# SMART AFFORESTATION AGENT

## Problem Statement:

To design an AI agent for suggesting a feasible distribution of trees to be planted in an area, given various factors, namely, Air Quality Index, Climate, Cost Budget, Area Population, Plantation Area available, Plant Utility, Plant Air Pollution Tolerance, etc.

## Overview:

First, we will be exploring how the problem can be viewed as just another example of the knapsack problem. We will look into the dynamic programming-based approach and the exhaustive search and high run time involved.
Artificial Intelligence-based methods like Genetic Algorithms have been explored in regards to solving MKP integer-programming like problems. Finally, we present an algorithm for solving the aforementioned optimisation task using Genetic Algorithm-based technique.

## Motivation and Approach:

In the present scenario, with hazardous air pollution levels, one doesn't just need to plant trees, instead, this task has to be done smartly, keeping in mind the ecological and economic implications. We can make use of technology for such tasks and obtain feasible solutions quickly.

The problem can be formulated as follows:
Given a set of M trees with scores S = {$s1$, $s2$,.., $sM$}, to choose from along with area occupied *Area* = {$a1$, $a2$,.., $aM$} and cost per tree *Cost* = {$c1$, $c2$,.., $cM$}, find a frequency distribution X = {$x1$, $x2$,.., $xM$} , so as to maximize the objective function:

$$\text{Maximize:} \quad f(X) = \Sigma\, si.xi$$

$$\text{Subject to Constraints:} \quad \Sigma\, ai.xi \le A, \quad \Sigma\, ci.xi \le C$$

Where A = plantation area available and C = cost budget available

The problem is basically a variant of the **Multi-dimensional Knapsack Problem (0/1 unbounded)**, where the multiple dimensions arise from the two constraints laid on the optimization procedure, namely area and cost.

Multi-dimensional Knapsack Problem (MKP) is a particularly difficult problem of **integer programming** since the constraint matrix is dense.

## Implementation:

This multi-dimensional knapsack problem can be solved using a **dynamic programming algorithm** as follows:

```
# values are taken as defined above
Knapsack_dp(A, C, M, S, Area, Cost, X):
    if M == 0 or A == 0 or C == 0:
        return 0
    If Area[M] > A or Cost[M] > C:
        return Knapsack_dp(A, C, M-1, S, Area, Cost, X)
    X1 = X[::]
    X2 = X[::]
    v1 = Knapsack_dp(A, C, M-1, S, Area, Cost, X1)
    v2 = S1[M] + Knapsack_dp(A-Area[M], C-Cost[M],M,S,Area,Cost, X2)
    if v1 <= v2:
        X[::] = X2[::]
        X[M] += 1
        return v2
    X[::] = X1[::]
    return v1
```

Although the above algorithm seems straightforward, it does an exhaustive search and has a **Time Complexity O(A*C*M)**.
For large values of A, C and M the algorithm takes an excessive amount of time to finish and return the optimal solution.

## The Compromise:

In order to obtain a solution much more quickly, we will need to do a compromise between time and optimality. We may not search for the best solution but a good, feasible one in its place. We will be using an evolutionary AI technique called the **Genetic Algorithm**.

## Genetic Algorithm based Approach:

The **multi-dimensional knapsack problem can be solved using a Genetic Algorithm**. These algorithms belong to the larger class of evolutionary algorithms (EA), that generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover. It can be said that the strongest individuals in a population will have a better chance to transfer their genes to the next generation.

In a genetic algorithm, a population of candidate solutions to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties which can be mutated and altered; traditionally, solutions are represented as chromosomes of binary strings of 0s and 1s, but other encodings are also possible.

To solve MKP using the genetic algorithm we can first generate a sampling set to pick from and then represent its frequency array (a chromosome) as a string of 0s and 1s.

```
Sample-Set = {}
for each tree i = 1, M:
      count = min(C/Cost[i], A/Area[i])
      Sample-Set.add({i}*count)    # array of i, count times
```

*Heuristic Function*:

We will be using an improved objective function incorporating the Area utilisation and Cost Budget Utilisation terms, we define fitness

$$f = w1 * total\_score + w2 * (total\_area\_used - A) + w3 * (total\_cost - C)$$

Because of the constraints $total\_cost <= C$ and $total\_area\_used <= A$ and the cost and area utilisation terms can have maximum values of 0, and hence might lead to negative values of fitness.

If constraints are violated then,

$$fitness(f) = -\infty \qquad \text{(high negative)}.$$

Here, w1, w2 and w3 are weights of the heuristic function, which are needed to be chosen so as to yield the best possible search result. w1 can also incorporate population term and be made to follow the relation:

$$w1 \propto 1/Population$$

*Reproduction Technique*:

**Crossovers** have been used to generate a new generation of chromosomes from the existing ones in the following manner:

1. Sort the chromosomes based on fitness values and discard the second half, resulting in an array of length $L/2$
2. Now for each $i$, take a pair of it $i$-$th$ and $(L/2-i-1)$-$th$ chromosomes, say A and B, and do the following:
   2.a. Keep A
   2.b. Keep B
   2.c. Choose a pivot p and keep A[:p] + B[p:]
   2.d. Keep B[:p] + A[p:]
3. Thus, we finally end up with L chromosomes again.

*Search Methodology*:

Search repeatedly by doing crossovers and storing the best chromosomes generated. If fitness value repeats for some $r$ consecutive iterations, then initialise new chromosomes randomly and continue for max_iter. Finally, return the chromosome encountered with the best fitness value.

## Conclusion:

The GA based evolutionary approach is much faster than the Dynamic Programming based approach. Although it compromises with the quality of the solution, it is still effective in returning a feasible solution.