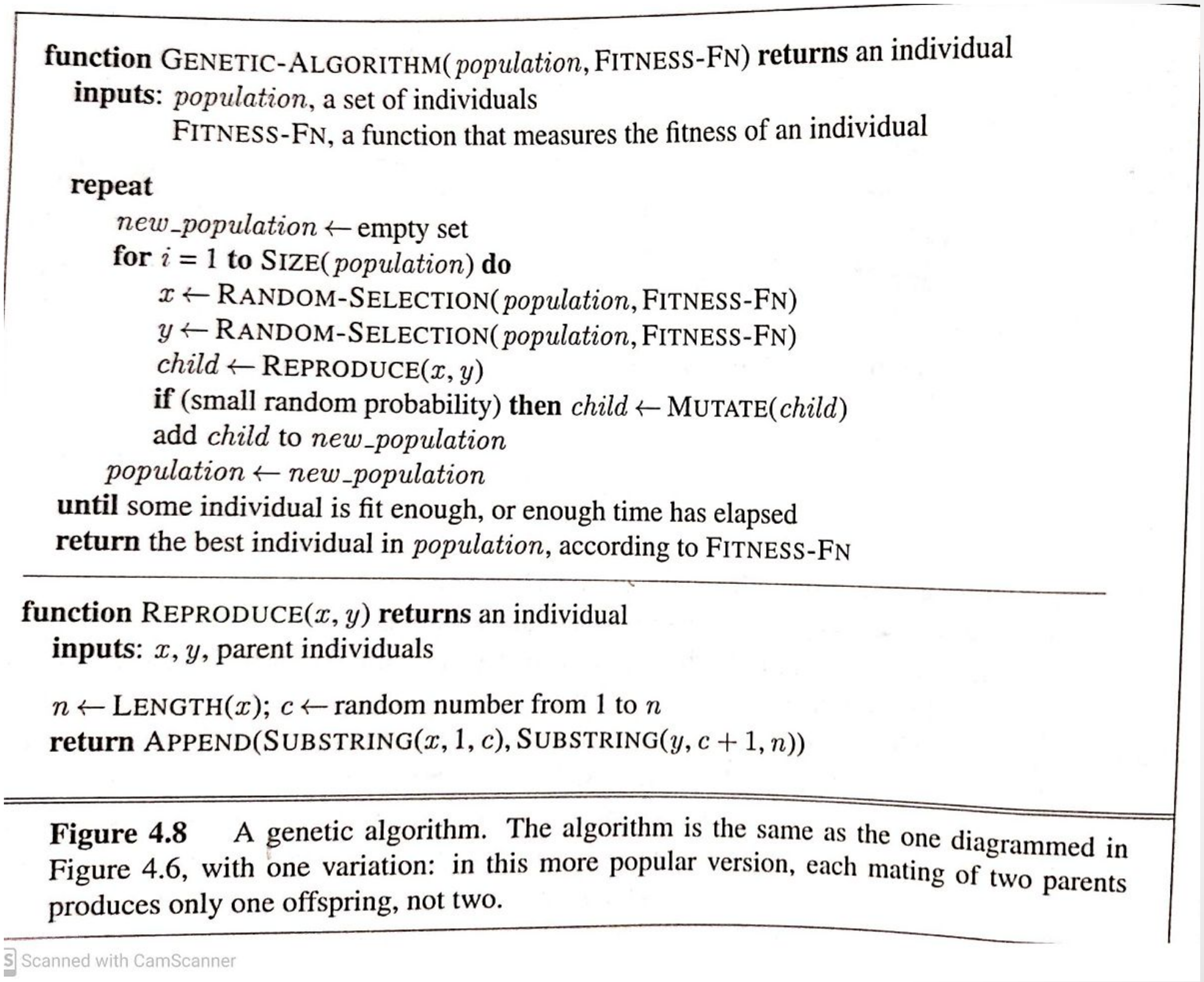


Fig : Genetic Algorithm Process

Genetic Algorithms



Genetic Algorithm



```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
     new_population ← empty set
     for $i$ = 1 to SIZE(population) do
        $x$ ← RANDOM-SELECTION(population, FITNESS-FN)
        $y$ ← RANDOM-SELECTION(population, FITNESS-FN)
        child ← REPRODUCE($x$, $y$)
        if (small random probability) then child ← MUTATE(child)
        add child to new_population
     population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN
```

```
function REPRODUCE($x$, $y$) returns an individual
  inputs: $x$, $y$, parent individuals

  $n$ ← LENGTH($x$); $c$ ← random number from 1 to $n$
  return APPEND(SUBSTRING($x$, 1, $c$), SUBSTRING($y$, $c+1$, $n$))
```

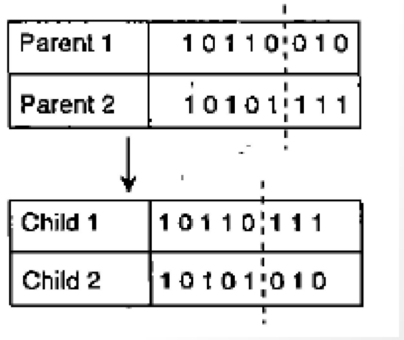
Figure 4.8 A genetic algorithm. The algorithm is the same as the one diagrammed in Figure 4.6, with one variation: in this more popular version, each mating of two parents produces only one offspring, not two.

Genetic Algorithms

- Representation of individuals
  - *Classic approach*: individual is a string with each element in the string called a *gene*
  - Usually binary (others: Octal, Hexadecimal, Permutation Encoding (Real Number Coding), Tree Encoding
- Selection strategy
  - Process of choosing two parents from the population for crossing
  - how to choose individuals in the population that will create offspring for the next generation and how many offspring each will create
  - method that randomly picks chromosomes out of the population according to their evaluation function.
  - The higher the fitness function, the better chance that an individual will be selected
  - Roulette-wheel selection, Tournament Selection, Elitist selection, Rank selection, Proportionate Selection, Steady state selection

Genetic Algorithms

- <u>Reproduction</u>
  - Random pairing of selected individuals
  - Random selection of *cross-over* points
    - Crossover -process of taking two parent solutions and producing from them a child.
    - After the selection (reproduction) process, the population is enriched with better individuals.
    - Reproduction makes clones of good strings but does not create new ones.
    - Crossover operator is applied to the mating pool with the hope that it creates a better offspring.
    - Single point crossover, Two-point crossover, Multi-point crossover, Uniform crossover, etc.


Parent 1 | 1 0 1 1 0 0 1 0
Parent 2 | 1 0 1 0 1 1 1 1
Child 1 | 1 0 1 1 0 1 1 1
Child 2 | 1 0 1 0 1 0 1 0

Genetic Algorithms

- Each gene can be altered by a random *mutation*
  - If crossover is supposed to exploit current solution to find better ones, mutation is supposed to help for the exploration of the whole search space.
  - Viewed as a background operator to maintain genetic diversity in the population.
  - Introduces new generic structures in the population by randomly modifying some of its building blocks
  - Approx 1 in 20

Genetic Algorithm-An Example

Consider the problem of maximizing the function,
$f(x) = x^2$
where x is permitted to vary between 0 and 31.

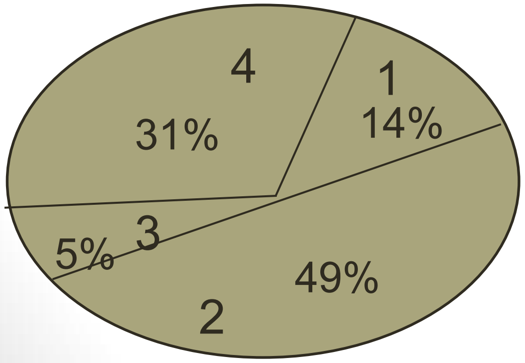
GA approach:

- Random initialization
- binary code, e.g. 01101 $\leftrightarrow$ 13
- Population size: 4
- Roulette wheel selection
- 1-point crossover, bitwise mutation

One generational cycle performed manually is shown here.

Genetic Algorithms

String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

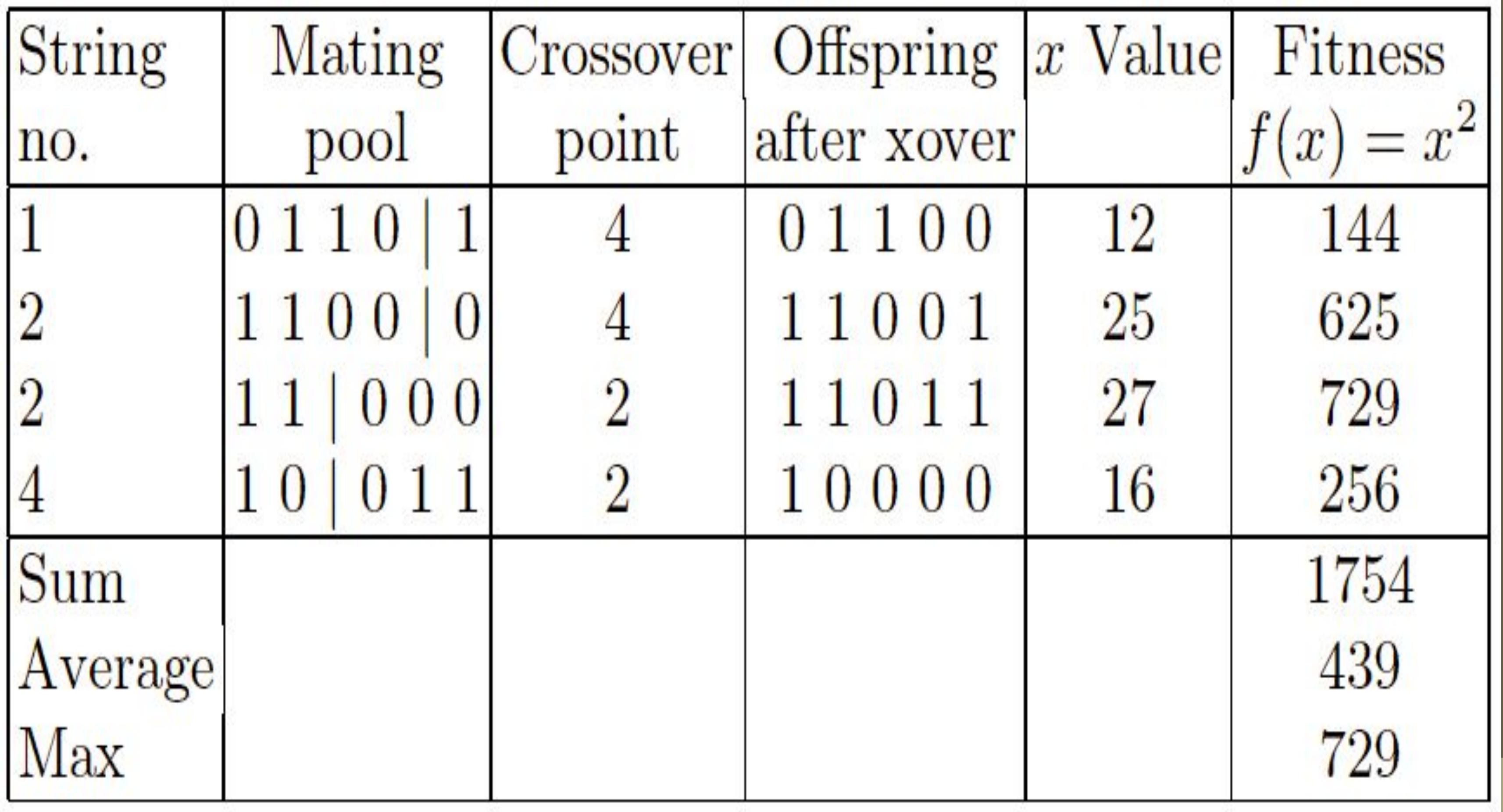


$$\text{Prob}_i = \frac{f(x)_i}{\sum_{i=1}^{n} f(x)_i}$$

$$\text{Expected count} = \frac{f(x)_i}{[\text{Avg}\, f(x)]_i}$$

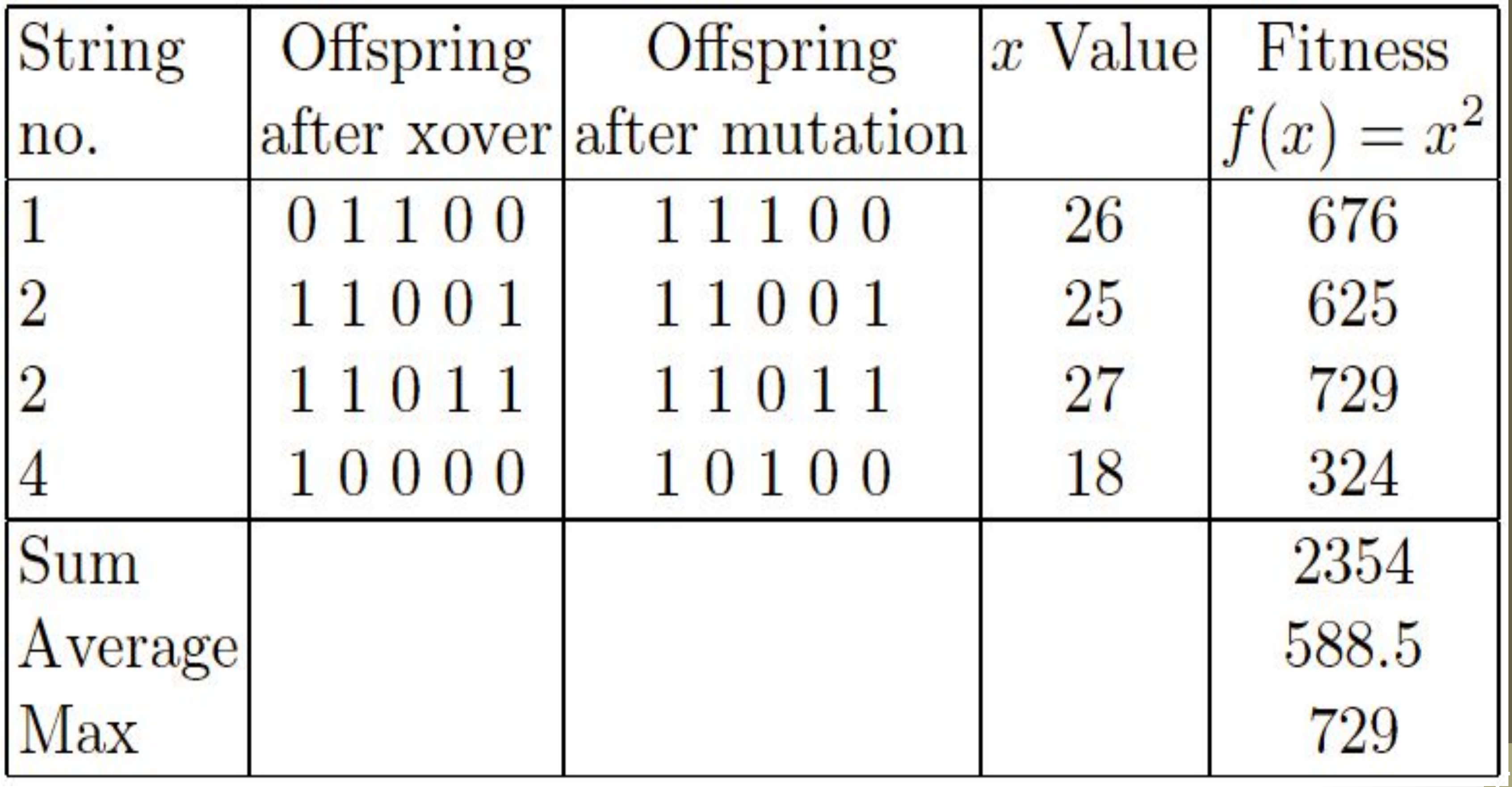
$$(\text{Avg}\, f(x))_i = \left[\frac{\sum_{i=1}^{n} f(x)_i}{n}\right]$$

Genetic Algorithm-An Example



String no.	Mating pool	Crossover point	Offspring after xover	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 | 1	4	0 1 1 0 0	12	144
2	1 1 0 0 | 0	4	1 1 0 0 1	25	625
2	1 1 | 0 0 0	2	1 1 0 1 1	27	729
4	1 0 | 0 1 1	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

Genetic Algorithm-An Example



String no.	Offspring after xover	Offspring after mutation	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	1 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 1 0 0	18	324
Sum				2354
Average				588.5
Max				729

Genetic Algorithm-N-Queens Example

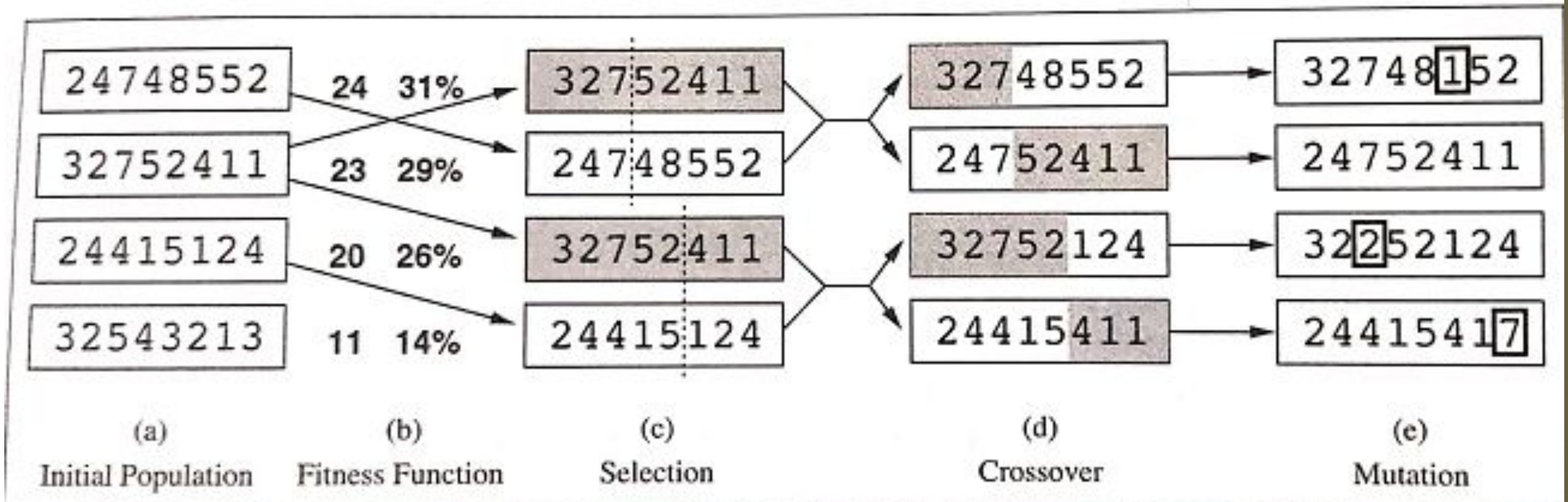


Figure 4.6 The genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by the fitness function in (b), resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

Genetic Algorithm-N-Queens Example

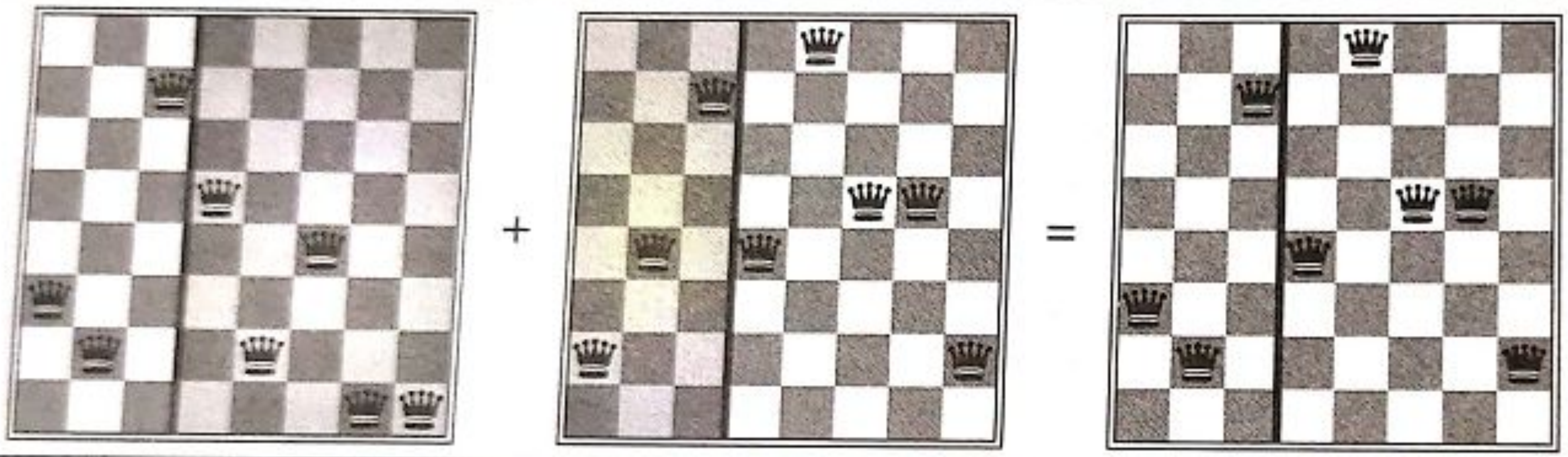


Figure 4.7 The 8-queens states corresponding to the first two parents in Figure 4.6(c) and the first offspring in Figure 4.6(d). The shaded columns are lost in the crossover step and the unshaded columns are retained.

Genetic Algorithms
When to use them?

- Genetic algorithms are easy to apply
- Results can be good on some problems, but bad on other problems (where to use GA?)
- Genetic algorithms are not well understood