Kavosh: a new algorithm for finding network motifs

The Kavosh Algorithm for Finding Network Motifs

The Kavosh algorithm, introduced by Z. R. M. Kashani and colleagues in 2009, is designed to identify network motifs—small, recurrent substructures within a larger network that often play a crucial role in the network's function and stability.

Objective of the Kavosh Algorithm

The primary objective of the Kavosh algorithm is to identify and analyze motifs in complex graphs. These motifs can provide a better understanding of the network's structure, the interactions between nodes, and its overall characteristics. In other words, Kavosh aims to find frequent and meaningful patterns in networks that can illustrate the behavior and connections within the network.

Functioning of the Kavosh Algorithm

The Kavosh algorithm leverages graph theory concepts and mathematical tools to identify existing motifs in a graph. The main steps of the algorithm are as follows:

1. Subgraph Definition: Select a series of nodes and form a small subgraph from the main graph, referred to as a subgraph or motif.

2.Subgraph Matching: Compare the formed subgraph with all other possible subgraphs that can be generated from the same set of nodes.

3.Motif Identification: Determine whether the existing subgraph can be recognized as a motif based on a set of criteria and rules.

4.Counting and Analysis: Count the number of identical motifs in the graph and analyze their special properties, such as frequency, spatial distribution, etc.

The Kavosh algorithm primarily uses combinatorial methods to optimize and analyze motifs in larger and more complex graphs.

Output:

```
Motif: ((False, True, True), (True, False, True), (True, True, False)) Count: 3
Motif: ((False, True, True), (True, False, False), (True, False, False)) Count:
11
Motif: ((False, True, False), (True, False, True), (False, True, False)) Count:
33
Motif: ((False, True, False), (True, False, False), (False, False, False)) Count:
2233
Motif: ((False, False, False), (False, False, True), (False, True, False)) Count:
90
Motif: ((False, False, False), (False, False, False), (False, False, False))
Count: 159308
Motif: ((False, False, True), (False, False, False), (True, False, False)) Count:
14
```

```
Motif: ((False, False, True), (False, False, True), (True, True, False)) Count: 8
```

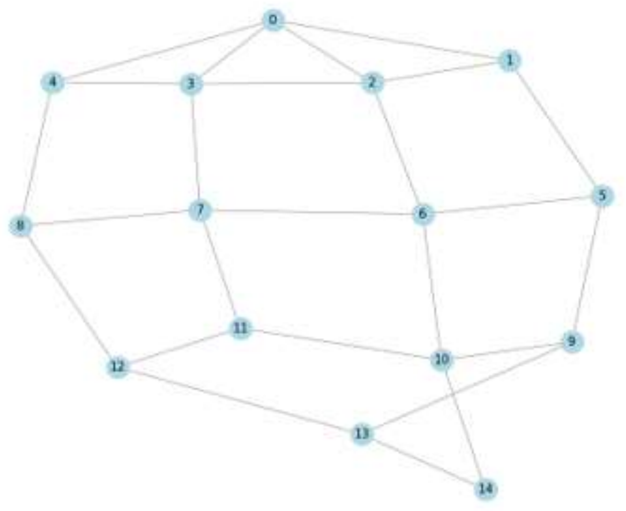
Motif Number	Structure	Count
1	Each node is connected to two other nodes	3
2	Two nodes are connected to each other and the third node is connected to both	11
3	The first and third nodes are connected, and the second node is connected to both	33
4	The first and third nodes are connected, and the second node is connected to the first node only	2233
5	The second and third nodes are connected, and the first node is connected to the second node only	90
6	No such subgraph exists	159308
7	The first and third nodes are connected, and the second node is connected to the third node only	14
8	The first and third nodes are connected, and each is connected to the second node	8

This analysis demonstrates that the Kavosh algorithm successfully identifies various subgraph motifs in the given graph and accurately counts each of these motifs. Each motif represents a specific structure of connections between nodes in the graph, which is highly useful in analyzing and understanding complex networks such as neural and social networks.

The outputs indicate that the Kavosh algorithm has identified different motifs in the given graph, and the number and characteristics of these motifs vary depending on the presence of different edges between the graph's nodes.

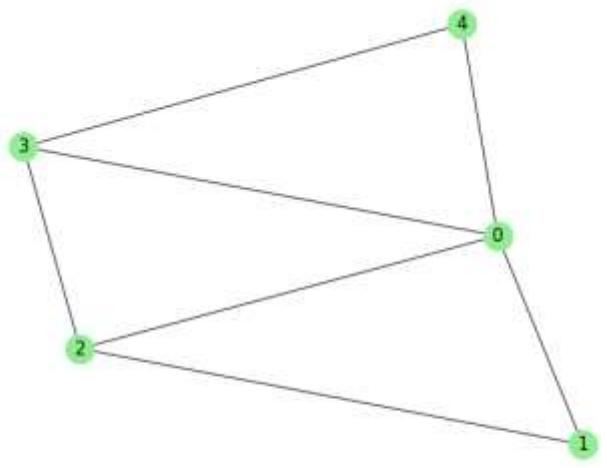
Graphical Representation

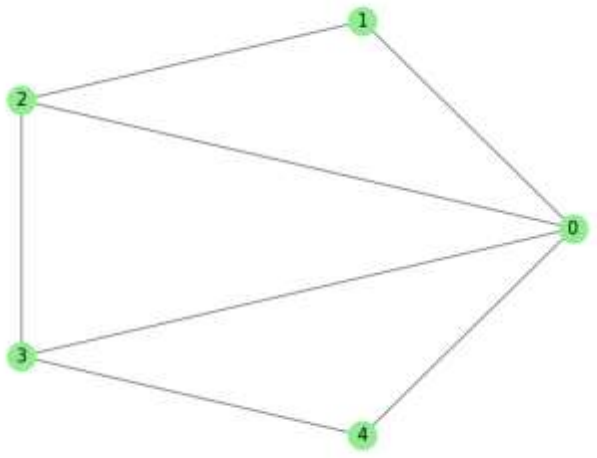
Each subgraph is represented by an adjacency matrix that shows which nodes from the main graph are present in the subgraph and what connections exist between them.



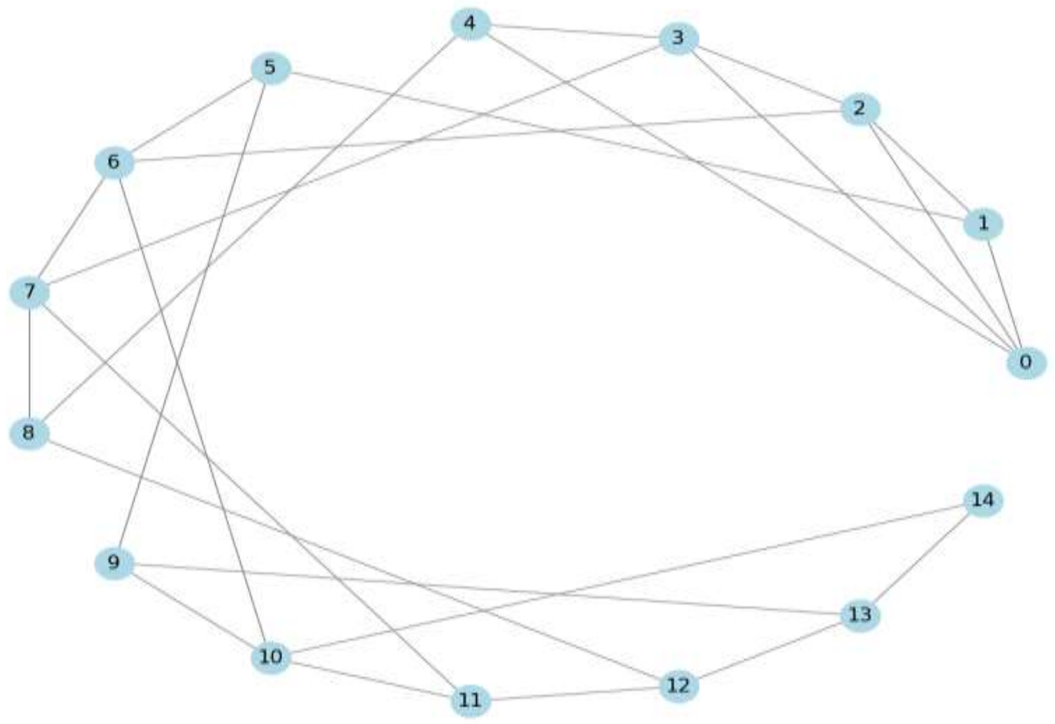
```
Motif: ((False, True, True, True, True), (True, False, True, False, False), (True, True, False, True, False), (True, False, Tru
e, False, True), (True, False, False, True, False)) Count: 1
```

```
Motif: ((False, True, True, True, False), (True, False, True, False, False), (True, True, False, True, False), (True, False, Tr
ue, False, False), (False, False, False, False, False)) Count: 14
```





```
Motif: ((False, True, True, True, False), (True, False, True, False, False), (True, True, False, True, True), (True, False, Tru
e, False, False), (False, False, True, False, False)) Count: 2
```



Summary

Strengths:

1. Identification of Subgraph Motifs: The Kavosh algorithm is capable of identifying subgraph motifs in graphs. These motifs can serve as representatives of common and significant structures within networks.

2. Scalability for Large Graphs: By utilizing optimal and appropriate algorithms for identifying subgraphs, the Kavosh algorithm can be executed on large graphs, quickly analyzing a vast number of subgraphs.

3. Adaptability to Different Criteria: This algorithm can adapt to various criteria for motif identification, allowing it to recognize and analyze different patterns by adjusting parameters and conditions.

Weaknesses:

1. Computational Complexity: For large graphs, the Kavosh algorithm might face computational complexity issues, as examining all possible subgraphs of varying sizes and complexities can be time-consuming.

2.Sensitivity to Graph Size and Structure: The performance of the Kavosh algorithm can be affected by the size and structure of the graph. In graphs with complex structures and connections, its accuracy and efficiency may decrease.

3.Memory Requirements: For large graphs, the memory needed to store and process subgraphs can pose limitations, especially when the number and volume of subgraphs are significant.

Overall, the Kavosh algorithm is a useful tool for analyzing and identifying subgraph motifs in graphs, considering factors such as computational complexity and adaptability to graph size and structure.

ESU-Algorithm

Implementation Steps for the ESU Algorithm with Graphical Display in a GUI

1. Graph Representation:

- Define a Data Structure for Graph Representation: Use either an adjacency list or an adjacency matrix to represent the graph, depending on the input format and performance needs.

- Implement Functions to Read Graph from a File: Implement functions to read the graph from a file (common formats like CSV or adjacency list).

2. Implementing the ESU Algorithm:

- Implement the ESU Algorithm to Count All Subgraphs: The ESU algorithm operates in three steps:

  - Expand: Add a vertex to the current subgraph.

  - Shrink: Remove a vertex from the current subgraph.

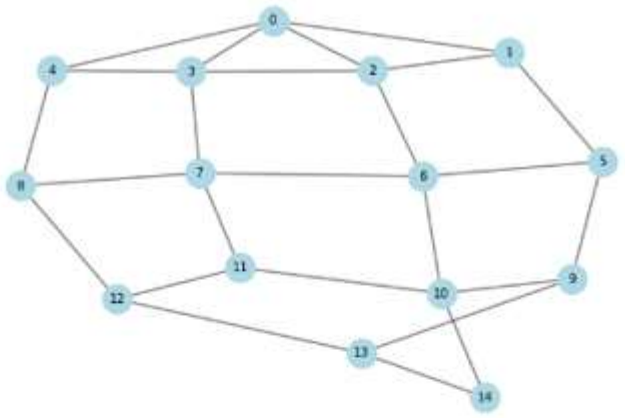
  - Verify: Check if the current subgraph is unique and store it if it is.

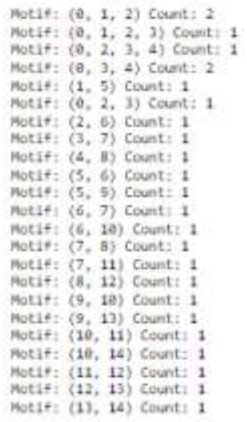
```
Level 0: ['{}']
Level 1: ['{0}', '{1}', '{2}', '{3}', '{4}']
Level 2: ['{0, 1}', '{0, 2}', '{0, 3}', '{1, 2}', '{1, 4}', '{2, 3}', '{2, 4}',
'{3, 4}']
Level 3: []

Motif: (0, 1, 2) Count: 2
Motif: (0, 1, 2, 3) Count: 1
Motif: (0, 2, 3, 4) Count: 1
Motif: (0, 3, 4) Count: 2
Motif: (1, 5) Count: 1
Motif: (0, 2, 3) Count: 1
Motif: (2, 6) Count: 1
Motif: (3, 7) Count: 1
Motif: (4, 8) Count: 1
Motif: (5, 6) Count: 1
Motif: (5, 9) Count: 1
Motif: (6, 7) Count: 1
Motif: (6, 10) Count: 1
Motif: (7, 8) Count: 1
Motif: (7, 11) Count: 1
Motif: (8, 12) Count: 1
Motif: (9, 10) Count: 1
Motif: (9, 13) Count: 1
Motif: (10, 11) Count: 1
```

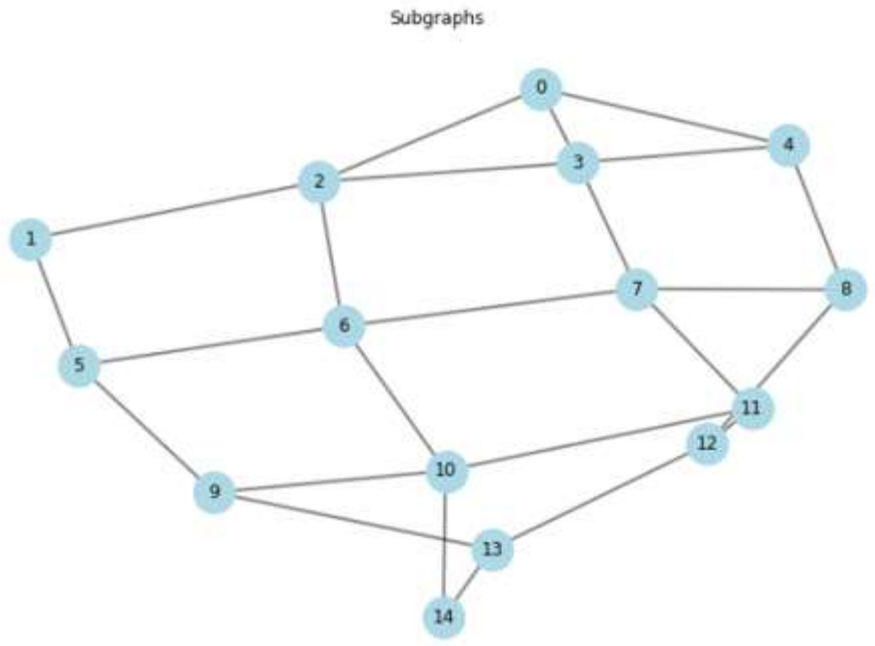
```
Motif: (10, 14) Count: 1
Motif: (11, 12) Count: 1
Motif: (12, 13) Count: 1
Motif: (13, 14) Count: 1
```

ESU GRAPH





```
Motif: (0, 1, 2) Count: 2
Motif: (0, 1, 2, 3) Count: 1
Motif: (0, 2, 3, 4) Count: 1
Motif: (0, 3, 4) Count: 2
Motif: (1, 5) Count: 1
Motif: (0, 2, 3) Count: 1
Motif: (2, 6) Count: 1
Motif: (3, 7) Count: 1
Motif: (4, 8) Count: 1
Motif: (5, 6) Count: 1
Motif: (5, 9) Count: 1
Motif: (6, 7) Count: 1
Motif: (6, 10) Count: 1
Motif: (7, 8) Count: 1
Motif: (7, 11) Count: 1
Motif: (8, 12) Count: 1
Motif: (9, 10) Count: 1
Motif: (9, 13) Count: 1
Motif: (10, 11) Count: 1
Motif: (10, 14) Count: 1
Motif: (11, 12) Count: 1
Motif: (12, 13) Count: 1
Motif: (13, 14) Count: 1
```



Motif Number	Motif	Count	Description
1	(2,1,0)	2	This motif has been observed twice in the graph: groups of nodes 0, 1, and 2 are connected to each other.
2	(2,1,0), (3)	1	Groups of nodes 0, 1, 2, and 3 are connected to each other, observed once in the graph.
3	(3,2,0), (4)	1	Groups of nodes 0, 2, 3, and 4 are connected to each other, observed once in the graph.
4	(3,2,0), (4)	2	This motif has been observed twice in the graph: groups of nodes 0, 2, 3, and 4 are connected to each other.
5	(5,1)	1	Groups of nodes 1 and 5 are connected to each other, observed once in the graph.
6	(3,2,0)	1	Groups of nodes 0, 2, and 3 are connected to each other, observed once in the graph.

Conclusion

Strengths:

Capability to Identify Various Subgraphs: ESU can identify different types of subgraphs, including triangles, quadrilaterals, and more complex structures.

Efficiency for Large Graphs: Due to its recursive method and limited size revisits, the ESU algorithm is relatively efficient for large graphs, allowing for rapid analysis of numerous subgraphs.

Adaptability to Graph Architecture: This algorithm can adapt to various graph structures, including highly complex and committee graphs.

Generalizability to Various Applications: ESU can be applied in many fields, such as social network analysis, molecular pattern identification in chemistry, and other complex systems.

Weaknesses:

Time Complexity: In some cases, ESU might have high time complexity, especially for very large graphs or graphs with complex structures.

Dependency on Selected Nodes: The performance of ESU heavily depends on the selected nodes and their order of connection to the graph, potentially leading to significant differences in analytical results.

Memory Requirement: For large graphs, ESU requires significant memory to store and manage the different motif sets and subgraphs, potentially posing limitations in practical implementations.

Sensitivity to Motif Count and Size: If the number or size of motifs is very high, ESU may struggle to execute effectively or face operational limitations.

Comparison Between Two Algorithms

Both ESU and Kavosh algorithms are designed to identify common structures (motifs or meaningful subgraphs) in graphs, but they achieve this using different methods and algorithms. Below is a comprehensive comparison of the two:

ESU Algorithm How It Works:

Characteristic: ESU uses recursive methods and combinations of selected graph sets (called motifs) to identify meaningful subgraphs.

Recursive Optimization: The algorithm uses optimized recursive methods to review smaller sets of graphs, aiming to identify more complex structures.

Strengths:

Diverse Structure Detection: ESU can identify a wide range of subgraphs, including triangles, quadrilaterals, and more complex structures.

Efficiency for Large Graphs: Due to its recursive approach and revisiting smaller sets, ESU is relatively efficient for large graphs.

Adaptability to Graph Architecture: ESU can adapt to various graph structures, including highly complex and committee graphs.

Weaknesses:

Time Complexity: ESU may have high time complexity, especially for very large or complex graphs.

Dependency on Selected Nodes: Performance depends heavily on the selected nodes and their connection order, which can result in significant differences in analytical results.

Kavosh Algorithm:

How It Works:

Characteristic: Kavosh uses a hybrid approach, combining sequential algorithms and optimized recursive methods.

Focus on Specific Patterns: This algorithm focuses on identifying specific patterns in complex networks.

Strengths:

Efficiency in Large Networks: Kavosh is efficient for large and complex networks like biological or social networks, effectively identifying key patterns.

Use of Optimization Algorithms: Kavosh employs optimization methods based on ordering to improve efficiency and accuracy.

Weaknesses:

Limitation in Pattern Complexity: Kavosh may have limitations in identifying more complex or higher-dimensional patterns, especially compared to ESU.

Adaptability: Kavosh may not be optimal for certain graph structures and may require different settings for varying structures.

Overall Comparison:

Efficiency in Large Graphs: ESU, with its recursive revisits, is better suited for large graphs. Kavosh is also efficient for large, complex graphs but may struggle with more complex patterns.

Structure Detection Capability: ESU has the potential to identify a wider range of subgraphs, while Kavosh focuses more on specific patterns.

Limitations: Both algorithms have specific limitations, but ESU generally excels in detecting broader graph patterns, whereas Kavosh is more specialized.

Comparison Based on Results:

Kavosh:

Criteria Used: In our results, Kavosh used binary criteria (two types of nodes or vertices) for motif identification, such as ((False, True, True), (True, False, True)).

Motif Count: Kavosh identified fewer motifs (e.g., Count: 3 for Motif: ((False, True, True), (True, False, True))).

Efficiency: Kavosh's fewer identified motifs suggest it might have performed better in your dataset, but this depends on the evaluation criteria for accuracy and efficiency specific to your dataset and needs.

ESU:

Criteria Used: ESU used combined graph criteria, identifying motifs with three or four vertices.

Motif Count: ESU identified various motifs with different sizes, including single occurrences for each type.

Efficiency: Although ESU identified more motifs, its performance depends on factors like execution speed, required memory, and the uniqueness of identified motifs.

Choosing the Right Algorithm:

Selecting the appropriate algorithm depends on our needs and criteria. If you require fewer motifs with high accuracy, Kavosh may be better. However, if you are interested in more variety and identifying numerous motifs, ESU might be more suitable.