Optimization

Optimization:

- Choosing the best option from a set of options.
- local search -> search algorithm that maintains a single node & searches by moving to a neighbouring node.
- General strategy in local search-
  - take a state, maintaining some current node, & move where we are looking at in this state-space landscape in order to try to find global max./global min. somehow.

Hill climbing (Pseudocode):

```
function HILL-CLIMB (problem):
    current = inital state of problem
    repeat:
     neighbour = highest valued neighbour of current
     if neighbour not better than current:
      return current
     current = neighbour
```

Drawbacks of Hill-climb: Shoulder problem, not always find optimal global max/min.

Teacher's Signature

Hill climb variants:

Variants	Definition
steepest-ascent	choose the highest valued neighbour.
stochastic	choose randomly from high-valued neighbours.
first-choice	choose the first higher-valued neighbours.
random-restart	conduct hill climbing multiple times
local beam search	choose the K-highest-valued neighbour

Simulated Annealing:

- Early on, higher "temperature": more likely to accept neighbors that are worse than current state.
- Later on, lower "temperature": less likely to accept neighbors that are worse than current state.

Pseudo code:

```
function SIMULATED-ANNEALING(problem, max):
current = inital state of problem
for t=1 to max:
 T = TEMPERATURE(t)
neighbor = random neighbor of current
ΔE = how much better neighbor is than current
```

Teacher's Signature

if $\Delta E > 0$:
Current = neighbor
With probability $e^{\Delta E/T}$ set current = neighbor

- Linear programming: minimize a linear objective function subject to linear equality and inequality constraints.

Goal:
- to minimize a cost function $c_1x_1 + c_2x_2 + \dots + c_nx_n$
- With constraints of form $a_1x_1 + a_2x_2 + \dots + a_nx_n \leq b$
- with bounds for each variable $l_i \leq x_i \leq u_i$

Example:
- Two machines X_1 & X_2. X_1 cost ₹50/h, X_2 cost ₹80/h. Goal is to minimize the cost.
- X_1 requires 5 unit of labor/h, X_2 requires 2 unit of labor/h. Total of 20 unit of labor to spend.
- X_1 produces 10 unit of output/h, X_2 produces 12 unit of output/h. Company needs 90 unit of output.

Cost function: $50x_1 + 80x_2$ – from(i)

Constraint: $5x_1 + 2x_2 \leq 20$ – from(ii)

Constraint: $10x_1 + 12x_2 \geq 90$ – from(iii)

$\Rightarrow (-10x_1) + (-12x_2) \leq -90$

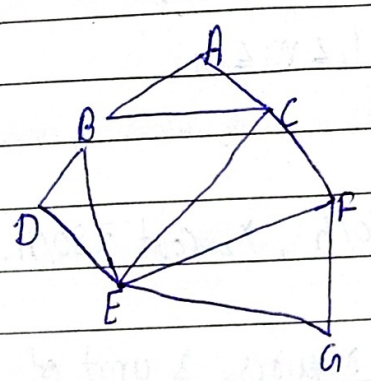
Algorithms:
- Simplex
- Interior-Point

Teacher's Signature

Constraint Satisfaction:

- Student Taking classes | Exam slots:

1	A B C	Monday
2	B D E	Tuesday
3	C E F	Wednesday
4	E F G	



Constraint Graph

Variable : {A, B, C, D, E, F, G}
Domains : {Monday, Tuesday, Wednesday}
Constraints : { A≠B, A≠C, B≠C, B≠D, B≠E
C≠E, C≠F, E≠F, F≠G, E≠G }

- Set of variables $\{X_1, X_2, \ldots X_n\}$
- Set of domains for each variable $\{D_1, D_2, \ldots D_n\}$
- Set of constraints C

- Sudoku Solving:

Variables : {(0,2), (1,1), (1,2), (2,0)....}
Domains : {1, 2, 3, 4, 5, 6, 7, 8, 9}
Constraints : {(0,2) ≠ (1,1) ≠ (1,2) ≠ (2,0) .....}

→ hard constraint : Constraint that must be satisfied in a correct sol.
→ soft constraint : Constraint that express some notation of which solution are preffered over others

→ Unary Constraint: Constraint involving only one variable
{A ≠ Monday}

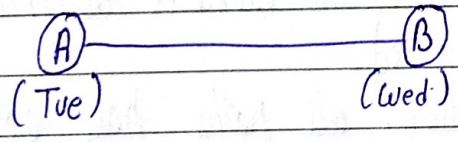
→ binary Constraint: Constraint involving two variable.
{A ≠ B}

→ node consistency → when all the values in a variable's domain satisfy the variable's unary constraints.

→ arc consistency → when all the values in a variable's domain satisfy the variable's binary constraints.

- To make X arc-consistent with respect to Y, remove elements from X's domain untill every choice for X has a possible choice for Y.

Constraints: {A ≠ Mon, B ≠ Tue, B ≠ Mon, A ≠ B}

node consistency : 

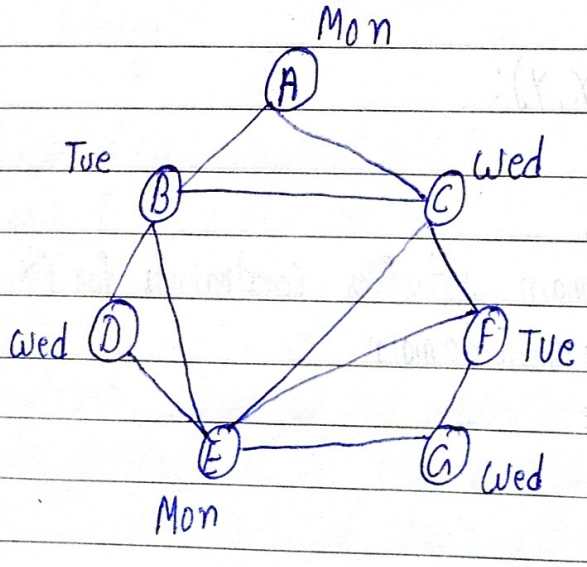
- pseudocode:

```
function REVISE(csp, X, Y):
    revised = false
    for u in X.domain:
        if no y in Y.domain satisfies constraints for (X, Y):
            delete u from X.domain
            revised = true
    return revised
```

```
function AC-3(csp):
    queue = all arcs in csp
    while queue non-empty:
        (X,Y) = DEQUEUE(queue)
        if REVISE(csp, X, Y):
            if size of X.domain == 0:
                return false
            for each Z in neighbors - {Y}:
                ENQUEUE(queue, (Z,X))
    return true
```

→ CSPs as Search Problems →

- inital state : empty assignment (no variable)
- actions : add a {variable = value} to assignment
- transition model : shows how adding an assignment changes the assignment.
- goal test : check if all variables assigned and constraints all satisfied.
- path cost function : all paths have same cost.



Backtracking Search:

```
function BACKTRACK(assignment, csp):
    if assignment complete: return assignment
    var = SELECT-UNASSIGNED-VAR(assignment, csp)
    for value in DOMAIN-VALUES(var, assignment, csp):
        if value consistent with assignment:
            add {var = value} to assignment
            inferences = INFERENCE(assignment, csp)
            if inferences ≠ failure: add inference to assignment
            result = BACKTRACK(assignment, csp)
            if result ≠ failure: return result
        remove {var = value} from assignment
        remove inferences
    return failure
```

- maintaining arc-consistency: algo. for enforcing arc-consistency everytime we make a new assignment.
  - When we make a new assignment to X, calls AC-3, starting with a queue of all arcs (Y, X) where Y is a neighbor of X.

→ SELECT-UNASSIGNED-VAR:

- minimum remaining values (MRV) heuristic: select the variable that has the smallest domain.
- degree heuristic: select the variable that has the highest degree.

→ DOMAIN-VALUES;

- least-constraining values heuristic: return variable in order by no. of choices that are ruled out for neighboring variables.
  - try least-containing values first

Teacher's Signature .....