Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)
Department of Computer Science and Engineering (Data Science)

**Subject: Artificial Intelligence (DJ19DSC502)**

**AY: 2023-24**

**Experiment 4**

**(Solution Space)**

**Name: Bhuvi Ghosh** **Sap ID: 60009210191** **Batch: D22**

**Aim:** Find the solution of a SAT (Satisfiability) problem using Variable Neighborhood Descent.

**Theory:**

## The SAT problem

Given a Boolean formula made up of a set of propositional variables V= {a, b, c, d, e, … } each of which can be *true* or *false*, or *1* or *0*, to find an assignment for the variables such that the given formula evaluates to *true* or *1*.

For example, F = ((aV~e) ∧ (eV~c)) ⊃ (~cV~d) can be made *true* by the assignment {a=*true*, c=*true*, d=*false*, e=*false*} amongst others.

Very often *SAT* problems are studied in the *Conjunctive Normal Form (CNF)*. For example, the following formula has five variables (a,b,c,d,e) and six clauses.

(bV~c) ∧ (cV~d) ∧ (~b) ∧ (~aV~e) ∧ (eV~c) ∧ (~cV~d)

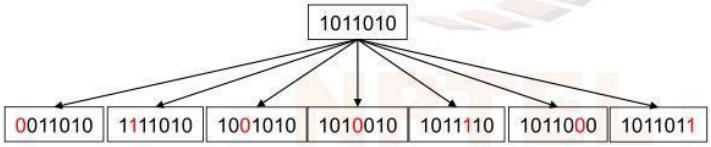
## Solution Space Search and Perturbative methods

The Solution Space is the space of candidate solutions.

A local search method generates the neighbours of a candidate by applying some perturbation to the given candidate
MoveGen function = neighbourhood function

A SAT problem with N variables has $2^N$ candidates
- where each candidate is a N bit string

When N= 7, a *neigbourhood function* may change one bit.




## Variable Neighbourhood Descent

```
VariableNeighbourhoodDescent()
1       node ← start
2       for i ← 1 to n
3               do moveGen ← MoveGen(i)
4                  node ← HillClimbing(node, moveGen)
5       return node
```

The algorithm assumes that the function *moveGen* can be passed as a parameter. It assumes that there are *N moveGen* functions sorted according to the density of the neighbourhoods produced.

**Lab Assignment to do:**

Solve the following SAT problems using VND

1. F = (A V ~B) ^ (B V ~C) ^ (~B) ^ (~C V E) ^ (A V C) ^ (~C V ~D)
2. F = ( A V B) ^ ( A ^ ~C) ^ ( B ^ D ) ^ ( A V ~E)

```python
from itertools import product

def problem_1_formula(A, B, C, D, E):
    return (A or not B) and (B or not C) and (not B) and (not C or E) and (A or C) and (not C or not D)

def problem_2_formula(A, B, C, D, E):
    return (A or B) and (A and not C) and (B and D) and (A or not E)
```

```python
def satisfiability(formula):
    variables = ['A', 'B', 'C', 'D', 'E']
    solutions = []

    for assignment in product([True, False], repeat=len(variables)):
        if formula(*assignment):
            solutions.append({variables[i]: assignment[i] for i in range(len(variables))})

    return solutions
```

```python
def satisfiability(formula):
    variables = ['A', 'B', 'C', 'D', 'E']
    solutions = []

    for assignment in product([True, False], repeat=len(variables)):
        if formula(*assignment):
            solutions.append({variables[i]: assignment[i] for i in range(len(variables))})

    return solutions
```

```python
problem_1_solutions = satisfiability(problem_1_formula)
print("Solutions for Problem 1:")
for solution in problem_1_solutions:
    print(solution)
```

```
Solutions for Problem 1:
{'A': True, 'B': False, 'C': False, 'D': True, 'E': True}
{'A': True, 'B': False, 'C': False, 'D': True, 'E': False}
{'A': True, 'B': False, 'C': False, 'D': False, 'E': True}
{'A': True, 'B': False, 'C': False, 'D': False, 'E': False}
CPU times: user 167 µs, sys: 23 µs, total: 190 µs
Wall time: 197 µs
```

```
problem_2_solutions = satisfiability(problem_2_formula)
print("Solutions for Problem 2:")
for solution in problem_2_solutions:
    print(solution)
```

```
Solutions for Problem 2:
{'A': True, 'B': True, 'C': False, 'D': True, 'E': True}
{'A': True, 'B': True, 'C': False, 'D': True, 'E': False}
CPU times: user 120 µs, sys: 0 ns, total: 120 µs
Wall time: 124 µs
```

```
[ ] from itertools import product

    def satisfiability_vnd(formula):
        variables = ['A', 'B', 'C', 'D', 'E']
        solutions = []
        initial_assignment = [True] * len(variables)

        def flip(assignment, index):
            new_assignment = assignment.copy()
            new_assignment[index] = not assignment[index]
            return new_assignment

        def vnd_search(assignment, index):
            if index == len(variables):
                if formula(*assignment):
                    solutions.append({variables[i]: assignment[i] for i in range(len(variables))})
                return

            vnd_search(assignment, index + 1)
            vnd_search(flip(assignment, index), index + 1)

        vnd_search(initial_assignment, 0)
        return solutions
```

```
problem_1_solutions = satisfiability_vnd(problem_1_formula)
print("Solutions for Problem 1:")
for solution in problem_1_solutions:
    print(solution)
```

```
Solutions for Problem 1:
{'A': True, 'B': False, 'C': False, 'D': True, 'E': True}
{'A': True, 'B': False, 'C': False, 'D': True, 'E': False}
{'A': True, 'B': False, 'C': False, 'D': False, 'E': True}
{'A': True, 'B': False, 'C': False, 'D': False, 'E': False}
```

```
problem_2_solutions = satisfiability_vnd(problem_2_formula)
print("Solutions for Problem 2:")
for solution in problem_2_solutions:
    print(solution)
```

```
Solutions for Problem 2:
{'A': True, 'B': True, 'C': False, 'D': True, 'E': True}
{'A': True, 'B': True, 'C': False, 'D': True, 'E': False}
```

```python
def satisfiability_vnd(formula):
    variables = ['A', 'B', 'C', 'D', 'E']
    solutions = []
    initial_assignment = [True] * len(variables)

    def flip(assignment, index):
        new_assignment = assignment.copy()
        new_assignment[index] = not assignment[index]
        return new_assignment

    def vnd_search(assignment, index):
        if index == len(variables):
            if formula(*assignment):
                solutions.append({variables[i]: assignment[i] for i in range(len(variables))})
            return

        print(f"Iteration {index + 1}: Assignment = {assignment}")
        vnd_search(assignment, index + 1)

        flipped_assignment = flip(assignment, index)
        print(f"Iteration {index + 1}: Flipping variable {variables[index]}")
        vnd_search(flipped_assignment, index + 1)

    vnd_search(initial_assignment, 0)
    return solutions

solutions = satisfiability_vnd(problem_2_formula)
```

```
Iteration 1: Assignment = [True, True, True, True, True]
Iteration 2: Assignment = [True, True, True, True, True]
Iteration 3: Assignment = [True, True, True, True, True]
Iteration 4: Assignment = [True, True, True, True, True]
Iteration 5: Assignment = [True, True, True, True, True]
Iteration 5: Flipping variable E
Iteration 4: Flipping variable D
Iteration 5: Assignment = [True, True, True, False, True]
Iteration 5: Flipping variable E
Iteration 3: Flipping variable C
Iteration 4: Assignment = [True, True, False, True, True]
Iteration 5: Assignment = [True, True, False, True, True]
Iteration 5: Flipping variable E
Iteration 4: Flipping variable D
Iteration 5: Assignment = [True, True, False, False, True]
Iteration 5: Flipping variable E
Iteration 2: Flipping variable B
Iteration 3: Assignment = [True, False, True, True, True]
Iteration 4: Assignment = [True, False, True, True, True]
Iteration 5: Assignment = [True, False, True, True, True]
Iteration 5: Flipping variable E
Iteration 4: Flipping variable D
Iteration 5: Assignment = [True, False, True, False, True]
Iteration 5: Flipping variable E
Iteration 3: Flipping variable C
Iteration 4: Assignment = [True, False, False, True, True]
Iteration 5: Assignment = [True, False, False, True, True]
```

```
Iteration 4: Flipping variable D
Iteration 5: Assignment = [False, True, True, False, True]
Iteration 5: Flipping variable E
Iteration 3: Flipping variable C
Iteration 4: Assignment = [False, True, False, True, True]
Iteration 5: Assignment = [False, True, False, True, True]
Iteration 5: Flipping variable E
Iteration 4: Flipping variable D
Iteration 5: Assignment = [False, True, False, False, True]
Iteration 5: Flipping variable E
Iteration 2: Flipping variable B
Iteration 3: Assignment = [False, False, True, True, True]
Iteration 4: Assignment = [False, False, True, True, True]
Iteration 5: Assignment = [False, False, True, True, True]
Iteration 5: Flipping variable E
Iteration 4: Flipping variable D
Iteration 5: Assignment = [False, False, True, False, True]
Iteration 5: Flipping variable E
Iteration 3: Flipping variable C
Iteration 4: Assignment = [False, False, False, True, True]
Iteration 5: Assignment = [False, False, False, True, True]
Iteration 5: Flipping variable E
Iteration 4: Flipping variable D
Iteration 5: Assignment = [False, False, False, False, True]
Iteration 5: Flipping variable E
```