Date: 07 November, 2017

Algorithm And Computability Project

Features and Experts Project

Filip Matracki

Album Nr: 268956

Mateusz Grossman

Album Nr: 245490

Fardin Mohammed

Album Nr: 273190

Table of Contents

Introduction

Features and Experts project is an optimisation allocation problem with the use of maximal resources given and with minimal loss of such resources.

The problem of the project consists of three main elements:

1. Number of features(F) : $F \in N$
2. A list of projects(P): $P = \{p_{i \in N} : p_i$ is a list of natural numbers $\}$.
3. A list of experts(E): $E = \{e_{i \in N} : e_i$ is a list of binary numbers $\}$

Aim of this project is to implement an algorithm that allocate experts to projects in such an optimised way that maximum numbers of projects are finished with the least amount of experts left as residue.

Such problems can be solved in two ways:

1. Maximise the use of experts and do less projects than intended or,
2. Use experts smartly and do more projects but leave some experts as residue.

Our implementation will be aimed to use experts smartly and do maximum number of projects and maybe leave some experts as residue depending on input data .

Description

Features are defined as a skill needed by an expert to complete a part of a project.

Experts are defined as a list of all the expert arrays where each expert array is a binary array. Each element on an index of an expert array indicates whether the current expert is an expert in that indexed feature or not.

```
Obtained experts: [[0, 1, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0]]
```

Projects are defined as a list of all the project arrays where each project array is an array of natural numbers. Each element on an index of a project array states the amount of experts in that index feature that will be needed to complete such project.

```
Obtained projects: [[0, 1, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0], [0, 3, 0, 0, 0, 0]]
```

For example :

To complete the third project in the above images we will be needing all the experts of the example experts in the image above but we can finish two projects and leave one expert as a residue.

So, brute force implementation of such an algorithm will be to find all the permutations of experts and projects. Loop through all of the combinations, find every possible solution and select the one that is the best fit.

Pseudocode

Brute Force Method:

```
expert_indices = [list of all the indices of experts]
project_indices = [list of all the indices of projects]
expert_combinations = [list of all the permutations of experts]

for length in range(1,length of projects + 1):
        for subset in all_the_permutations_of_project:
                project_combinations.append(subsets)

for expert_combination in expert_combinations:
        for project_combination in project_combinations:
                current_experts = list of all expert permutations
                current_projects = list of all project permutations

                expert_idx = 0
                experts_used = []
                experts_used = []
                for expert in current_experts:
                        expert_was_used = False
                        project_idx = 0
                        for project in current_projects:
                                if not expert_was_used:
                                        for j in range of number_of_features :
                                                if project[j] > 0 and expert[j] > 0:
                                                        project[j] -= 1
                                                        expert_was_used = True
                                                        experts_used.append(expert_combination[expert_idx])
                                                        if all( [val == 0 for val in project]):
                                                                projects_solved.append(project_combination[project_idx])
                                                        Break
                                        project_idx += 1
                        expert_idx += 1
                solutions.append(current solution)

For solution in solutions:
        best_solution = choose the best solution

print(best_solution)
```

Choice of the Input Data

Input data for the algorithm is supplied by a Comma Separated File in the below format :

<Number of Project>,<Number of Experts>,<Number of Features>
<List of Projects with each project separated by an end of line character>
<List of Experts with each expert separated by an end of line character>

For example:

3,3,6
0,1,0,0,0,0
0,1,0,0,0,0
0,3,0,0,0,0
0,1,0,0,0,0
0,1,0,0,0,0
0,1,0,0,0,0

This file is parsed and supplied to the program as :

1. Number of projects.
2. Number of experts.
3. Number of features.
4. Projects as list of arrays consisting of natural numbers.
5. Experts as a list of arrays consisting of binary numbers.

Implementation of the Algorithm

Brute Force Algorithm:

```
# Loop through every combination of experts and projects and find a solution
for expert_combination in self.expert_combinations:
    for project_combination in self.project_combinations:
        # Copy list of experts and projects to a temp variable so we can distinguish between solutions
        current_experts = list([list(self.experts_as_lists[i]) for i in expert_combination])
        current_projects = list([list(self.projects_as_lists[i]) for i in project_combination])
        # print("Current Projects: " + str(current_projects))
        # print("Current experts: " + str(current_experts))
        expert_idx = 0
        experts_used = []
        projects_solved = []
        for expert in current_experts:
            expert_was_used = False
            project_idx = 0
            # Try to apply expert to all projects
            for project in current_projects:
                if not expert_was_used:
                    for j in range(self.number_of_features):
                        if project[j] > 0 and expert[j] > 0:
                            project[j] -= 1
                            # expert[j] = 0  # So this expert does not get used again
                            expert_was_used = True
                            experts_used.append(expert_combination[expert_idx])
                            if all([val == 0 for val in project]):
                                projects_solved.append(project_combination[project_idx])
                            break
                project_idx += 1
            expert_idx += 1
        self.solutions.append(Solution(experts_used, projects_solved))
```

Technical Documentation

Expert Class:

This class has two class member variables:

- Features : This class member contains the feature array that this expert is skilled in.
- Used_for_feature : This class member contains the integer which indicates for which feature it has been used in the project assigned.

Project Class:

This class has two class member variables:

- Experts_needed : This is an array of integers which contains the number of features needed in each index to finish the project.
- Experts_used : This is an array of integers which contains the indices of experts already used to complete the project.

This class has three member functions:

- __lt__ : This is an overloaded less than operator for comparing two projects.
- __eq__: This is an overloaded equality operator for comparing two projects equality.

Solution Class:

This is class has four member variables:

- Project: This is an array of projects the solutions is intended to solve.
- Project_solved: This is an array of projects the solution succeeded to solve.
- Num_of_experts_used: This an integer value which represents the number of experts used.
- Num_of_projects_solved: This an integer value which represents the number of projects solved.

Complexity of the Algorithm

Complexity of the Brute Force algorithm:

- Num of project combinations = (N! + (N - 1)! + (N - 2)! + … + 1) where N is the amount of projects
- Num of Experts combinations = E!
- Num of features = F
- Current_projects = at most N
- Current_experts = E

Therefore the time complexity of the algorithm is:
O((N! + (N - 1)! + (N - 2)! + … + 1) * E! * N * E * F)

As the brute force algorithm finds all the possible solutions the pessimistic and average complexities are equal in such cases.

Proof of Correctness

It is bruteforce so we consider every possible combination and choose best combination, namely we choose combination which solves most projects.

Reports from Tests

1. Test1 : number of projects - 3, number of experts - 3 and number of features - 6

```
Expected amount of iterations: 810
Execution time: 0.03s
--SOLUTION RESULT--
Project 0
Experts used: [0]
Project 1
Experts used: [1]
Num of projects solved: 2
Projects solved: [0, 1]
Number of experts used: 2
```

2. Test2 : number of projects - 5, number of experts - 5 and number of features - 5

```
Expected amount of iterations: 975000
Execution time: 10.50s
--SOLUTION RESULT--
Project 0
Experts used: [0, 1, 2, 3, 4]
Num of projects solved: 1
Projects solved: [0]
Number of experts used: 5
```

3. Test3 : number of projects - 7, number of experts - 7 and number of features - 6

Expected amount of iterations: 3383105040

In this test we experience the failure of bruteforce method because of limited CPU capability.

Conclusion

After executing the brute force algorithm using all possible permutations of the projects and the experts.

The solutions resulted were up to mark and fast for small number (N <=10) of projects and features but as we kept increasing the numbers the algorithm got slower but still the solutions resulted were correct ones and it was easy to determine which is the best solution using a faster computing unit and really big computation time .

Bibliography

1. “Optimization: principles and algorithms” by Michel Bierlaire.

2. “Proofs and Algorithms: An Introduction to Logic and Computability” by Gilles Dowek.

3. “Introduction to Automata Theory, Languages, and Computation” by Jeffrey Ullman and John Hopcroft.

4. “Computability and Unsolvability” by Martin Davis.