Means-Ends Analysis in Artificial Intelligence

- We have studied the strategies which can reason either in forward or backward, but a mixture of the two directions is appropriate for solving a complex and large problem. Such a mixed strategy, make it possible that first to solve the major part of a problem and then go back and solve the small problems arise during combining the big parts of the problem. Such a technique is called **Means-Ends Analysis**.
- Means-Ends Analysis is problem-solving techniques used in Artificial intelligence for limiting search in AI programs.
- It is a mixture of Backward and forward search technique.
- The MEA technique was first introduced in 1961 by Allen Newell, and Herbert A. Simon in their problem-solving computer program, which was named as General Problem Solver (GPS).
- The MEA analysis process centered on the evaluation of the difference between the current state and goal state.

How means-ends analysis Works:

The means-ends analysis process can be applied recursively for a problem. It is a strategy to control search in problem-solving. Following are the main Steps which describes the working of MEA technique for solving a problem.

a. First, evaluate the difference between Initial State and final State.
b. Select the various operators which can be applied for each difference.
c. Apply the operator at each difference, which reduces the difference between the current state and goal state.

Operator Subgoaling

In the MEA process, we detect the differences between the current state and goal state. Once these differences occur, then we can apply an operator to reduce the differences. But sometimes it is possible that an operator cannot be applied to the current state. So we create the subproblem of the current state, in which operator can be applied, such type of backward chaining in which operators are selected, and then sub goals are set up to establish the preconditions of the operator is called **Operator Subgoaling**.

Algorithm for Means-Ends Analysis:

Let's we take Current state as CURRENT and Goal State as GOAL, then following are the steps for the MEA algorithm.

- **Step 1:** Compare CURRENT to GOAL, if there are no differences between both then return Success and Exit.
- **Step 2:** Else, select the most significant difference and reduce it by doing the following steps until the success or failure occurs.

a. Select a new operator O which is applicable for the current difference, and if there is no such operator, then signal failure.

   b. Attempt to apply operator O to CURRENT. Make a description of two states.
   i) O-Start, a state in which O?s preconditions are satisfied.
   ii) O-Result, the state that would result if O were applied In O-start.

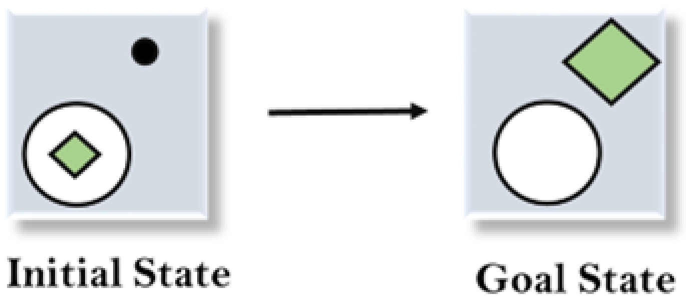
   c. If
   (First-Part <------ MEA (CURRENT, O-START)
   And
   (LAST-Part <----- MEA (O-Result, GOAL), are successful, then

signal Success and return the result of combining FIRST-PART, O, and LAST-PART.

The above-discussed algorithm is more suitable for a simple problem and not adequate for solving complex problems.

Example of Mean-Ends Analysis:

Let's take an example where we know the initial state and goal state as given below. In this problem, we need to get the goal state by finding differences between the initial state and goal state and applying operators.

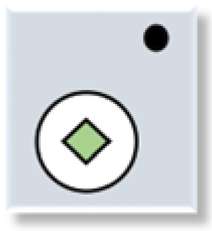


Solution:

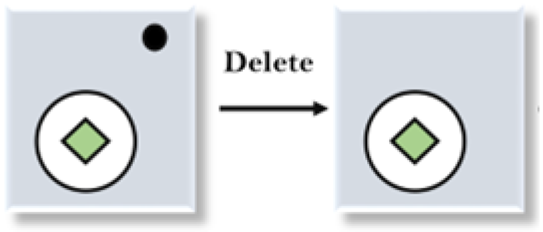
To solve the above problem, we will first find the differences between initial states and goal states, and for each difference, we will generate a new state and will apply the operators. The operators we have for this problem are:

- **Move**
- **Delete**
- **Expand**

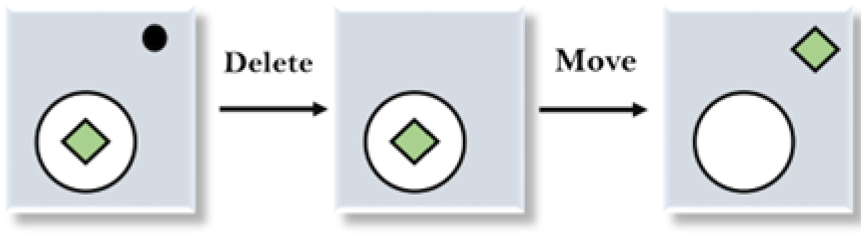
1. Evaluating the initial state: In the first step, we will evaluate the initial state and will compare the initial and Goal state to find the differences between both states.



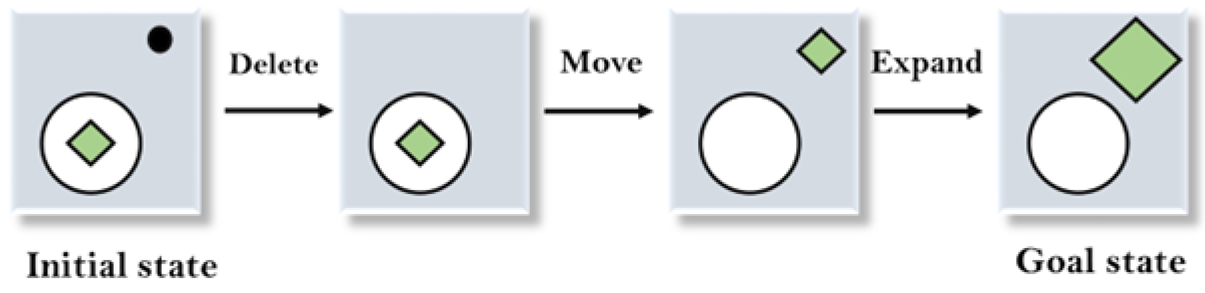
2. Applying Delete operator: As we can check the first difference is that in goal state there is no dot symbol which is present in the initial state, so, first we will apply the **Delete operator** to remove this dot.



3. Applying Move Operator: After applying the Delete operator, the new state occurs which we will again compare with goal state. After comparing these states, there is another difference that is the square is outside the circle, so, we will apply the **Move Operator**.



4. Applying Expand Operator: Now a new state is generated in the third step, and we will compare this state with the goal state. After comparing the states there is still one difference which is the size of the square, so, we will apply **Expand operator**, and finally, it will generate the goal state.



Constraint Satisfaction Problems (CSP) in Artificial Intelligence

Finding a solution that meets a set of constraints is the goal of constraint satisfaction problems (CSPs), a type of AI issue. Finding values for a group of variables that fulfill a set of restrictions or rules is the aim of constraint satisfaction problems. For tasks including resource allocation, planning, scheduling, and decision-making, CSPs are frequently employed in AI.

There are mainly three basic components in the constraint satisfaction problem:

Variables: The things that need to be determined are variables. Variables in a CSP are the objects that must have values assigned to them in order to satisfy a particular set of constraints. Boolean, integer, and categorical variables are just a few examples of the various types of variables, for instance, could stand in for the many puzzle cells that need to be filled with numbers in a sudoku puzzle.

Domains: The range of potential values that a variable can have is represented by domains. Depending on the issue, a domain may be finite or limitless. For instance, in Sudoku, the set of numbers from 1 to 9 can serve as the domain of a variable representing a problem cell.

Constraints: The guidelines that control how variables relate to one another are known as constraints. Constraints in a CSP define the ranges of possible values for variables. Unary constraints, binary constraints, and higher-order constraints are only a few examples of the various sorts of constraints. For instance, in a sudoku problem, the restrictions might be that each row, column, and 3×3 box can only have one instance of each number from 1 to 9.

Constraint Satisfaction Problems (CSP) representation:

- The finite set of variables V1, V2, V3 ……………..Vn.
- Non-empty domain for every single variable D1, D2, D3 …………..Dn.
- The finite set of constraints C1, C2 ………, Cm.
    - where each constraint Ci restricts the possible values for variables,
        - e.g., V1 ≠ V2
    - Each constraint Ci is a pair <scope, relation>
        - Example: <(V1, V2), V1 not equal to V2>
    - Scope = set of variables that participate in constraint.
    - Relation = list of valid variable value combinations.
        - There might be a clear list of permitted combinations. Perhaps a relation that is abstract and that allows for membership testing and listing.
    -

Constraint Satisfaction Problems (CSP) algorithms:

- The **backtracking algorithm** is a depth-first search algorithm that methodically investigates the search space of potential solutions up until a solution is discovered that satisfies all the restrictions. The method begins by choosing a variable and giving it a value before repeatedly attempting to give values to the other variables. The method returns to the prior variable and tries a different value if at any time a variable cannot be given a value that fulfills the requirements. Once all assignments have been tried or a solution that satisfies all constraints has been discovered, the algorithm ends.

- The **forward-checking algorithm** is a variation of the backtracking algorithm that condenses the search space using a type of local consistency. For each unassigned variable, the method keeps a list of remaining values and applies local constraints to eliminate inconsistent values from these sets. The algorithm examines a variable’s neighbors after it is given a value to see whether any of its remaining values become inconsistent and removes them from the sets if they do. The algorithm goes backward if, after forward checking, a variable has no more values.
- Algorithms for **propagating constraints** are a class that uses local consistency and inference to condense the search space. These algorithms operate by propagating restrictions between variables and removing inconsistent values from the variable domains using the information obtained

Map colouring problem

Map colouring problem states that given a graph G {V, E} where V and E are the set of vertices and edges of the graph, all vertices in V need to be coloured in such a way that no two adjacent vertices must have the same colour.

The real-world applications of this algorithm are – assigning mobile radio frequencies, making schedules, designing Sudoku, allocating registers etc.

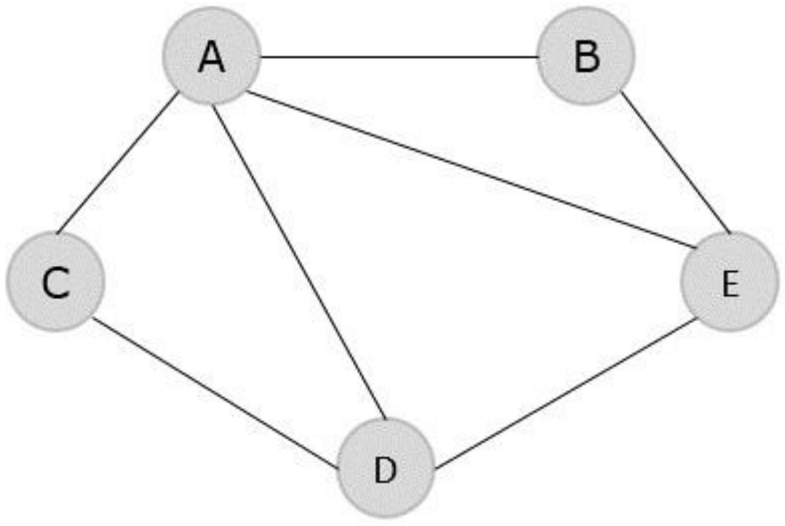
Map Colouring Algorithm

With the map colouring algorithm, a graph G and the colours to be added to the graph are taken as an input and a coloured graph with no two adjacent vertices having the same colour is achieved.

Algorithm

- Initiate all the vertices in the graph.
- Select the node with the highest degree to colour it with any colour.
- Choose the colour to be used on the graph with the help of the selection colour function so that no adjacent vertex is having the same colour.
- Check if the colour can be added and if it does, add it to the solution set.
- Repeat the process from step 2 until the output set is ready.

Examples



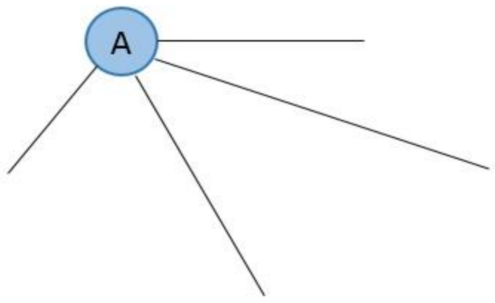
Step 1

Find degrees of all the vertices −

```
A – 4
B – 2
C – 2
D – 3
E – 3
```

Step 2

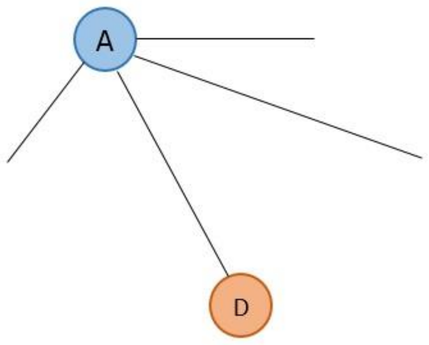
Choose the vertex with the highest degree to colour first, i.e., A and choose a colour using selection colour function. Check if the colour can be added to the vertex and if yes, add it to the solution set.



Step 3

Select any vertex with the next highest degree from the remaining vertices and colour it using selection colour function.

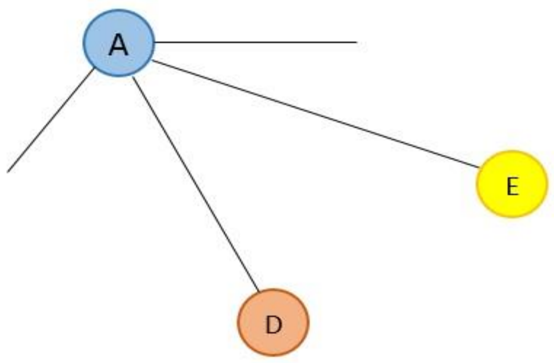
D and E both have the next highest degree 3, so choose any one between them, say D.



D is adjacent to A, therefore it cannot be coloured in the same colour as A. Hence, choose a different colour using selection colour function.

Step 4

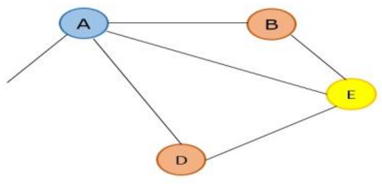
The next highest degree vertex is E, hence choose E.



E is adjacent to both A and D, therefore it cannot be coloured in the same colours as A and D. Choose a different colour using selection colour function.

Step 5

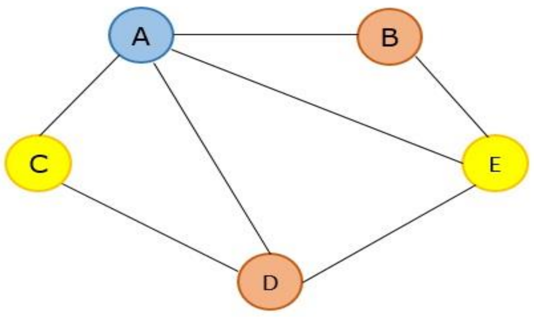
The next highest degree vertices are B and C. Thus, choose any one randomly.



B is adjacent to both A and E, thus not allowing to be coloured in the colours of A and E but it is not adjacent to D, so it can be coloured with D's colour.

Step 6

The next and the last vertex remaining is C, which is adjacent to both A and D, not allowing it to be coloured using the colours of A and D. But it is not adjacent to E, so it can be coloured in E's colour.



Example

Following is the complete implementation of Map Colouring Algorithm in various programming languages where a graph is coloured in such a way that no two adjacent vertices have same colour.

Open Compiler

```
#include<stdio.h>
#include<stdbool.h>
#define V 4
bool graph[V][V] = {
  {0, 1, 1, 0},
  {1, 0, 1, 1},
  {1, 1, 0, 1},
  {0, 1, 1, 0},
};
bool isValid(int v,int color[], int c){   //check whether putting a color valid for v
  for (int i = 0; i < V; i++)
    if (graph[v][i] && c == color[i])
      return false;
  return true;
}
bool mColoring(int colors, int color[], int vertex){
  if (vertex == V) //when all vertices are considered
    return true;
  for (int col = 1; col <= colors; col++) {
    if (isValid(vertex,color, col)) { //check whether color col is valid or not
      color[vertex] = col;
      if (mColoring (colors, color, vertex+1) == true) //go for additional vertices
        return true;
      color[vertex] = 0;
    }
  }
  return false; //when no colors can be assigned
}
int main(){
  int colors = 3; // Number of colors
  int color[V]; //make color matrix for each vertex
  for (int i = 0; i < V; i++)
    color[i] = 0; //initially set to 0
  if (mColoring(colors, color, 0) == false) { //for vertex 0 check graph coloring
    printf("Solution does not exist.");
  }
  printf("Assigned Colors are: \n");
  for (int i = 0; i < V; i++)
    printf("%d ", color[i]);
  return 0;
}
```

Output

```
Assigned Colors are:
1 2 3 1
```