A logo with a lion and building

Description automatically generated with medium confidence

POLITECNICO DI TORINO

DEPARTMENT OF CONTROL & COMPUTER ENGINEERING

*COMPUTATIONAL INTELLIGENCE*

**Class Activity\_01**

**Analysis & Comparison of All Set Covering Algorithms**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Matricula** | **GitHub** | **Date**  **submission** |
| Hossein Khodadadi | 313884 | [Link\_1](https://github.com/HOSSENkhodadadi/Computational_Intelligence/tree/main/Class_Activity_01) | 24/10/2023 |
| Abolfazl Javidian | 314441 | [Link\_2](https://github.com/Abolfazl-Javidian/Computational-Intelligence/tree/main/Class_Activity_01) | 24/10/2023 |

Fall 2023/2024

1. **Report: Set Covering Depth First Recursive Algorithm**

**1.1 Introduction**

The provided code is a Python implementation of an algorithm to solve the Set Cover problem. Set Cover is a well-known NP-hard optimization problem where you are given a universal set and a collection of subsets, and the goal is to find the minimum number of subsets required to cover the universal set.

**1.2 Code Flow**

Global Variable: node\_counter is a global variable used to count the number of nodes explored during the execution of the algorithm. It provides insight into the efficiency of the algorithm.

**1.3 find\_min\_set\_cover Function:**

This is the main function responsible for solving the set cover problem.

The function takes four parameters:

* universe: The set to be covered.
* subsets: A list of subsets.
* state: The current solution state (subset indices).
* subset\_indices: A list of indices for the subsets.

It uses a recursive approach to explore all possible combinations of subsets to cover the universe. The algorithm works by considering two possibilities: Including the first subset (chosen\_subset) in the solution and recursively solving the problem with the reduced universe and subsets.Not including the first subset and solving the problem with the original universe and the remaining subsets.The function returns the minimum cover found among these two possibilities.

**1.4 Constants and Data Generation:**

SETS is generated as a tuple of NUM\_SETS lists, where each list contains binary values (True/False) indicating whether an element is in the respective subset.subset\_indices is a list of indices corresponding to the subsets.

**1.5 Execution of the Algorithm:**

The universal set and subsets are generated from the constants and data.The find\_min\_set\_cover function is called with the universal set, subsets, an empty state, and subset indices to find the minimum set cover.

**1.6 Output and Reporting:**

If a valid set cover is found (result is not None), it prints the selected subsets and the number of subsets used for the minimum set cover.If no valid set cover is found, it prints "No valid set cover found." The code also prints the number of nodes explored (node\_counter), which represents the number of recursive calls made during the execution of the algorithm.

**1.7 Time Complexity Analysis**

The time complexity of this code is exponential, and it depends on the number of subsets and the size of the universal set. In the worst case, the algorithm explores all possible combinations of subsets, resulting in a time complexity of O(2^k), where 'k' is the number of subsets. The code is not optimized for large problem instances and can become very slow for a significant number of subsets.

Additionally, the global variable node\_counter is used to count the number of nodes explored. This variable provides insight into the algorithm's efficiency and can be used to analyze the actual number of recursive calls made during execution.

In summary, this code provides a basic implementation of the Set Cover problem-solving algorithm with a focus on clarity and correctness, but it is not optimized for large problem sizes due to its exponential time complexity.

1. **Comparison Table**

As an extra activity we evaluated the performance of other algorithms to perform a complete comparison.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Algorithm*** | ***# Iteration*** | ***Counter*** | ***Problem\_Size*** | ***# Subsets*** |
| Depth First |  |  |  |  |
| Breadth First |  |  |  |  |
| A\* |  |  |  |  |
| Recursive |  |  |  |  |