Introduction to AI Mini-Project

-

Agricultural Plan Optimization

Team Members:

MEFTAH Zineb (Leader)

BENAMGHAR Amina

DJOUBANI Sarah

BENMANSOUR Aya

Introduction

Agricultural production is a cornerstone of Algeria's economy and a vital aspect of its food security. Given the country's diverse climatic conditions and rich agricultural heritage, optimizing agricultural production is essential to meet the demands of its growing population and reduce dependency on imports.

Data

This report is based on an extensive dataset detailing agricultural production across various wilayas in our region. The data file, which can be accessed here, encompasses information on available land, production efficiency, and cost structures for key agricultural products.

Problem Definition:

This project aims to develop a strategic plan to enhance agricultural productivity, maximize production, and achieve self-sufficiency in key agricultural products in Algeria.To achieve these objectives, it is essential to clearly define key terminologies:

1. Highest Production

Highest production refers to the goal of maximizing the outcomes of agricultural products in terms of quantity and yield. This involves:

- **Resource Efficiency**: Utilizing agricultural inputs in the most effective way, ensuring Wilayas are motivated for optimal outcomes best suited to the climatic and soil conditions.

2. Lowest Price

Lowest price refers to the objective of minimizing the cost of agricultural products for consumers while ensuring fair compensation for producers. This involves:

- **Price Stability**: Ensuring consistent prices throughout the season and offseason throughout effective management of production in line with consumption.
- **Seasonal Management**: Adjusting production and distribution strategies based on seasonal price fluctuations to maintain affordability for consumers and profitability for producers

3. Self-Sufficiency

Self-sufficiency refers to the capacity of Algeria to meet its own needs for essential agricultural products without relying on import by Encouraging the adoption of sustainable agricultural methods in suitable land

General Graph Search

Problem formulation :

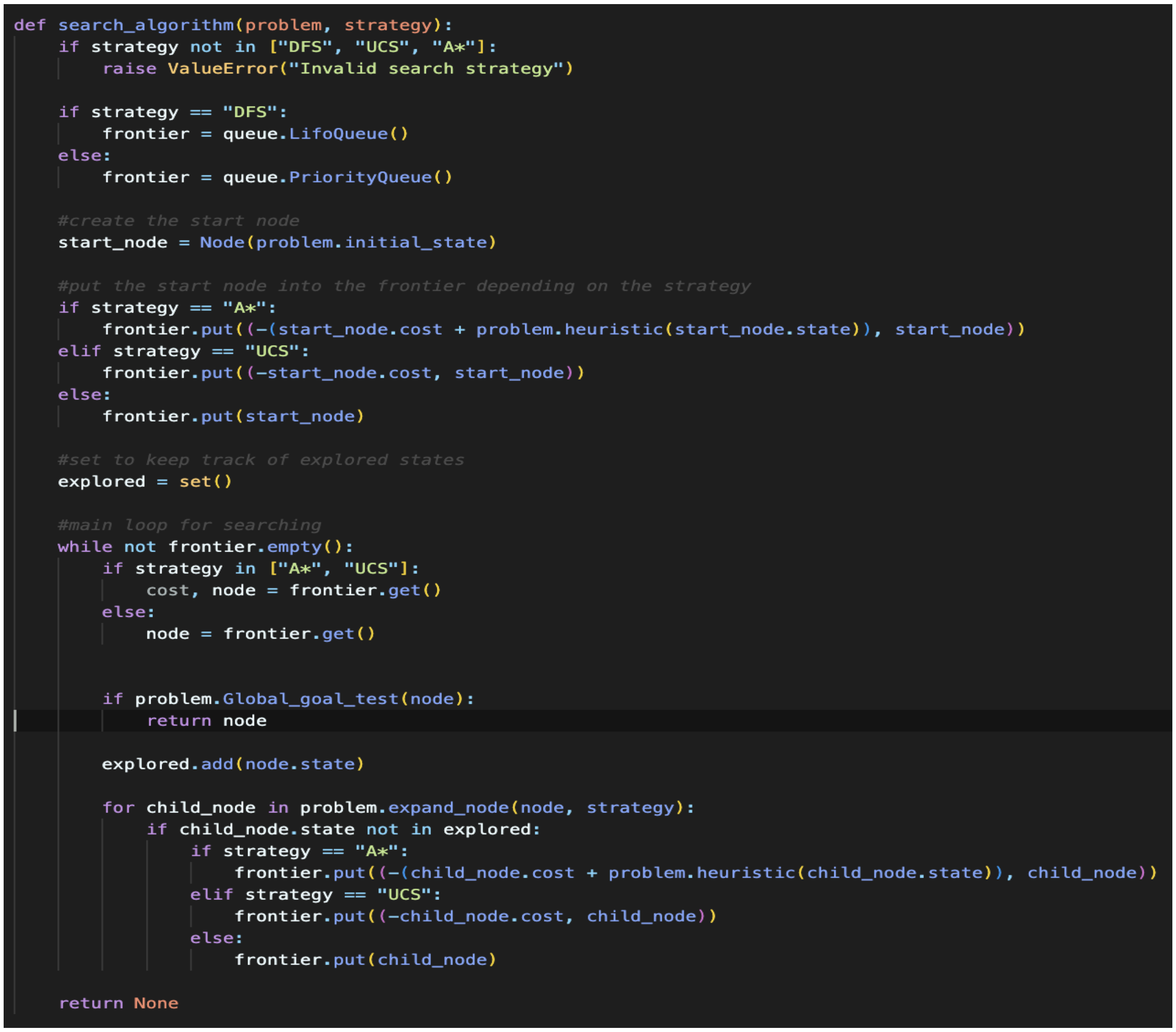
- **Initial State:**
    - The initial state represents an **empty strategic plan** for the agricultural sector in Algeria.
    - Each Wilaya (administrative region) is listed along with its **unexploited lands**.
    - For each Wilaya and each type of agricultural product, the potential **yield** per hectare is specified.
    - Initial **prices** for each product are set to 0.
    - Initial **self-sufficiency** ratio for each product is set to 0
    - Initial **production** for each product is set to 0.
    - **Land usage** for each product in each Wilaya is set to 0.
    - **Total unexploited land** across all wilayas is calculated, assuming **none of the lands are utilized yet.**

Practically, our initial state is an object from the class **Plan** that includes   the previously cited information.

- **Transition Model:**
    - Applying an action at a certain **parent state** results in another **state**, representing a **modified strategic plan**.
    - Only one product's coordinates in a specific Wilaya changes with each action.
    - The possible children that a parent **plan** can have are plans in which each of them contains an improvement on a specific product that didn't satisfy the constraints yet, this means that the **branching factor** is **eight** , and the number of children will decrease through time.
- **Goal Test:**
    - The primary goal is to ensure self-sufficiency in all agricultural products.
    - If self-sufficiency is achieved, the secondary goal is to identify additional production potential by exploiting the maximum amount of unutilized land within a specified price interval set as a threshold to respect supply and demand rules.
- **Path Cost:**
    - The path cost is the total cost incurred in producing each product, considering specific factors such as labor, resources, and infrastructure.
    - It accounts for the cost associated with utilizing land for agricultural purposes.
- **Actions:**
    - Actions involve the exploitation of lands in wilayas suitable for specific agricultural products.
    - Each action expands the production area for one product in exactly a chosen
    - Wilayas by a specific amount.
- **State Space:**

- The state space encompasses all possible configurations of the strategic plan.
- It includes combinations of production levels for each product in each Wilaya.
- The state space dynamically changes as actions are applied, reflecting the evolving agricultural landscape of Algeria.

Search Strategies



```python
def search_algorithm(problem, strategy):
    if strategy not in ["DFS", "UCS", "A*"]:
        raise ValueError("Invalid search strategy")

    if strategy == "DFS":
        frontier = queue.LifoQueue()
    else:
        frontier = queue.PriorityQueue()

    #create the start node
    start_node = Node(problem.initial_state)

    #put the start node into the frontier depending on the strategy
    if strategy == "A*":
        frontier.put((-(start_node.cost + problem.heuristic(start_node.state)), start_node))
    elif strategy == "UCS":
        frontier.put((-start_node.cost, start_node))
    else:
        frontier.put(start_node)

    #set to keep track of explored states
    explored = set()

    #main loop for searching
    while not frontier.empty():
        if strategy in ["A*", "UCS"]:
            cost, node = frontier.get()
        else:
            node = frontier.get()


        if problem.Global_goal_test(node):
            return node

        explored.add(node.state)

        for child_node in problem.expand_node(node, strategy):
            if child_node.state not in explored:
                if strategy == "A*":
                    frontier.put((-(child_node.cost + problem.heuristic(child_node.state)), child_node))
                elif strategy == "UCS":
                    frontier.put((-child_node.cost, child_node))
                else:
                    frontier.put(child_node)

    return None
```

In our search algorithm, the node expansion process follows a specific logic centered around planting products in suitable regions:

Node Expansion:

- Each action represents planting one of eight possible products.
- Only actions where the product can be planted in available lands within suitable Wilayas (regions) are considered.
- If a product cannot be planted due to lack of suitable land, that action is not expanded into a new node for the sake of optimizing the search.

Choosing the Wilaya:

- For each valid action, the product is planted in the most suitable Wilaya.
- The selection of the Wilaya is based on a formula that considers the current state parameters of the node.

Goal Condition:

- The search continues, expanding nodes and planting products, until it finds a goal node.
- The goal node satisfies all the constraints of the problem, such as land availability and suitability for each product.

This approach ensures that only feasible actions are pursued, guiding the search towards a solution that meets the problem's requirements.As well as using the searching algorithms.

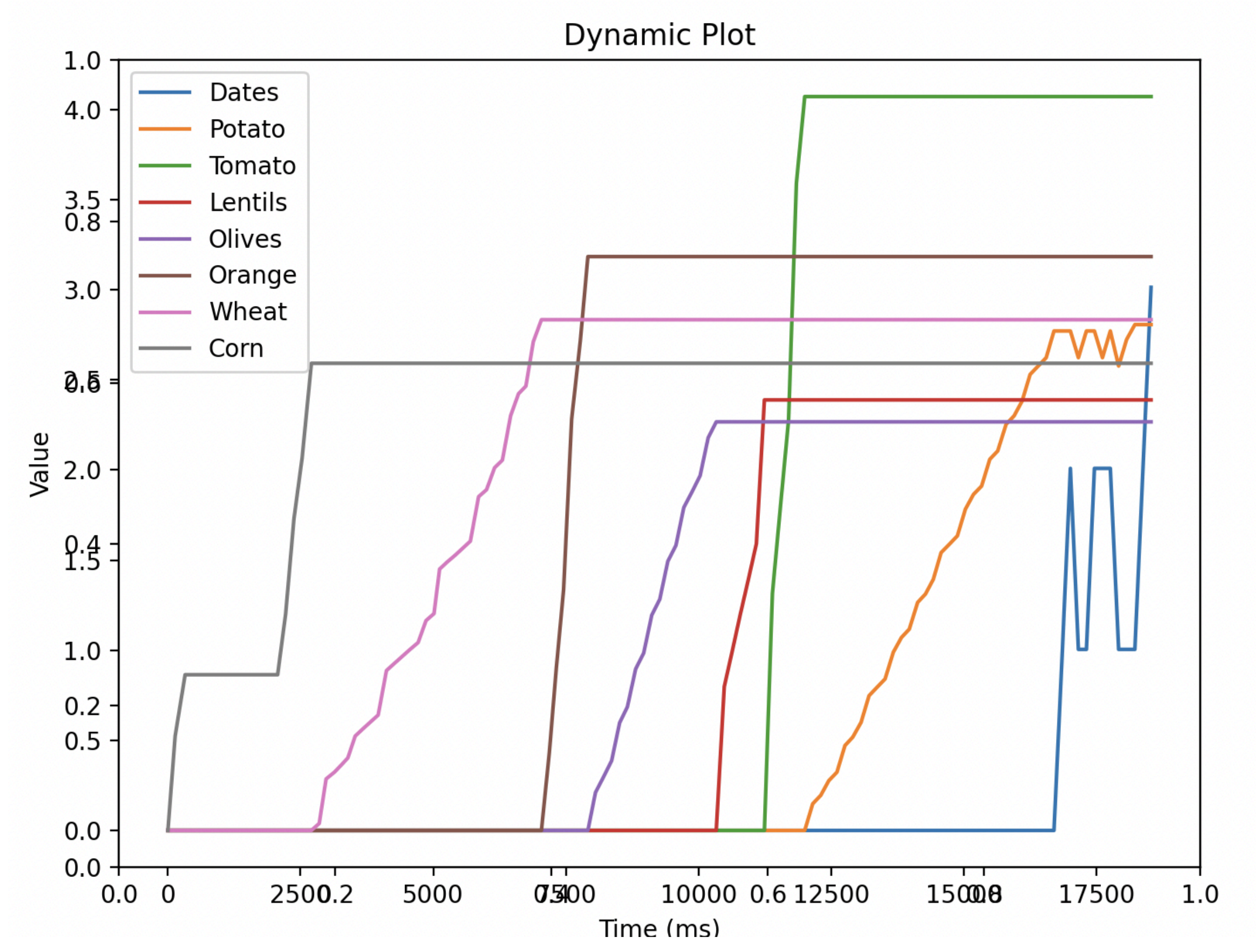
We used a **general search function**, that included the searching algorithms: **DFS, UCS, A*.**

Uninformed Search

Depth First Search:

explores as deeply as possible along each branch of a tree or graph before backtracking.It employs a stack (LIFO) to maintain a record of nodes awaiting exploration and is frequently implemented using recursion. While not always optimal, DFS is useful for exploring deeply into a graph and can efficiently find one solution or explore all possible solutions.

DEMO Graph DFS for self-sufficiency:



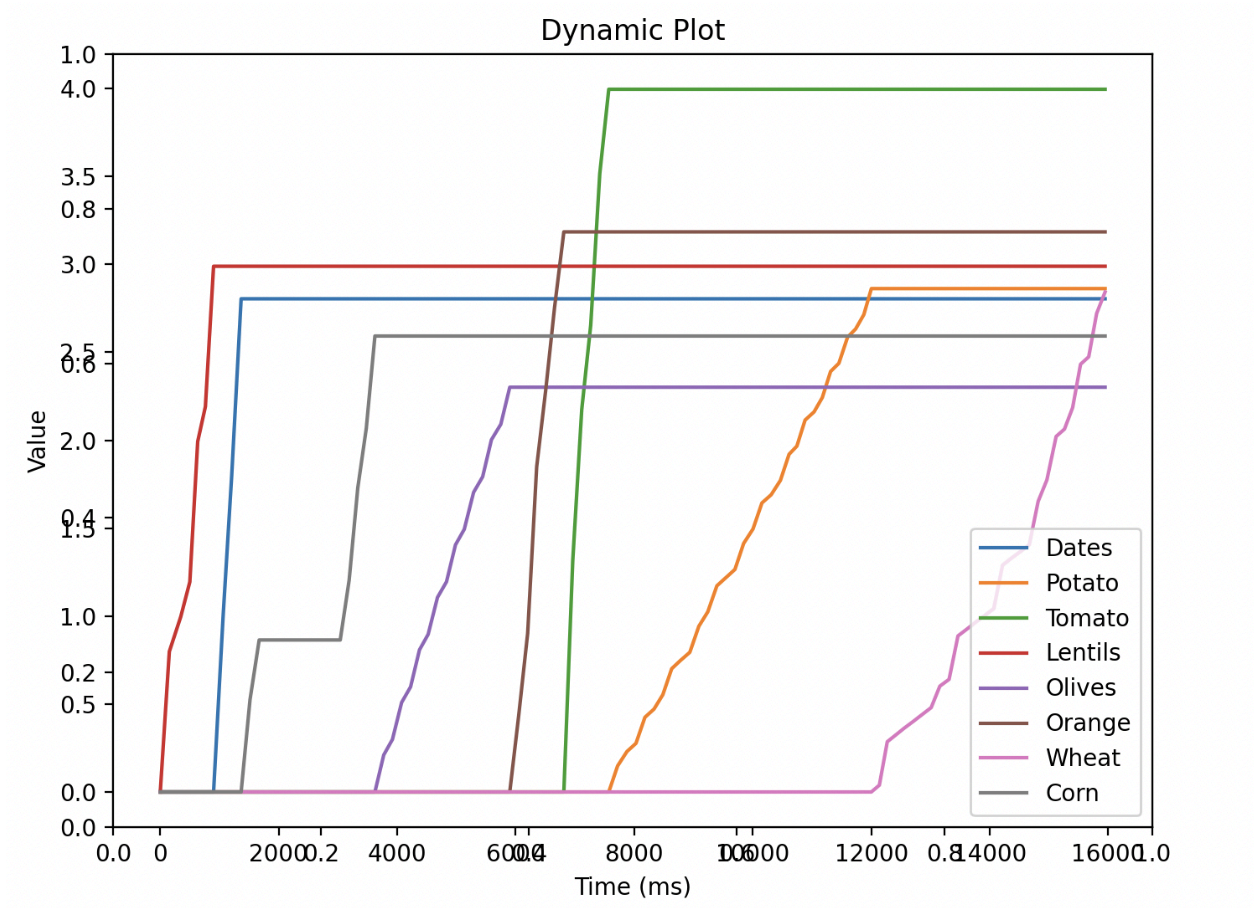
Uniform Cost Search :

Finds the cheapest path from a starting node to a goal node. It explores paths based on their cost (depending on the cost function), using a priority queue to prioritize nodes with the lowest path cost. UCS guarantees finding the optimal solution when edge costs are non-negative.

The cost function:

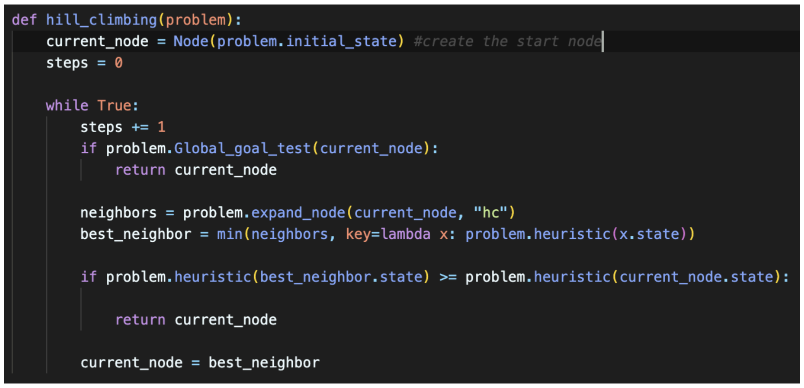
The cost function essentially calculates a cost metric based on production efficiency. The cost is inversely proportional to the total production of the product in the suitable wilayas. If the production is high, the cost will be low, indicating that it is efficient to produce the product in those wilayas. Conversely, if the production is low, the cost will be high, indicating that it is inefficient to take that path.

DEMO Graph UCS for self-sufficiency:



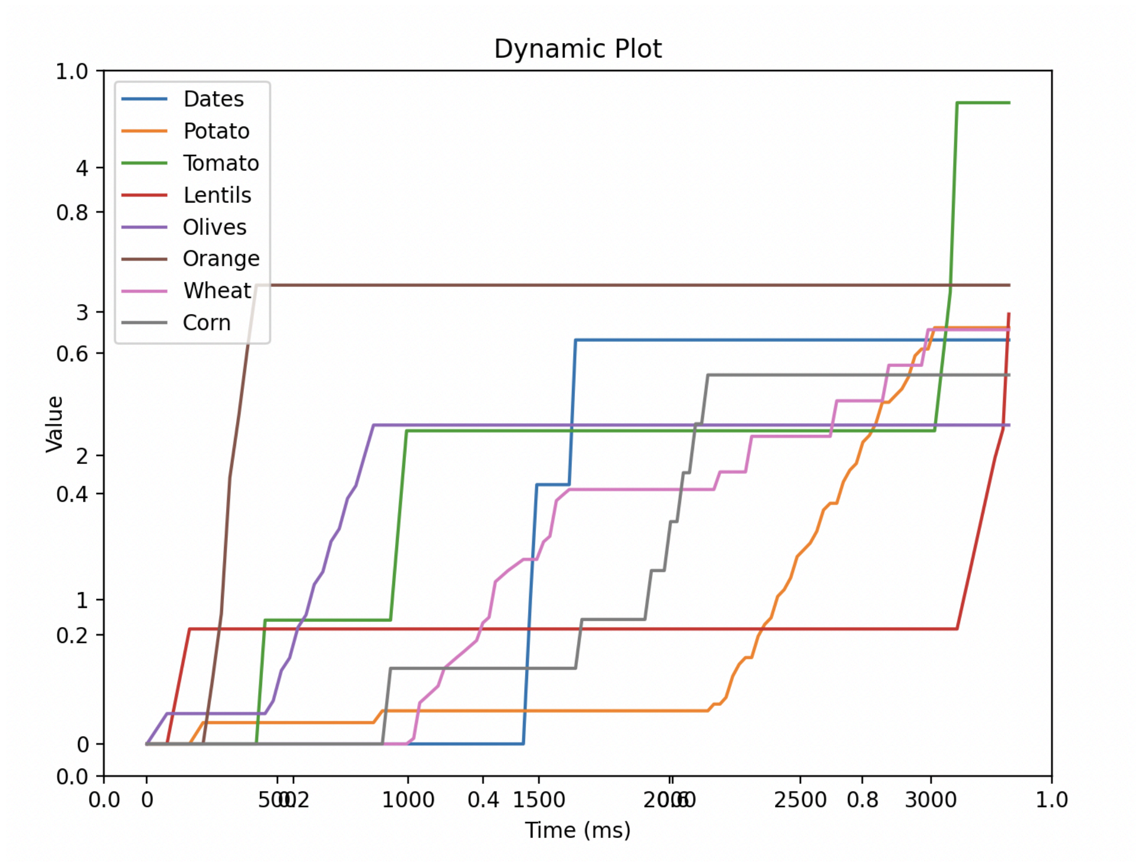
Informed Search

Hill Climbing:


```
def hill_climbing(problem):
    current_node = Node(problem.initial_state) #create the start node
    steps = 0

    while True:
        steps += 1
        if problem.Global_goal_test(current_node):
            return current_node

        neighbors = problem.expand_node(current_node, "hc")
        best_neighbor = min(neighbors, key=lambda x: problem.heuristic(x.state))

        if problem.heuristic(best_neighbor.state) >= problem.heuristic(current_node.state):

            return current_node

        current_node = best_neighbor
```

is a local search algorithm that iteratively improves a solution by moving to neighboring solutions with higher values according to a heuristic function. It terminates when it reaches a peak where no better neighbor exists, even though it's quick but can get stuck in local optima.

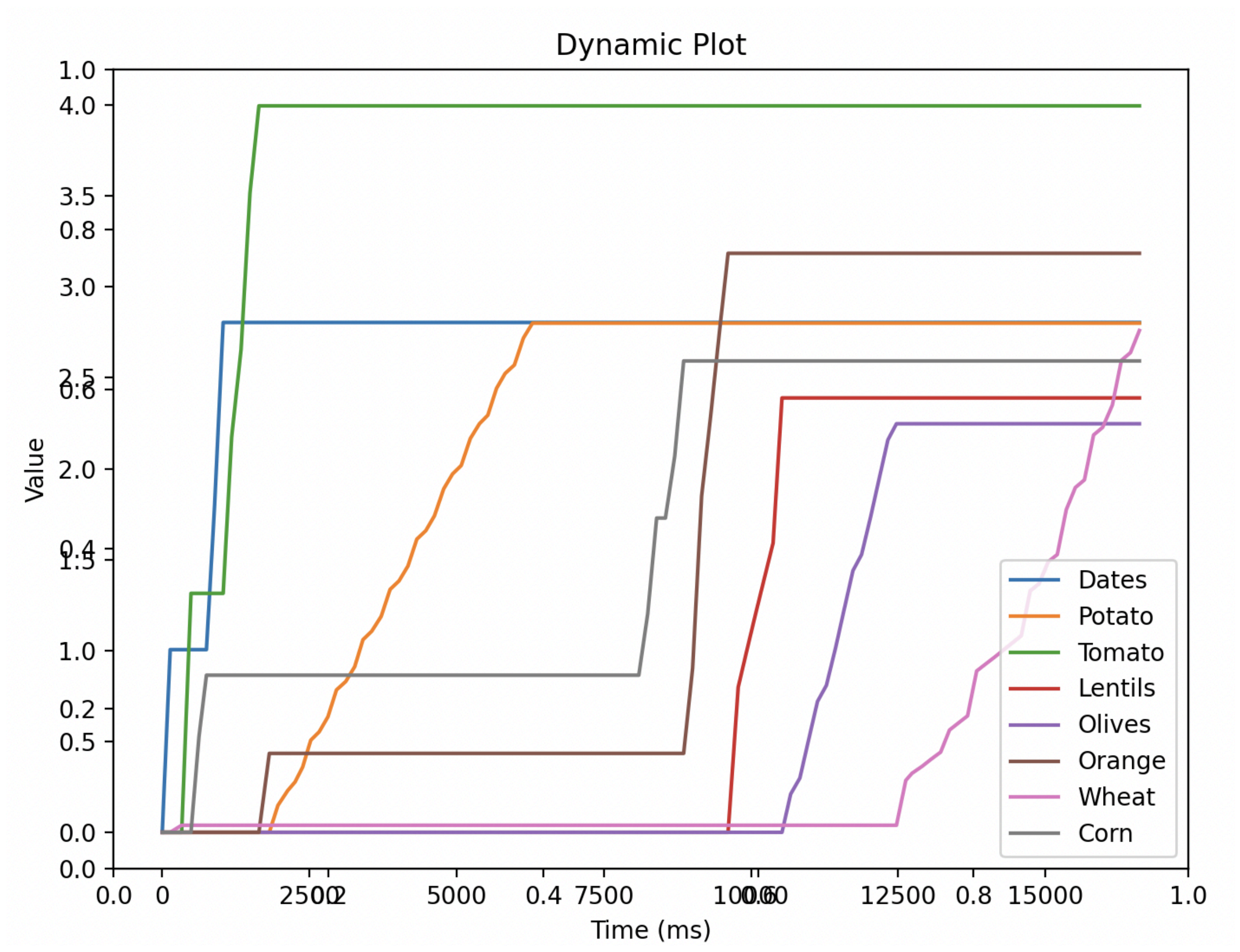
DEMO Graph Hill Climbing for self-sufficiency:



A*:

Search finds the shortest path from a start node to a goal node. It prioritizes nodes based on a combination of their actual cost from the start + estimated cost to the goal. A* guarantees finding the optimal solution under certain conditions.

DEMO Graph A* for self-sufficiency:



The heuristic Function:

The heuristic function aims to provide a comprehensive evaluation of the agricultural state's current status by considering both land usage efficiency and product self-sufficiency. It makes use of two heuristics:

- **Self sufficiency heuristic:** which evaluates the state's ability to meet its agricultural needs based on the self-sufficiency ratios of various products.
- **Land heuristic:** which computes the percentage of land that is still available for agricultural use by comparing the current unused land to the initial total land.
- The results of the two heuristics are summed to provide a single heuristic value **Admissibility**

The heuristic function is admissible since it is based on two individual heuristics that themselves are admissible. It does not overestimate the efficiency or self-sufficiency of the state, ensuring accurate and reliable guidance for decision-making in agricultural planning.

Comparative Table (time (ms) , Nodes explored):

	Time (ms)	Nodes explored	Comparison
DFS	72.37	124	● The least efficient algorithm between these algorithms, taking the most time and exploring the most nodes. DFS does not consider path cost or heuristics (blindly), leading to potentially longer paths and more nodes explored
UCS	55.26	107	● It is the second most efficient. It finds the least-cost path without considering heuristics, which might explain the higher number of nodes explored compared to A*.
A*	49.99	100	● The most efficient both in terms of time and the nodes explored. It combines both heuristics and path cost, which likely contributes to its efficiency
Hill Climbing	66.48	121	● This algorithm performs moderately well. It uses a heuristic to choose the next node, but can get stuck in local optima, which explains its relatively higher time and node count.

Comparison between search algorithms :

- **A*** is the best choice for efficiency in both time and nodes explored.
- **UCS** is a close second, especially in case the heuristic information is not available.
- **Hill Climbing** can be effective but not reliable as A* or UCS.
- **DFS** is the least efficient and is generally not suitable (big space searches) for optimal pathfinding when compared to the other algorithms provided.

Constraint Satisfaction Problem

Problem formulation:

Domains: Are all the possible planned coordinates for each product. Each value is represented as follows:

Value

Price: (Season Price, OffSeason Price)

Production: production amount

Land:

Wilaya number 1 :land used

Wilaya number n : land used

Self-sufficiency rate: [0, +∞]

Variables:

consist of the eight strategic products :

Dates, Potato, Tomato, Orange, Wheat, Corn, Olives, Lentils

Constraints:

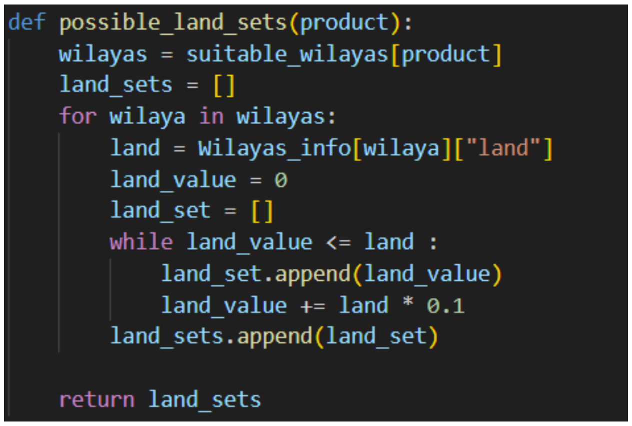
Self-sufficiency constraint: Self-sufficiency ratio must be equal to or greater than 1

Maximizing production with respect to a threshold as an interval of prices

Main Methods Explanation:

1- possible_land_sets:

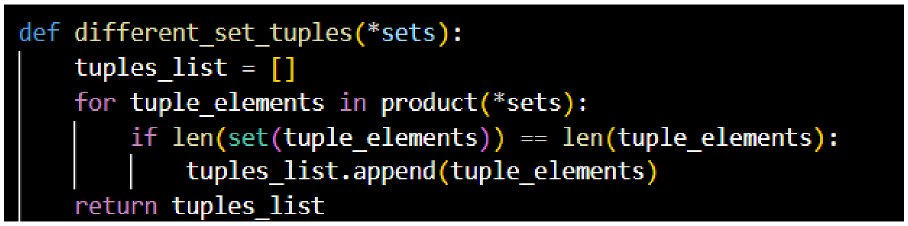
This function generates a set, containing **sets of possible land values** of a product, for each Wilaya that the product can grow in , we generate a single set having values from 0 to the total land size , the values are incrementing by 10% from the total land size **(e.g: S = { 0, 19 , 38, 57, 76, 95, 114, 133, 152, 171, 190 }).**



```
def possible_land_sets(product):
    wilayas = suitable_wilayas[product]
    land_sets = []
    for wilaya in wilayas:
        land = Wilayas_info[wilaya]["land"]
        land_value = 0
        land_set = []
        while land_value <= land :
            land_set.append(land_value)
            land_value += land * 0.1
        land_sets.append(land_set)

    return land_sets
```

2- different_set_tuples

This function creates tuples containing all combinations from the elements of sets passed as parameters in which the order does not matter:

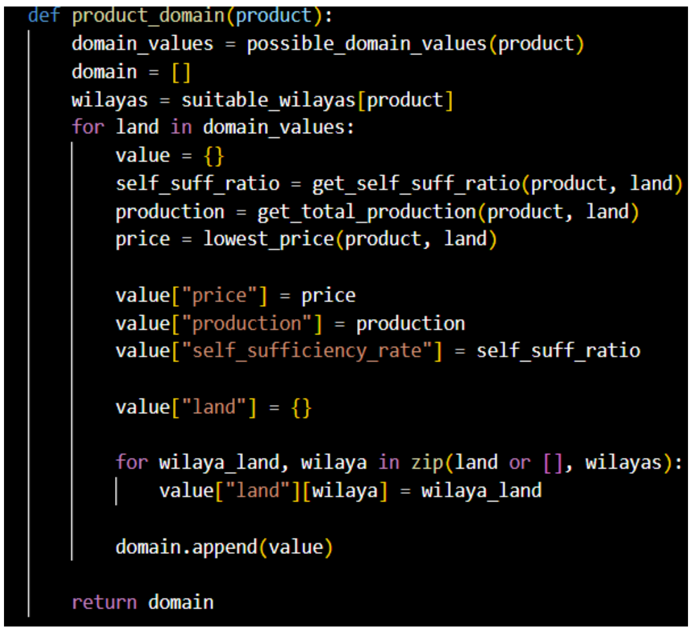


```
def different_set_tuples(*sets):
    tuples_list = []
    for tuple_elements in product(*sets):
        if len(set(tuple_elements)) == len(tuple_elements):
            tuples_list.append(tuple_elements)
    return tuples_list
```

3- Possible_domain_values:

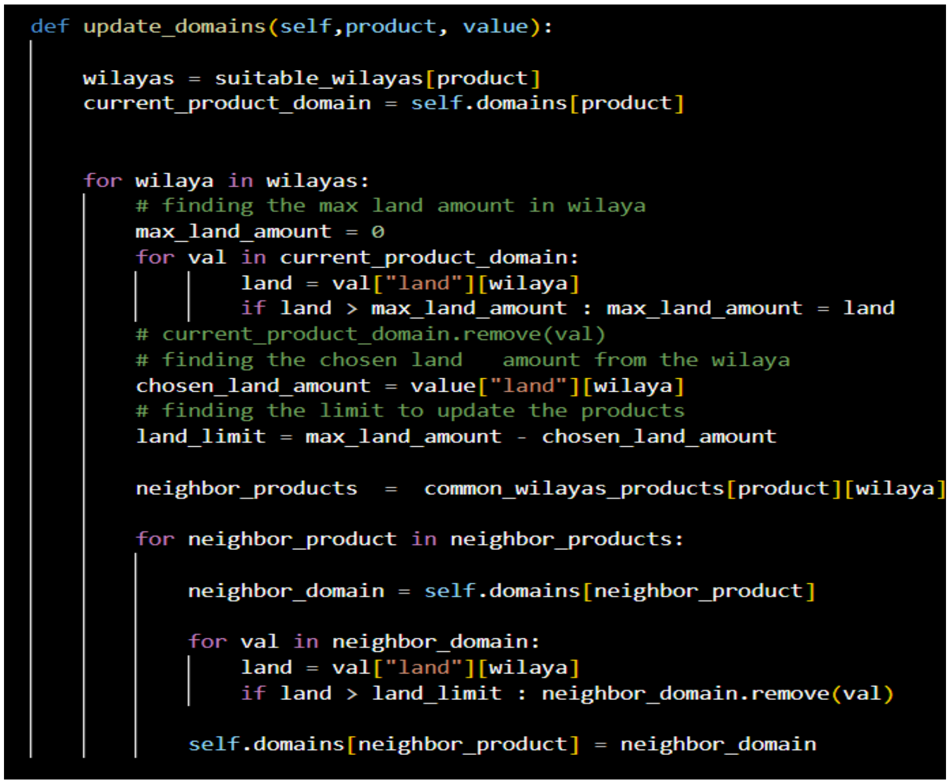
The possible_domain_values function combines the functionality of **possible_land_sets** and **different_set_tuples** to generate all possible tuples of land values for a given product, ensuring that each tuple contains unique values

4-Product_domain: The product_domain function generates a detailed domain for a given product. This domain includes various potential configurations of **land** usage across different wilayas along with **price**, **production**, and **self-sufficiency** rate

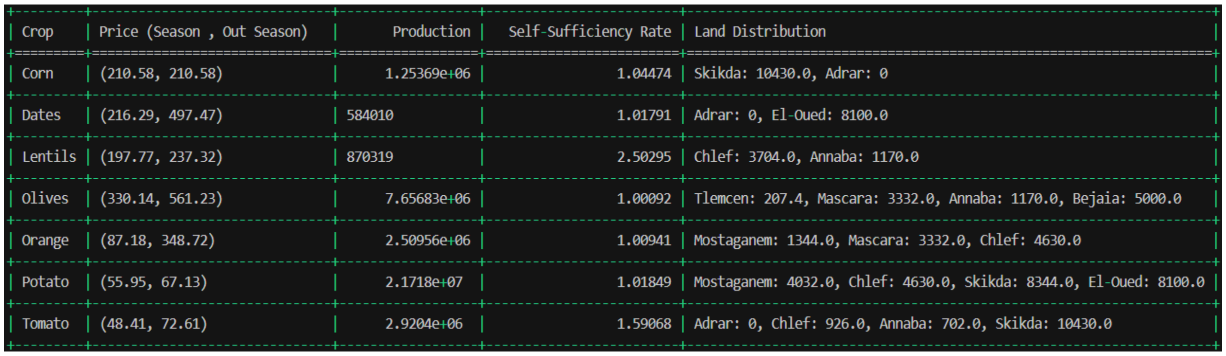

```python
def product_domain(product):
    domain_values = possible_domain_values(product)
    domain = []
    wilayas = suitable_wilayas[product]
    for land in domain_values:
        value = {}
        self_suff_ratio = get_self_suff_ratio(product, land)
        production = get_total_production(product, land)
        price = lowest_price(product, land)

        value["price"] = price
        value["production"] = production
        value["self_sufficiency_rate"] = self_suff_ratio

        value["land"] = {}

        for wilaya_land, wilaya in zip(land or [], wilayas):
            value["land"][wilaya] = wilaya_land

        domain.append(value)

    return domain
```

Update_domains Method:

Function that updates the domains of neighboring products by removing values that exceed the land limit.based on the current assignment of values to the product (**inferencing**).

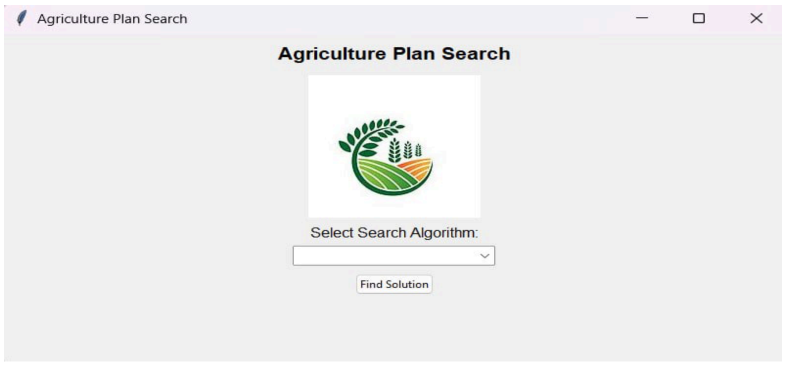

```python
def update_domains(self,product, value):

    wilayas = suitable_wilayas[product]
    current_product_domain = self.domains[product]


    for wilaya in wilayas:
        # finding the max land amount in wilaya
        max_land_amount = 0
        for val in current_product_domain:
                land = val["land"][wilaya]
                if land > max_land_amount : max_land_amount = land
        # current_product_domain.remove(val)
        # finding the chosen land   amount from the wilaya
        chosen_land_amount = value["land"][wilaya]
        # finding the limit to update the products
        land_limit = max_land_amount - chosen_land_amount

        neighbor_products  =  common_wilayas_products[product][wilaya]

        for neighbor_product in neighbor_products:

            neighbor_domain = self.domains[neighbor_product]

            for val in neighbor_domain:
                land = val["land"][wilaya]
                if land > land_limit : neighbor_domain.remove(val)

            self.domains[neighbor_product] = neighbor_domain
```

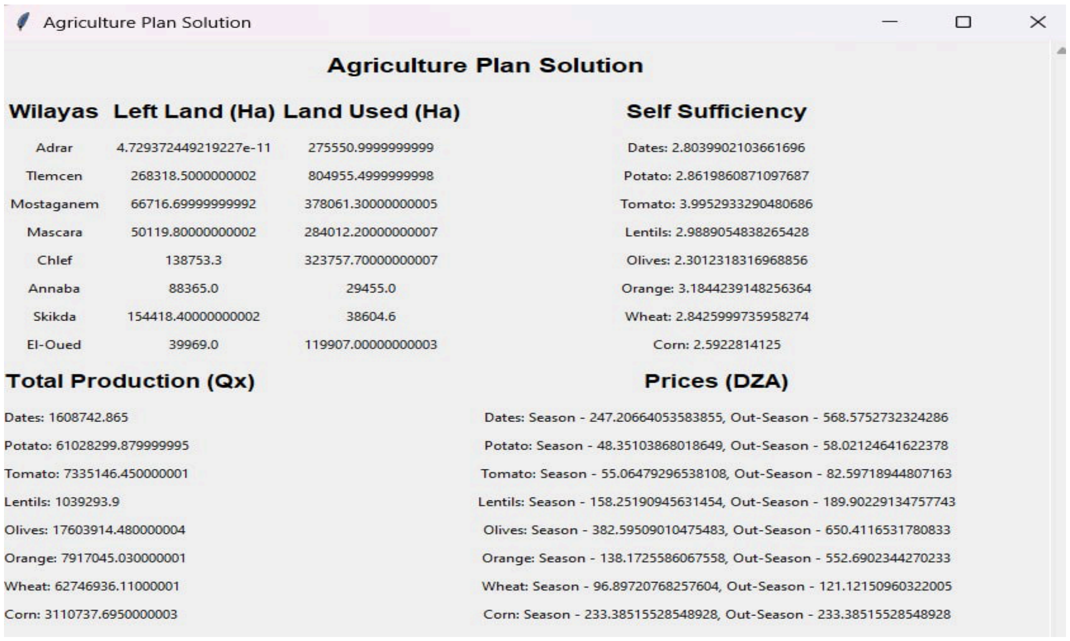
CSP’s Solution Demonstration: this solution is found by the assignment of values to all products using **backtrack method**



```
+---------+-----------------------------+---------------+-----------------------+--------------------------------------------------------------------+
| Crop    | Price (Season , Out Season) |    Production | Self-Sufficiency Rate | Land Distribution                                                  |
+=========+=============================+===============+=======================+====================================================================+
| Corn    | (210.58, 210.58)            |   1.25369e+06 |               1.04474 | Skikda: 10430.0, Adrar: 0                                          |
+---------+-----------------------------+---------------+-----------------------+--------------------------------------------------------------------+
| Dates   | (216.29, 497.47)            | 584010        |               1.01791 | Adrar: 0, El-Oued: 8100.0                                          |
+---------+-----------------------------+---------------+-----------------------+--------------------------------------------------------------------+
| Lentils | (197.77, 237.32)            | 870319        |               2.50295 | Chlef: 3704.0, Annaba: 1170.0                                      |
+---------+-----------------------------+---------------+-----------------------+--------------------------------------------------------------------+
| Olives  | (330.14, 561.23)            |   7.65683e+06 |               1.00092 | Tlemcen: 207.4, Mascara: 3332.0, Annaba: 1170.0, Bejaia: 5000.0    |
+---------+-----------------------------+---------------+-----------------------+--------------------------------------------------------------------+
| Orange  | (87.18, 348.72)             |   2.50956e+06 |               1.00941 | Mostaganem: 1344.0, Mascara: 3332.0, Chlef: 4630.0                 |
+---------+-----------------------------+---------------+-----------------------+--------------------------------------------------------------------+
| Potato  | (55.95, 67.13)              |   2.1718e+07  |               1.01849 | Mostaganem: 4032.0, Chlef: 4630.0, Skikda: 8344.0, El-Oued: 8100.0 |
+---------+-----------------------------+---------------+-----------------------+--------------------------------------------------------------------+
| Tomato  | (48.41, 72.61)              |   2.9204e+06  |               1.59068 | Adrar: 0, Chlef: 926.0, Annaba: 702.0, Skikda: 10430.0             |
+---------+-----------------------------+---------------+-----------------------+--------------------------------------------------------------------+
```

Visualization of the result :





Agriculture Plan Solution

Agriculture Plan Solution

Wilayas	Left Land (Ha)	Land Used (Ha)
Adrar	4.729372449219227e-11	275550.9999999999
Tlemcen	268318.5000000002	804955.4999999998
Mostaganem	66716.69999999992	378061.30000000005
Mascara	50119.80000000002	284012.20000000007
Chlef	138753.3	323757.70000000007
Annaba	88365.0	29455.0
Skikda	154418.40000000002	38604.6
El-Oued	39969.0	119907.00000000003

Self Sufficiency

Dates: 2.8039902103661696
Potato: 2.8619860871097687
Tomato: 3.9952933290480686
Lentils: 2.9889054838265428
Olives: 2.3012318316968856
Orange: 3.1844239148256364
Wheat: 2.8425999735958274
Corn: 2.5922814125

Total Production (Qx)

Dates: 1608742.865
Potato: 61028299.879999995
Tomato: 7335146.450000001
Lentils: 1039293.9
Olives: 17603914.480000004
Orange: 7917045.030000001
Wheat: 62746936.11000001
Corn: 3110737.6950000003

Prices (DZA)

Dates: Season - 247.20664053583855, Out-Season - 568.5752732324286
Potato: Season - 48.35103868018649, Out-Season - 58.02124641622378
Tomato: Season - 55.06479296538108, Out-Season - 82.59718944807163
Lentils: Season - 158.25190945631454, Out-Season - 189.90229134757743
Olives: Season - 382.59509010475483, Out-Season - 650.4116531780833
Orange: Season - 138.1725586067558, Out-Season - 552.6902344270233
Wheat: Season - 96.89720768257604, Out-Season - 121.12150960322005
Corn: Season - 233.38515528548928, Out-Season - 233.38515528548928

Contribution of team members in the project :

In the project ,all team members made significant contributions.They were involved in every aspect of the project ,demonstrating teamwork and collaboration.Regular meetings were planned each Tuesday at the school to discuss the progress , solve issues , and work . The collective effort of the team members played a crucial role in the project's success.

MEFTAH ZINEB	DATA - SEARCH - CSP
BENAMGHAR AMINA	DATA - SEARCH - CSP
DJOUBANI SARAH	DATA - SEARCH - CSP
BENMANSOUR AYA	DATA - SEARCH - CSP

Appendix and some references:

- For the prices to be real, we have managed to contact a farmer - Bachir Kaouachi - .
- https://weatherandclimate.com/algeria
- https://www.populationdata.net/pays/algerie/divisions
- https://www.climatsetvoyages.com/climat/algerie
- https://madr.gov.dz/wp-content/uploads/2022/04/SERIE-B-2019.pdf