Local search algorithms

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution
  - Local search: widely used for *very big* problems
  - Returns good but *not optimal* solutions in general

- The state space consists of "complete" configurations
  - For example, every permutation of the set of cities is a configuration for the traveling salesperson problem
- The goal is to find a “close to optimal” configuration satisfying constraints
  - Examples: n-Queens, VLSI layout, exam time table

- Local search algorithms
  - Keep a single "current" state, or small set of states
  - Iteratively try to improve it / them
  - Very memory efficient since only a few states are stored

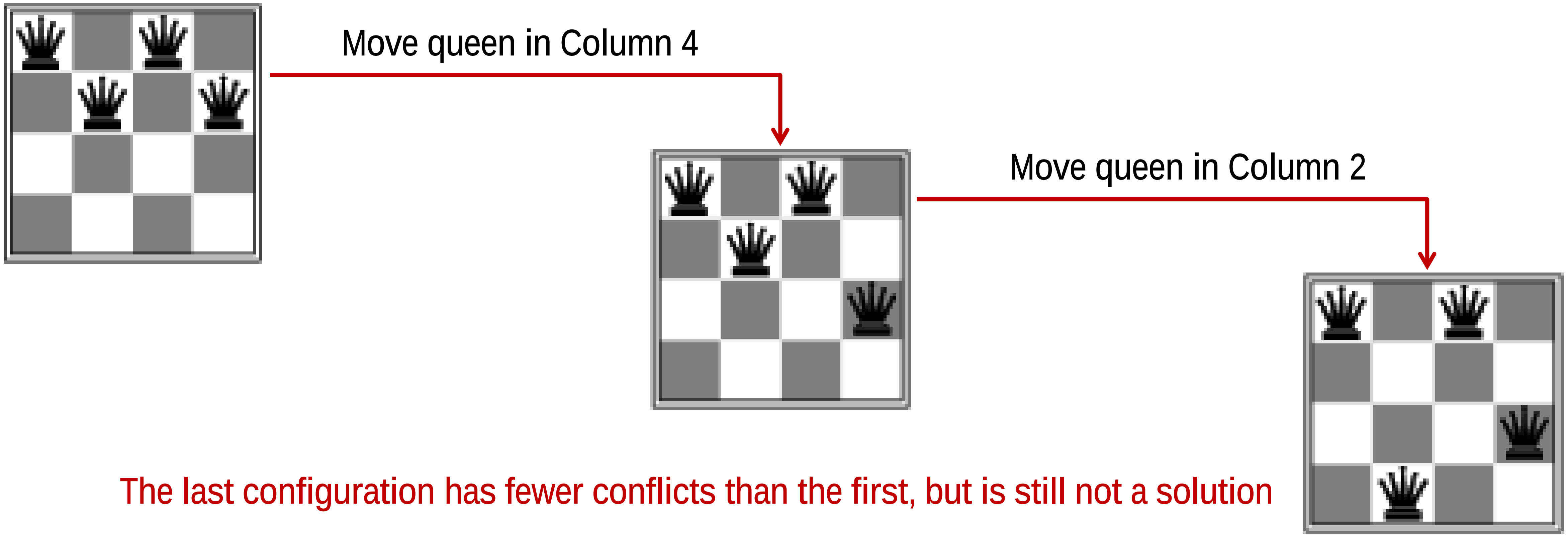
Example: *4*-queens

Goal: Put *4* queens on an *4* × *4* board with no two queens on the same row, column, or diagonal

State space: All configurations with the queens in distinct columns

State transition: Move a queen from its present place to some other square in the same column

Local Search: Start with a configuration and repeatedly use the moves to reach the goal



The last configuration has fewer conflicts than the first, but is still not a solution

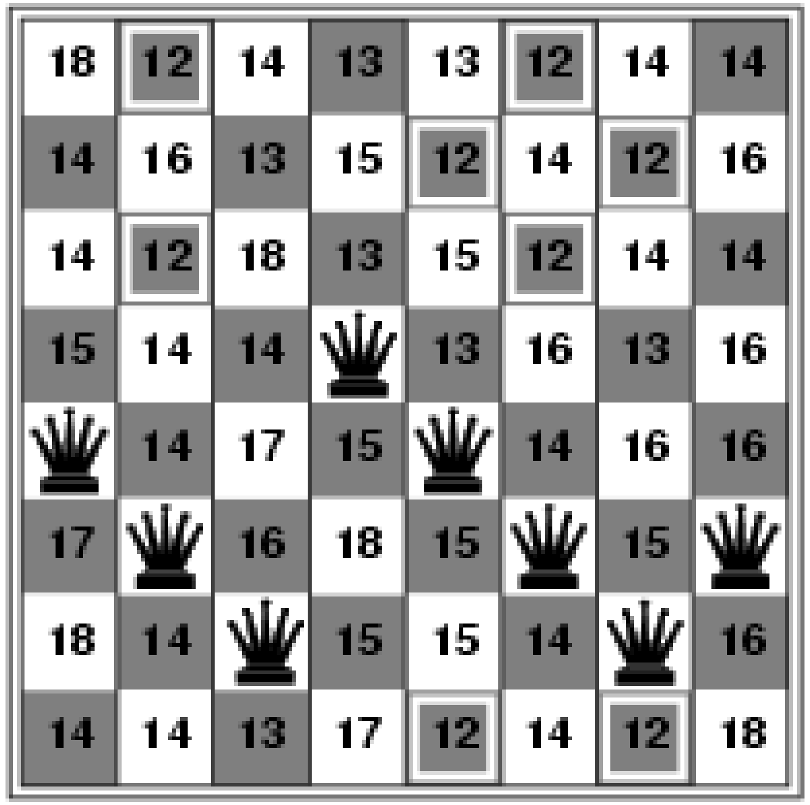
Gradient Descent in 8-queens

Value[state] = The numbers pairs of queens that are attacking each other, either directly or indirectly.

Value[state] = 17 for the state shown in the Fig.

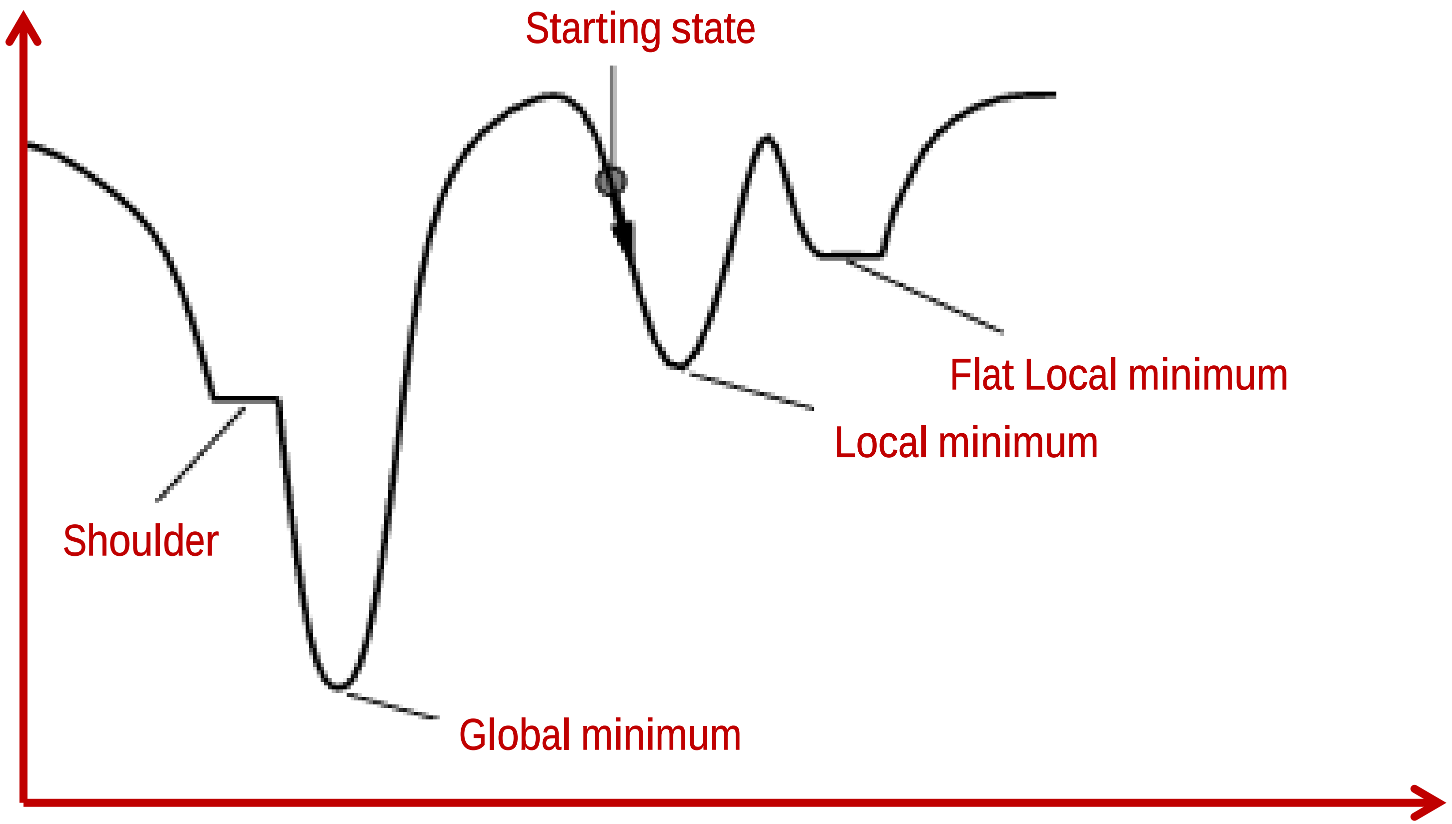
- The number in each square is the value of state if we move the queen in the same column to that square.

- Therefore the best greedy move is to move a queen to a square labeled with 12.
    - There are many such moves. We choose one of them at random.



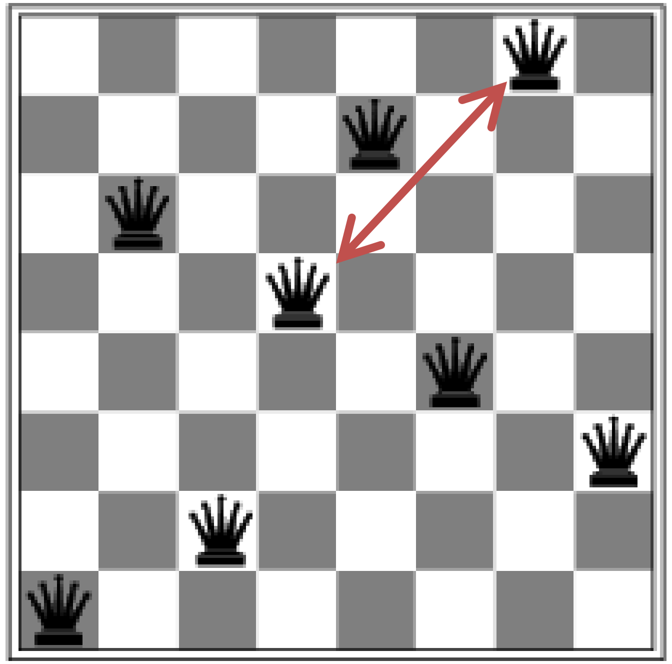
Gradient descent can get stuck in local minima

- Each neighbor of a minimum is inferior with respect to the minimum
- No move in a minimum takes us to a better state than the present state



Local minimum in 8-queens

- A local minimum with only one conflict
- All one-step neighbors have more than one conflict



How to get out of local minima?

S

1	2	3
5	7	-
4	8	6

4

up →

1	2	3
5	7	6
4	8	-

3

1	2	3
5	7	6
4	-	8

4

1	2	3
5	-	6
4	7	8

4

1	2	3
-	5	6
4	7	8

3

1	2	3
4	5	6
-	7	8

2

1	2	3
4	5	6
7	-	8

1

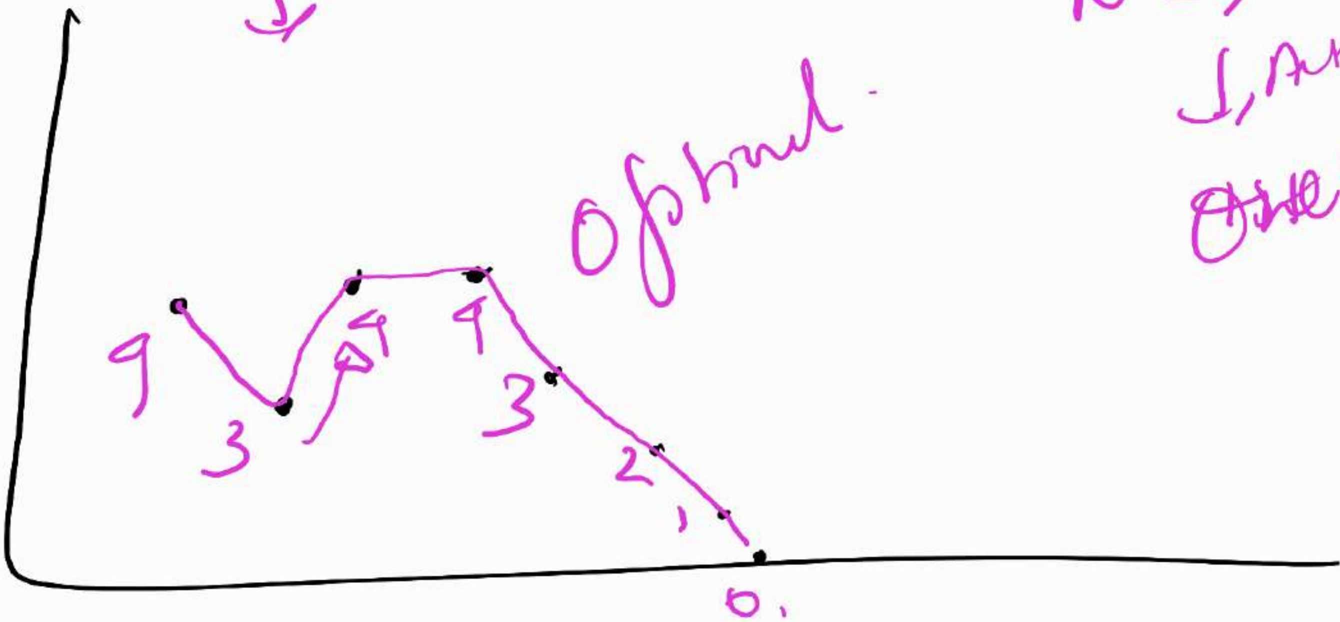
1	2	3
4	5	6
7	8	-

0

Goal

1	2	3
4	5	6
7	8	

$N \to 8$

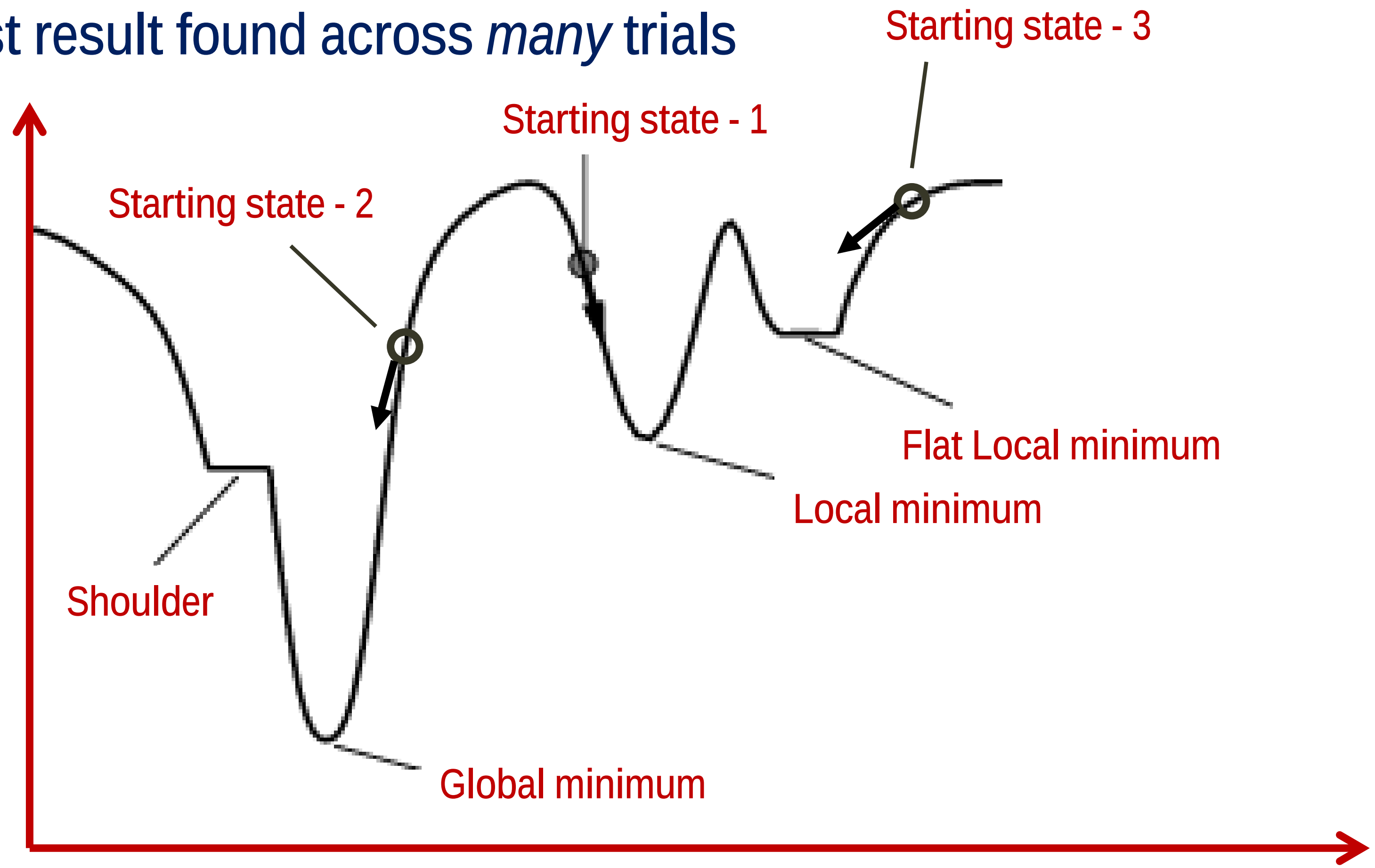


Gradient Descent with Random Restart

Using many random restarts improves our chances

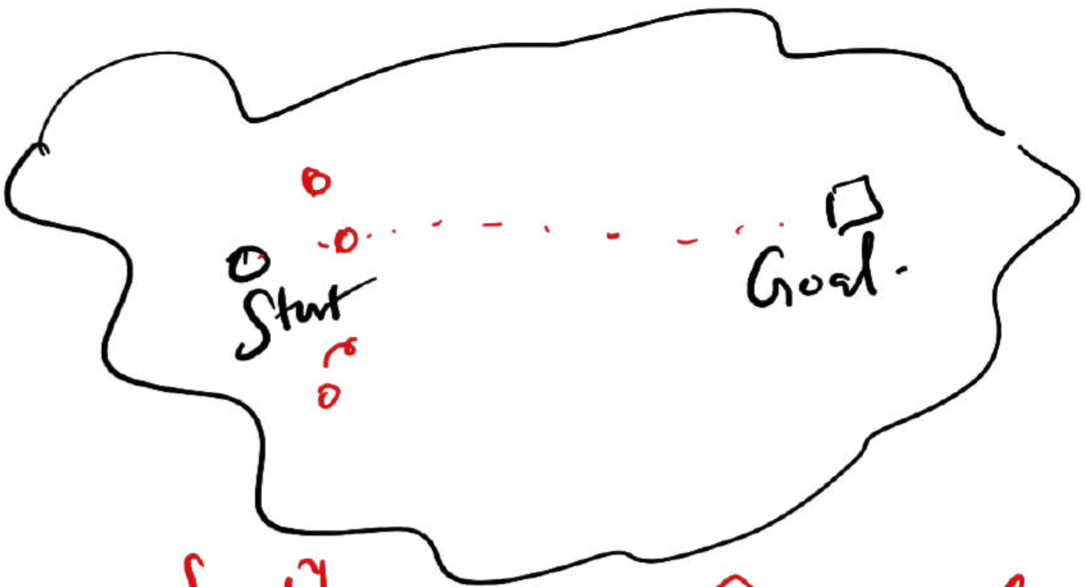
Restart a random initial state, *many times*

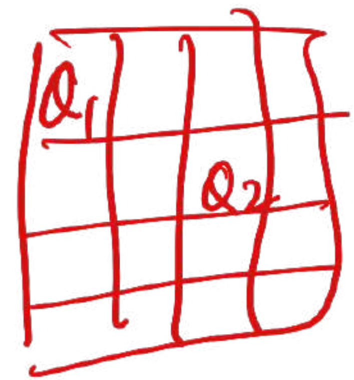
- Report the best result found across *many* trials



State Space.

MoveGen()





{0, 1}.

Valid. N variable = { Solution Search Space. }

2^N Candidates.

MoveGen() ⟵ Neighbourhood function

Problem:- Boolean Expression which has several Clauses, with propositional variables F = {A, B, C, D, E}, Where each Clause may be 0 (or) 1 [true/false] After evaluation of overall equation, the result should be true.

CNF:-

$$\overbrace{\underbrace{(a \vee \sim b)}_{1} \wedge \underbrace{(a \vee c)}_{2}}^{0} \wedge \overbrace{\underbrace{(\sim c \vee \sim d)}_{3} \wedge \underbrace{(d \vee \sim e)}_{4}}^{2}$$

True (1)

0

N variables → 2^N Candidates

⇑ Neighbourhood (Move Gen())

Hill Climbing algorithm :-

A ← Start

NewNode ← Best (MoveGen(A)).

While NewNode is better than A

Do

A ← newnode.

newNode ← Best (MoveGen(A))

end While.

return A

Neighbourhood function

11011 — 2^N

0 (or) 1

01011 10011 11111 11001 11001

00001

00011 10011 . . .

0(or)1 2

10001 01001 00101 00011 00000

3 3 2 3 2

0(or) 1 (6 or)

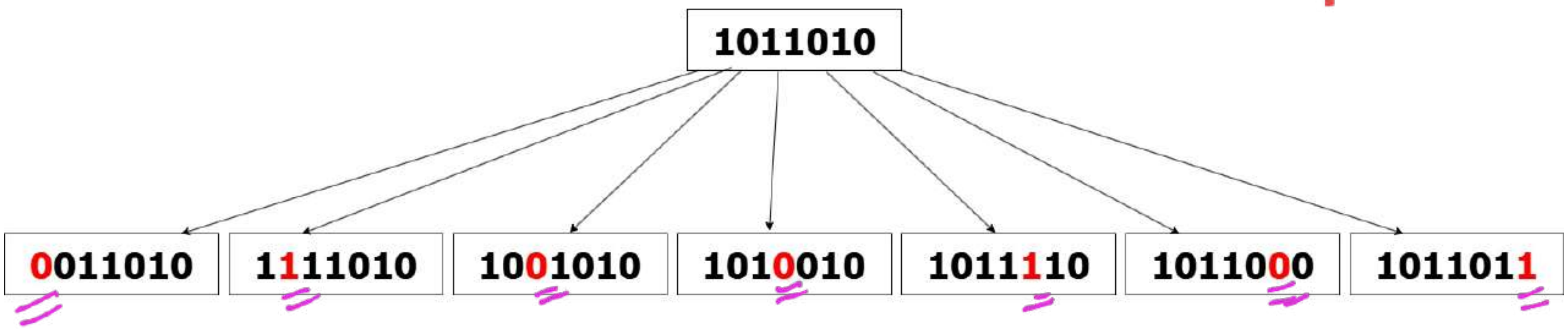
0111100 0001110 111000

4 4 4 5

5 4

Tabu Search for SAT

Move Gen ( ) = Neighbourhood function



Initailly M= 0000000

Tabu Search :an illustration (let tt=2)

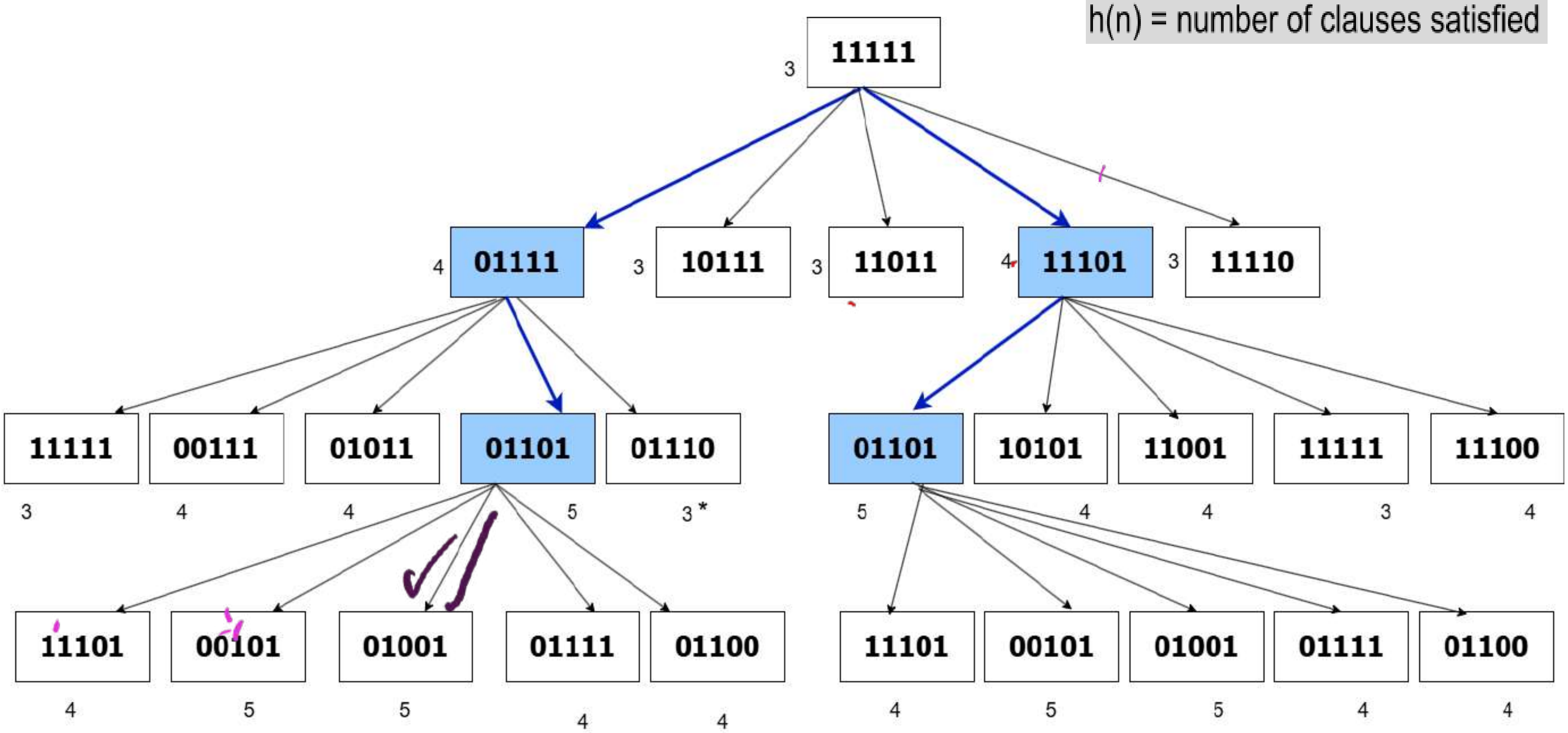
1011010	**Initially M=**	0000000	
1001010	**Change of bit 3: M=**	0020000	Cannot change bit 3 for 2 cycles
1001000	**Change of bit 6: M=**	0010020	
0001000	**Change of bit 1: M=**	2000010	

Beam Search for SAT

1 2 3 4 5 6

(b V ~c) ∧ (c V ~d) ∧ (~b) ∧ (~a V ~e) ∧ (e V ~c) ∧ (~c V ~d)

h(n) = number of clauses satisfied



Iterative Hill Climbing :-

best node ← random Candidate Solution.

for i = 1 to N

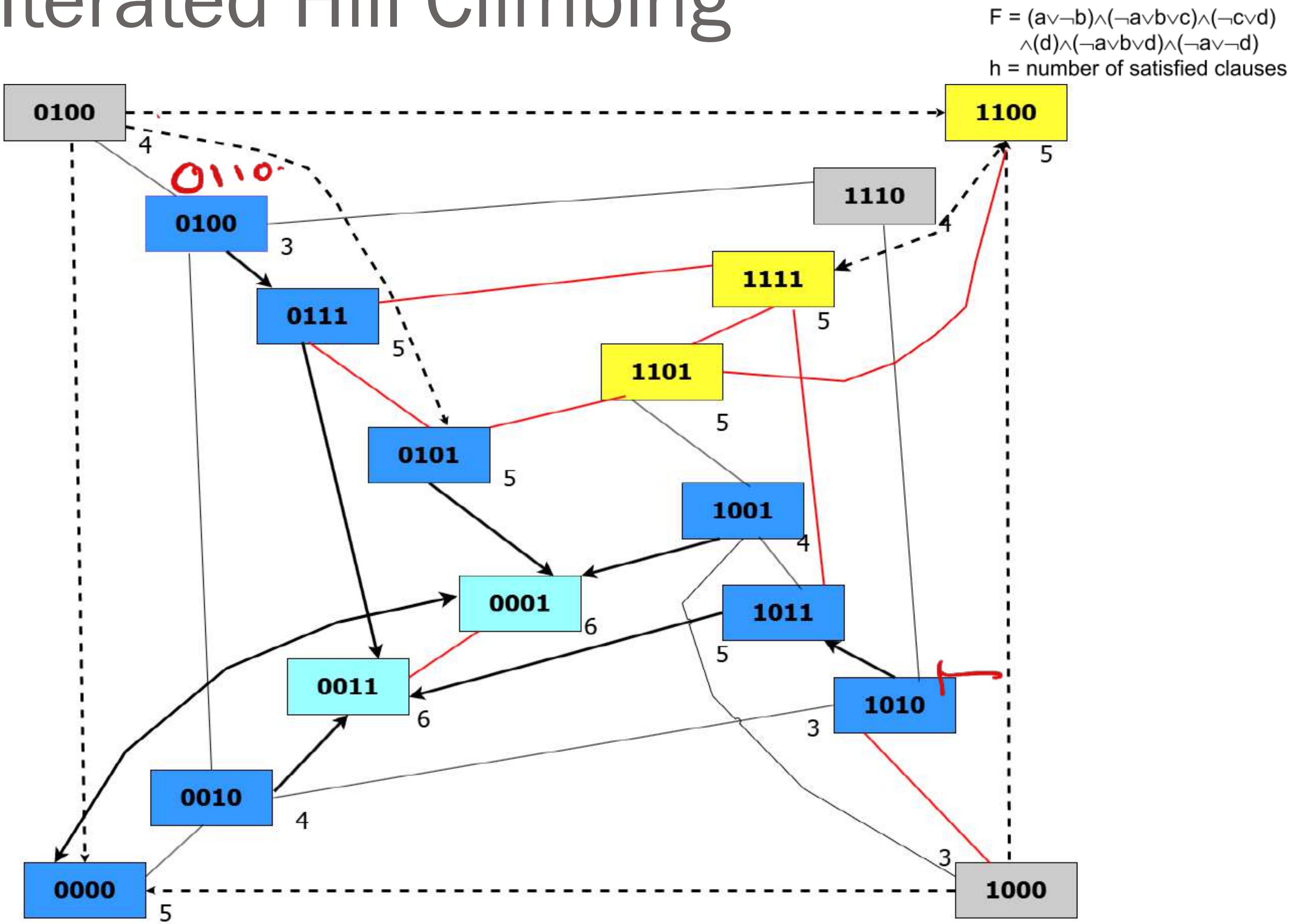
  Current best ← Hill-climbing (new random Candidate solution)

  if h(Current best) is better than h(best node)

    best node ← Current best

return best node.

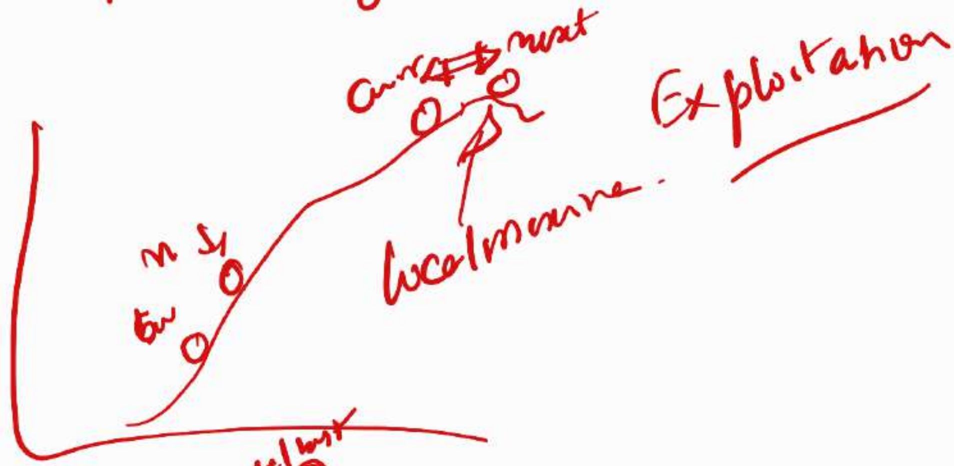
Iterated Hill Climbing

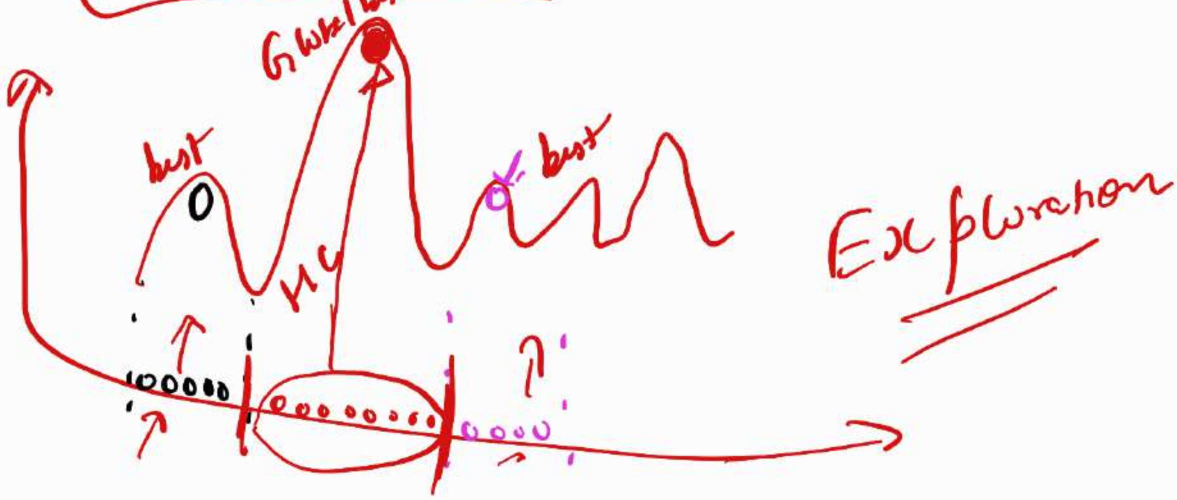


Sto chastic Hill Climbing:-

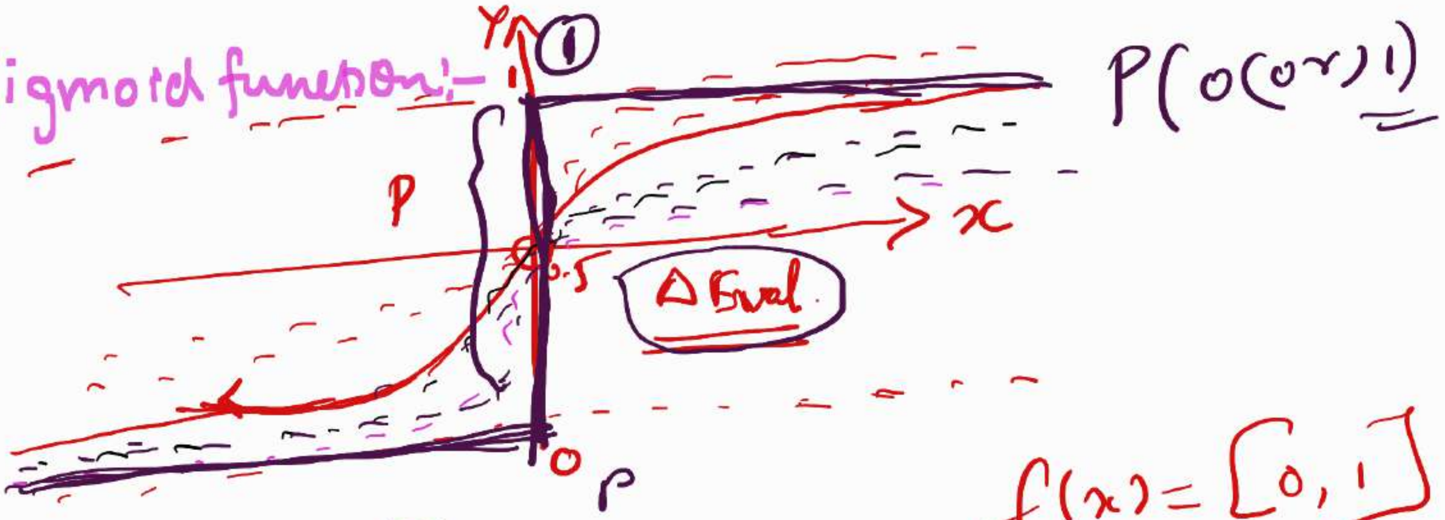
① Accept a good move with high probablity

② Accept a bad move also with low probablity.





Sigmoid function:-



$f(x) = [0, 1]$

$\boxed{\Delta Eval} = Eval(\text{neighbour}) - Eval(\text{Current})$

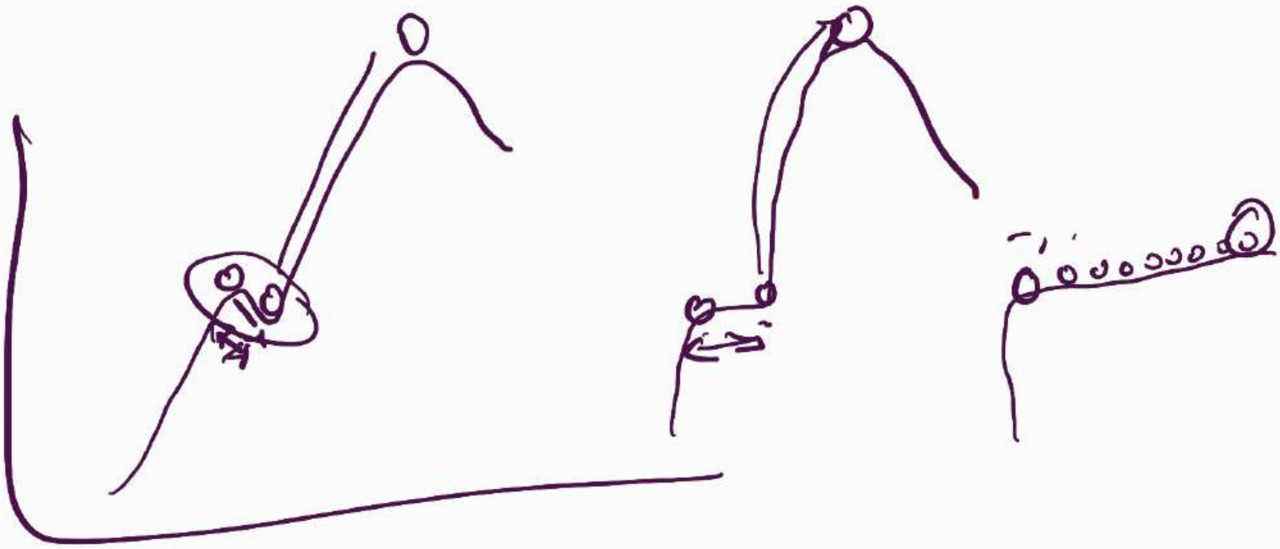
+ve, -ve

$-\Delta Eval = \text{Sigmoid} = \dfrac{1}{1+e^{-z}}$

$$\boxed{P = \frac{1}{1+e^{\frac{-\Delta Eval.}{T}}}}$$

$T \rightarrow$ Temperature

SA $\rightarrow$ Mix of both exploitation & exploration



Simulated Annealing ( )

node $\leftarrow$ random Candidate Solution. // Start

$T \leftarrow$ Higher Temperature Value.

for $i \leftarrow 1$ to no. of epochs // Termination Condition
do

Current $\leftarrow$ RandomNeighbour(node).

Δeval = EVAL(Current) − EVAL(node).

if (EVAL(Current) > EVAL(node)).

node $\leftarrow$ Current

elseif(random(0, 1) $\leq \left(\frac{1}{1+e^{\frac{-\Delta eval}{T}}} \right)$)

node $\leftarrow$ Current

$T \leftarrow$ Cooling (T, time)

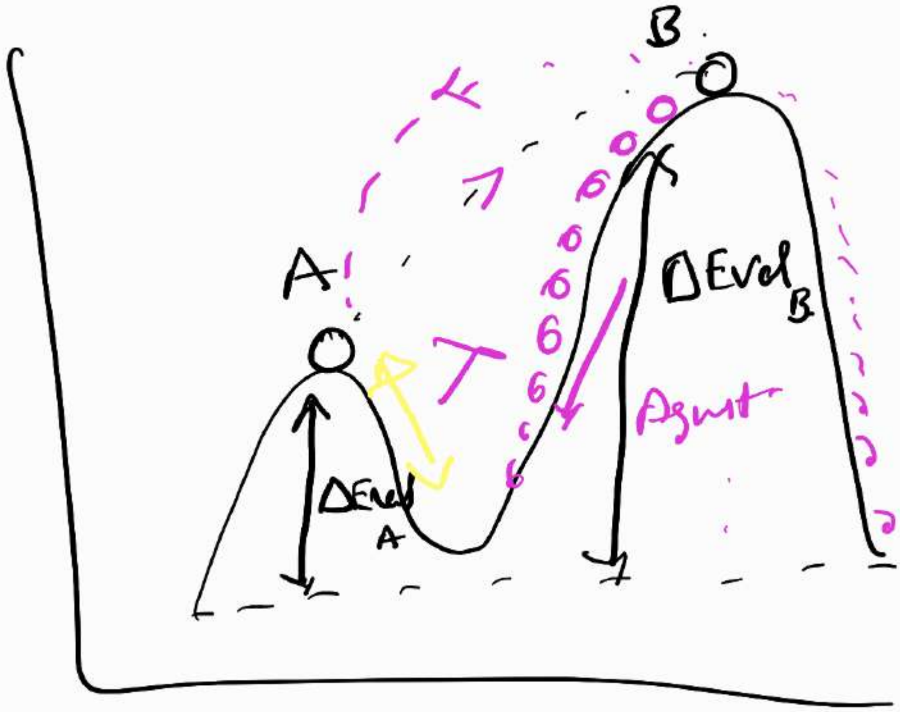
End for.

return node

// Cooling function lowers the temperature after each epoch.

// random (0, 1) generates a random number in the range.

$A \to B$
$B \to A$.



$\Delta Eval_A < \Delta Eval_B$.

Population based

Genetic Algorithm ← Survival of fittest

Gene :
Selection
Mutation :
Crossover :
→ function