A Tool to improve Drug Discovery

Ponugoti Shravya
IIIT Hyderabad
Hyderabad, India
shravya.ponugoti@students.iiit.ac.in

Nida Shahnoor
IIIT Hyderabad
Hyderabad, India
nida.shahnoor@students.iiit.ac.in

Abstract

Addressing complex diseases such as cancer and HIV necessitates the discovery of effective drug combinations through the analysis of chemical compounds and their structural fragments, often modeled as graph structures. Earlier efforts in this domain have leveraged algorithms like Graph Transactional Coverage Patterns (GTCPs) and pattern mining techniques from Graph Transactional Data (GTD) to identify combinations of compounds that collectively span significant portions of fragment datasets. Building on this, our work introduces an efficient framework for extracting Subgraph Coverage Patterns (SCPs), a generalization of GTCPs. This framework has been evaluated on three real-world chemical datasets, demonstrating both high performance and practical relevance. A case study in computer-aided drug design further validates the utility of the approach.

To enhance accessibility and user experience, we developed a web-based platform that integrates this SCP-based mining framework. This platform provides an intuitive interface for researchers to perform real-time SCP analysis, visualize key patterns, and collaborate effectively. By translating complex graph mining capabilities into a user-friendly web solution, this system aims to accelerate the discovery of promising drug compound combinations and broaden the reach of computational drug design tools.

I. Introduction

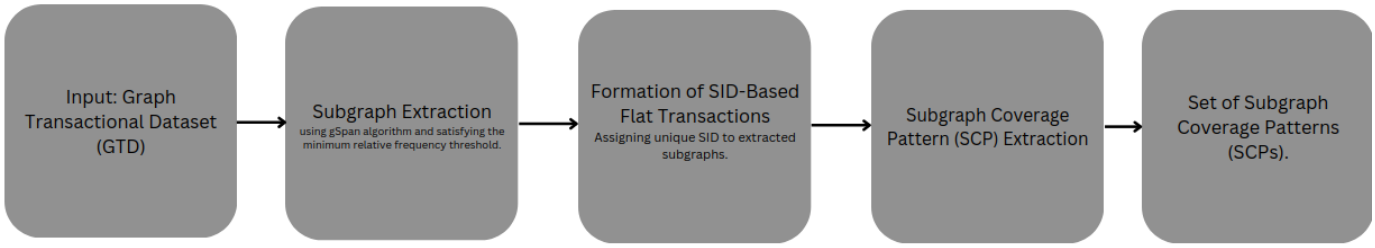
Modern medicine has increasingly relied on the use of drug combinations to treat complex diseases such as cancer, HIV, and autoimmune disorders. This approach often proves more effective than single-drug therapies because combinations can target multiple pathways of a disease simultaneously, reducing the likelihood of resistance and improving patient outcomes. Furthermore, carefully chosen drug combinations can minimize adverse side effects, providing a higher quality of life for patients undergoing treatment. However, the discovery and optimization of these combinations remain a significant challenge for researchers.

Identifying effective drug combinations is an expensive, time-intensive process. Traditional laboratory methods require exhaustive experimental testing to evaluate the efficacy of combinations, which becomes increasingly impractical as the number of potential combinations grows exponentially. For example, testing three-drug combinations from a pool of 100 drugs results in over 161,000 unique combinations. This combinatorial explosion makes it infeasible to test all possibilities in a laboratory setting. Consequently, researchers face the dilemma of narrowing down which combinations to investigate without compromising the discovery of highly effective treatments.

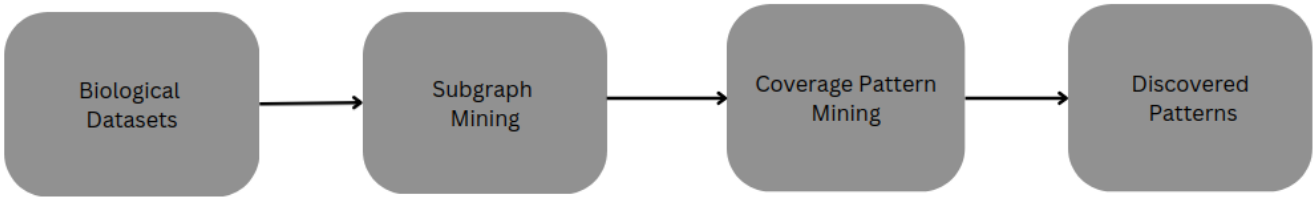
While computational methods have been introduced to aid in drug discovery, many existing tools lack the ability to fully leverage the structural information of drugs. Most focus on properties such as chemical similarity or prior knowledge of known drug interactions, which can limit their applicability to novel drug discovery. Additionally, current approaches often fail to ensure diversity in selected combinations, leading to redundancy and diminishing the likelihood of identifying innovative therapies. Moreover, there is limited integration of single graph-based coverage-related knowledge to enhance the utility of computational tools for drug combination discovery.

To tackle this issue, various data mining techniques have been proposed to analyze biological targets and clinical trials, helping researchers discover new drug candidates and understand how drugs work.

One promising method is Subgraph Coverage Patterns (SCPs). This approach models chemical compounds as graphs, where atoms are represented as vertices and chemical bonds as edges, forming a Graph Transactional Dataset (GTD). Sub-compounds or molecular fragments are treated as subgraphs. Using a subgraph mining algorithm, these fragments are extracted from the GTD. Rather than evaluating all possible fragment combinations, SCPs efficiently identify a minimal set of subgraphs that collectively achieve user-specified coverage of the dataset. This significantly streamlines the drug discovery process by focusing on representative patterns with maximal relevance. The following flowchart illustrates how this SCP-based workflow operates:



Another one is Graph Transactional Coverage Patterns (GTCPs). This approach models chemical compounds as graphs, where atoms are vertices and bonds are edges, forming a Graph Transactional Dataset (GTD). Sub-compounds, or fragments, are represented as subgraphs. Using an existing subgraph mining algorithm, we extract these fragments from the GTD. Instead of evaluating every possible combination, GTCPs efficiently identify patterns that meet user-specified fragment coverage, streamlining the drug discovery process. The following is the flowchart how this works -



A practical tool is under development with the aim of benefiting researchers and chemists by simplifying and enhancing drug discovery processes. By making the tool more user-friendly and accessible, it has the potential to significantly impact the field. The primary objectives of this project are as follows:

- To explore how Structural Coverage Pattern (SCP)-based approaches can advance drug discovery, particularly in identifying diverse and effective drug combinations.
- To establish meaningful input and output variables that will guide data preparation, analysis, and interpretation effectively.
- To improve the current tool capable of extracting SCPs from biological datasets, thereby offering new insights into drug interactions and mechanisms.
- To deliver a robust, intuitive tool that enables researchers to efficiently identify promising drug combinations, bridging the gap between computational predictions and laboratory validation.

By introducing a system that analyzes drug structures at a fragment level, integrates diverse knowledge from single graph representations, and is accessible via a user-friendly web platform, this research aims to revolutionize drug combination discovery. The ultimate goal is to reduce the time and cost of the drug development pipeline while enhancing the effectiveness of treatments for complex diseases.scalable and accessible solutions for modern drug discovery challenges.

II. Related work

This section discusses previous research in areas related to drug discovery, focusing on methods for identifying drug combinations, using smaller fragments of drugs for study, and utilizing coverage concepts to improve efficiency. The following summarizes key findings from prior work and highlights how our approach differs and builds on these methods.

A. A Model of Subgraph Coverage Patterns with Applications to Drug Discovery

The introduction of Subgraph Coverage Patterns (SCPs) enables the identification of drug combinations that meet specified fragment set coverage requirements. By employing subgraph mining techniques, SCPs extract drug combinations that satisfy fragment set coverage and overlap ratio constraints, effectively addressing the combinatorial explosion in drug combination discovery. This framework ensures diversity in the combinations and reduces computational complexity.

1) Earlier Drug Combination Methods: Earlier research explored ways to identify effective drug combinations by using techniques borrowed from digital communications. These methods required testing each combination to evaluate its biological effects. One key finding from these studies was that using a wide variety of chemical compounds significantly increases the likelihood of discovering successful drug combinations.

2) Fragment-Based Drug Research: Researchers have also studied how breaking drugs down into smaller pieces, or "fragments," can improve our understanding of their properties. Some approaches have used these fragments to:

- Assess the toxicity or safety of drugs.
- Investigate how drugs interact with proteins in the body.
- Apply machine learning techniques to efficiently screen potential drug candidates.

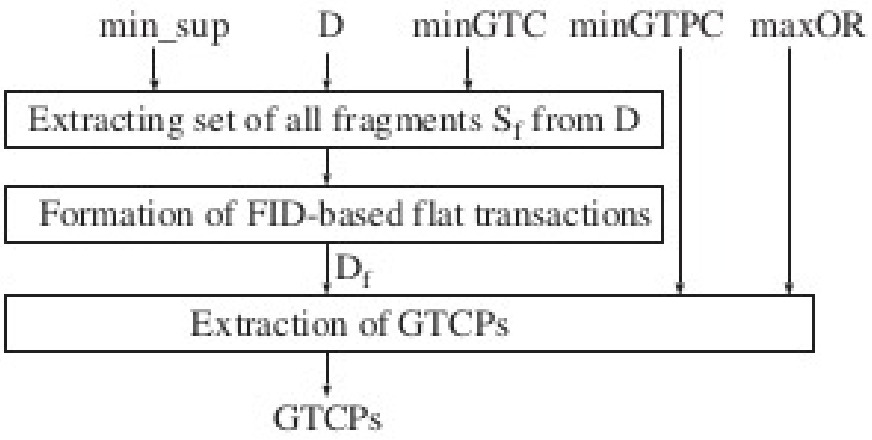
This fragment-based research has helped refine drug discovery by focusing on smaller, more manageable pieces of information, which can then be recombined to form new drug candidates.

3) Coverage Concepts: The concept of "coverage" is used across multiple fields, from optimizing the placement of services or facilities to planning efficient routes and even targeting online advertisements. Previous studies have developed methods to identify patterns in data that provide effective coverage. However, the application of these coverage concepts to drug discovery has not been explored extensively. Our work introduces a novel way of applying coverage patterns to the process of identifying useful drug combinations, combining ideas from various fields to optimize this task.

4) The Overlap Problem in Drug Combinations: A significant challenge in drug discovery is the overwhelming number of possible drug combinations, making it impractical to test each one individually. To address this, researchers have focused on understanding the overlap between different combinations, allowing them to eliminate unnecessary tests. By considering the overlap, we can prioritize combinations that offer diversity and increase the likelihood of finding effective treatments.

5) Summary of Key Concepts: To better understand the terminology used in this paper, the following key terms are defined:

- **GTD (Graph Transactions Dataset):** A method for representing drug data in a graph structure.
- **Fragments:** The smaller components of drugs that are studied individually.
- **Patterns:** Common substructures or configurations in which drug fragments co-occur.
- **SCP (Subgraph Coverage Pattern):** A pattern mining approach used to identify a minimal set of subgraphs that collectively cover a large portion of the dataset, helping in the discovery of relevant drug combinations.

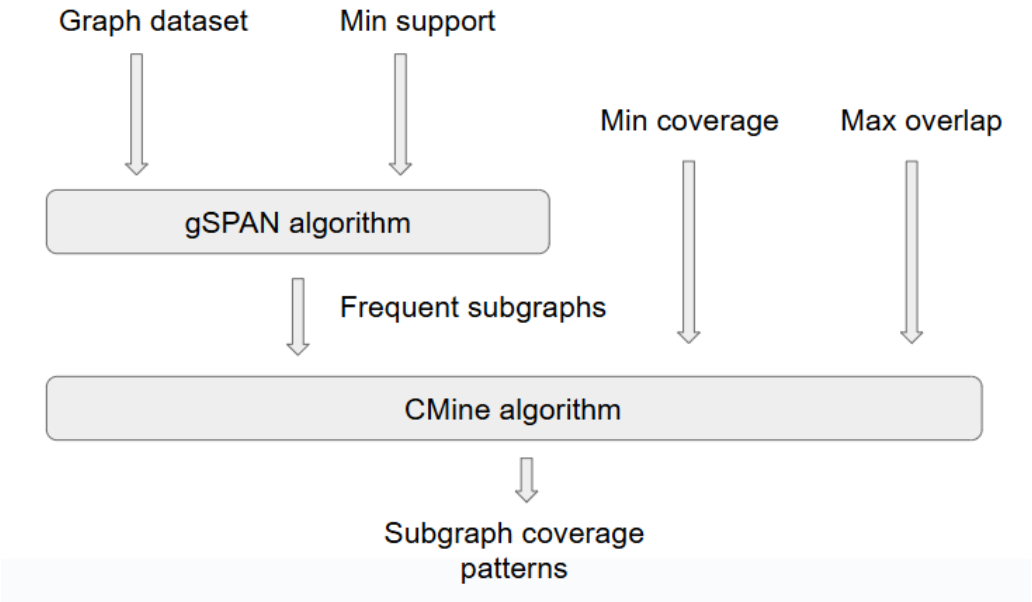


B. Mining subgraph coverage patterns from graph transactions

This paper introduces Subgraph Coverage Patterns (SCPs) for Graph Transactional Data (GTD), an area within graph mining with applications in bioinformatics, chemical informatics, and social networks. While frequent subgraph mining is commonly used in GTD, the aspect of coverage has not been sufficiently explored. A novel approach is proposed to extract SCPs, with constraints on frequency, coverage, and overlap. The key innovation is the Subgraph ID-based Flat Transactional (SIFT) framework, which enables efficient extraction of SCPs by transforming graph transactions into flat transactions. This method is demonstrated through a case study in drug design.

Graph Transactional Data (GTD) refers to datasets consisting of multiple graph transactions, each containing a set of subgraphs that can be part of the dataset. Frequent Subgraph Mining (FSM) is a technique used to find interesting patterns within these datasets. However, previous research has not sufficiently addressed the coverage aspect of subgraphs in GTD.

This work proposes a new framework for extracting Subgraph Coverage Patterns (SCPs), focusing on the coverage support, frequency, and overlap of subgraphs. The key contribution is the Subgraph ID-based Flat Transactional (SIFT) framework, which simplifies the extraction process by transforming graph transactions into flat transactions with unique Subgraph Identifiers (SIDs).



1) Key Concepts:

- **Graph Transactional Dataset (GTD)**: A dataset consisting of graph transactions $G_1, G_2, \ldots, G_n$.
- **Subgraph Pattern (SP)**: A set of subgraphs from a universe of subgraphs $\mathcal{U}$, which may be part of the GTD. A subgraph S_j covers a graph transaction G_i if $S_j \subseteq G_i$.
- **Minimum Support**: The minimum percentage of graph transactions in the dataset D that must be covered by a subgraph S_j for it to be considered frequent.
- **Coverage**: The percentage of transactions in D covered by at least one subgraph in a subgraph pattern S_P. A pattern is considered interesting if its coverage support satisfies a user-defined threshold.
- **Overlap**: The amount of duplication in the transactions covered by subgraphs within a pattern.
- **Subgraph Coverage Pattern (SCP)**: A subgraph pattern S_P is considered an SCP if it meets the conditions set by the relative frequency, coverage support and overlap parameters.

2) Subgraph ID-based Flat Transactional (SIFT) Framework: The key innovation in this work is the Subgraph ID-based Flat Transactional (SIFT) framework. The SIFT framework transforms graph transactions into flat transactions with unique Subgraph Identifiers (SIDs). This transformation makes it possible to apply set-based operations rather than computationally expensive graph-based calculations.

The process involves representing each subgraph as a unique identifier and transforming graph transactions into flat transactions that contain these identifiers. This allows for efficient extraction of SCPs by leveraging set-based operations, significantly reducing the complexity of the problem.

III. Current Status

A. Current status of research

The current research on Subgraph Coverage Patterns (SCPs) presents an innovative method for drug discovery by identifying coverage-based subgraph patterns from graph transactional datasets. In this approach, chemical compounds are represented as graphs, and sub-compounds (fragments) are modeled as subgraphs within a graph transaction. A dedicated framework has been developed to efficiently extract SCPs, incorporating user-defined constraints such as minimum coverage and maximum overlap. This ensures that the resulting patterns are diverse, non-redundant, and practically meaningful for drug design applications.

Experimental evaluations on real-world datasets demonstrate the scalability and pruning efficiency of the SCP extraction process. Furthermore, a case study in computer-aided drug design highlights the practical relevance of SCPs in identifying candidate drug combinations. This research marks a significant advancement by integrating subgraph coverage mining with chemical compound analysis, offering a robust tool for graph-based drug discovery. With the growing interest in graph representations for biomedical data, SCPs provide a strong foundation for future research in areas like cheminformatics, bioinformatics, and large-scale molecular screening. Further enhancements may explore optimized algorithms and broader dataset integration to expand its applicability.

B. Existing tool

The tool operates on a dataset known as Graph Transaction Datasets (GTDs), where chemical compounds are modeled as graph transactions. Using the gSpan algorithm, it performs subgraph mining to discover frequently occurring substructures or fragments within these graph transactions. These fragments, represented as subgraphs, are identified based on a user-defined minimum support threshold that governs how frequently a fragment must appear in the dataset.

After extracting these frequent subgraphs, the tool converts the GTD into a set of flat transactions. Instead of preserving full graph structures, each transaction is now expressed as a set of fragment identifiers (fid) indicating the presence of specific frequent subgraphs. This transformed representation serves as the input for the Subgraph Coverage Pattern (SCP) extraction process.

Users can configure parameters such as minimum coverage and maximum overlap ratio to guide the SCP extraction. The resulting SCPs represent diverse and informative combinations of fragments that collectively cover large portions of the dataset while minimizing redundancy. These patterns are then presented to the user along with their associated coverage statistics, enabling insightful analysis for applications such as drug discovery.

IV. Gaps Identified in SCP Framework

While the concept of Subgraph Coverage Patterns (SCPs) and the proposed framework for their extraction represent notable contributions to drug discovery, there are several gaps and areas that require further exploration:

1) **Lack of Practical Implementations:** Despite the well-defined theory behind SCPs, there is a lack of readily available tools or frameworks for extracting SCPs from graph transactional datasets, limiting practical usage in real-world scenarios.
2) **Scalability Challenges:** While the framework has been tested on a limited number of datasets, its ability to scale to larger, more complex datasets that are typical in bioinformatics and drug discovery research has yet to be thoroughly explored.
3) **Integration with Existing Tools:** The framework does not yet seamlessly integrate with popular graph mining or data analytics platforms, which could prevent wider adoption by researchers who rely on other established tools.

4) **Data Quality and Preprocessing Issues:** Real-world graph transactional datasets often contain noise, incomplete data, or inconsistencies in representations, which may complicate the practical implementation of the SCP extraction framework and affect its accuracy.

V. Completion of Project Objectives

1) **Literature Review**:
A comprehensive survey of prior research was conducted to understand key concepts in subgraph mining, Graph Transactional Datasets (GTDs), and coverage-based pattern discovery algorithms such as GTCP and SCP. The review helped in identifying gaps in existing tools and guided the selection of algorithms and datasets.
2) **Input and Output Specifications**:
The input was defined as a SMILES-based chemical dataset, which is converted to a graph transactional format. Output specifications included frequent subgraph fragments, subgraph coverage patterns (SCPs), and their associated coverage statistics, all visualized through an interactive interface.
3) **Figma Layout Design**:
A detailed front-end interface prototype was created using Figma. The layout was designed to ensure intuitive user interaction, clarity in visualizing mined patterns, and easy configuration of parameters such as minimum support and coverage thresholds.

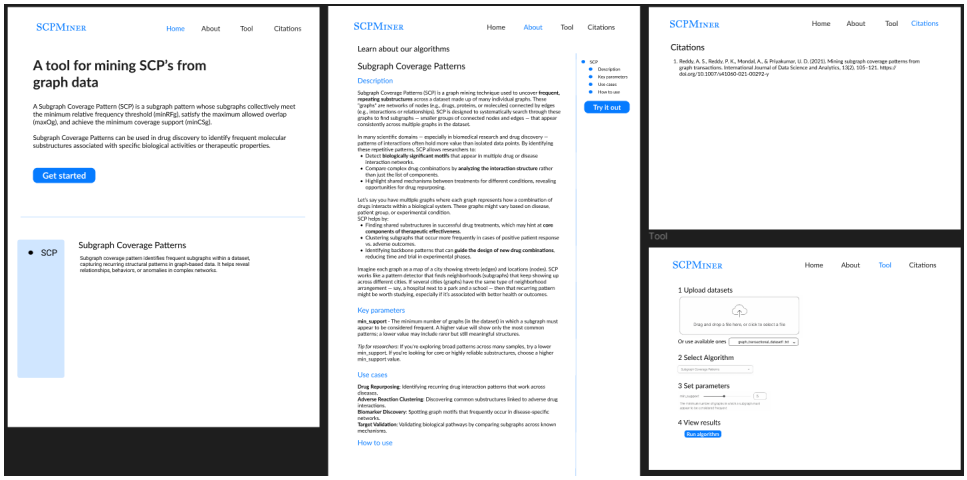


Fig. 1. Figma layout

4) **Frontend Integration**:
The designed Figma components were translated into a functional web interface using modern web technologies such as React.js, which was chosen for its component-based architecture and efficient state management. Features include file uploads, parameter selection, and real-time result display, ensuring a smooth and dynamic user experience. React's virtual DOM and reusable components allowed for fast rendering and easy integration with backend APIs, making it ideal for building a responsive and maintainable interface.
5) **Backend Scalability**:
The backend was built to efficiently handle graph conversion, frequent subgraph mining using gSpan, and SCP extraction using the C-Mine algorithm. Emphasis was placed on modular design, asynchronous processing, and scalability to support large datasets. FastAPI was chosen for its high performance and support for asynchronous operations, ideal for handling graph mining tasks efficiently. Its automatic validation and API documentation accelerated development and ensured robust, user-friendly endpoints. Seamless integration with Python algorithms simplified backend logic and maintained consistency across the stack. Lightweight and scalable, FastAPI provided a reliable framework for both rapid prototyping and production deployment.
6) **Testing and Debugging**:
Testing was conducted to validate the correctness of the subgraph mining process and SCP extraction, using a few representative SMILES datasets. Both unit and integration tests were performed to assess functionality across the pipeline. Errors related to parameter input and file uploads were identified and gracefully handled. Bugs encountered during data transformation, mining, and rendering were systematically resolved to ensure robustness and stability of the tool.

VI. REFERENCES

1) A. Srinivas Reddy, P. Krishna Reddy, Anirban Mondal, and U. Deva Priyakumar, "Mining subgraph coverage patterns from graph transactions," *International Journal of Data Science and Analytics*.
2) A. Srinivas Reddy, P. Krishna Reddy, Anirban Mondal, and U. Deva Priyakumar, "A Model of Graph Transactional Coverage Patterns with Applications to Drug Discovery," *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC)*.
3) M. Snowden and D. V. Green, "The impact of diversity-based, high-throughput screening on drug discovery: chance favours the prepared mind," *Current Opinion in Drug Discovery & Development*, vol. 11, no. 4, pp. 553–558, 2008.
4) Aida, M., Pieter, M., Wout, B., Pieter, M., Boris, C., and Bart Goethals, K. L., "Grasping frequent subgraph mining for bioinformatics applications," *BioData Mining*, vol. 11, no. 1, pp. 1–20, 2018.

VII. APPENDIX

A. gSpan algorithm

```
class GSpan:
    initialize(iFile, minSupport, outputSingleVertices, maxNumberOfEdges,
    outputGraphIds):
        set parameters: minSup, maxNumberOfEdges, outputSingleVertices,
        outputGraphIds, inPath = iFile
        initialize counters and lists: frequentSubgraphs, patternCount,
        graphCount, etc.

    method mine():
        if maxNumberOfEdges <= 0: return
        set frequentSubgraphs = empty list, patternCount = 0
        start time t1
        graphDb = readGraphs(inPath)
        set minSup = ceil(minSup * len(graphDb))
        call gSpan(graphDb, outputSingleVertices)
        end time t2
        calculate runtime and memory usage
        set patternCount = len(frequentSubgraphs)

    method save(oFile):
        open oFile for writing
        for subgraph in frequentSubgraphs:
            build subgraph description (vertices, edges, support)
            if outputGraphIds: append graph IDs
            write description to oFile

    method readGraphs(path):
        initialize graphDatabase and vMap
        for each line in file:
            if line starts with "t": append graph to database if vMap is not empty
            if line starts with "v": extract vertexId and label, create Vertex
            if line starts with "e": extract edge data, create Edge and update
            vertices append last graph if vMap is not empty
        return graphDatabase

    method subgraphIsomorphisms(c, g):
        initialize isomorphisms with startLabel from DFS code
        for each edge in DFS code c:
            for iso in isomorphisms:
                extend and validate iso with edge
        return isomorphisms
```

```
method rightMostPathExtensionsFromSingle(c, g):
    if DFS code is empty: return all edges in g as extensions
    find isomorphisms and extend with rightmost path
    return extensions

method removeInfrequentVertexPairs(graphDb):
    initialize support matrices if required
    for each graph g:
        update support for vertex pairs and edge labels
    remove infrequent entries based on minSup
    for each graph g: remove infrequent edges based on support

method getMemoryRSS(): return memoryRSS
method getMemoryUSS(): return memoryUSS
method getRuntime(): return runtime

method getFrequentSubgraphs():
    initialize subgraph descriptions
    for each subgraph: build description (vertices, edges, support)
    return joined descriptions as a string

method getSubgraphGraphMapping():
    initialize mappings
    for each subgraph: create mapping with FID, DFS code, and graph IDs
    return mappings

method saveSubgraphsByGraphId(oFile):
    group subgraphs by graph ID
    sort and write each graph ID with subgraph indices to oFile
```

B. CMine algorithm

```
class CMine:
def __init__(self, database, min_support, min_coverage, max_overlap):
    """
    Parameters:
        database: list of sets; each set is a transaction
        min_support: minimum number of transactions an itemset must appear in
        min_coverage: minimum fraction of database covered by itemset (GTPC)
        max_overlap: maximum allowed overlap with previously accepted patterns
    """
    self.database = database
    self.min_support = min_support
    self.min_coverage = min_coverage
    self.max_overlap = max_overlap
    self.num_transactions = len(database)
    self.L = [] # final patterns
    self.prev_patterns = [] # to check overlap

def get_frequent_1_itemsets(self):
    item_counts = {}
    for transaction in self.database:
        for item in transaction:
            item_counts[item] = item_counts.get(item, 0) + 1
    frequent_items = []
    for item, count in item_counts.items():
```

```
            if count >= self.min_support:
                frequent_items.append((frozenset([item]), count))
        return frequent_items

    def support(self, itemset):
        count = 0
        for transaction in self.database:
            if itemset.issubset(transaction):
                count += 1
        return count

    def coverage(self, itemset):
        covered = set()
        for idx, transaction in enumerate(self.database):
            if itemset.issubset(transaction):
                covered.add(idx)
        return len(covered) / self.num_transactions

    def overlap(self, itemset):
        overlap_count = 0
        for prev, _ in self.prev_patterns:
            if itemset & prev:
                overlap_count += 1
        return overlap_count / len(self.database)

    def join_step(self, prev_level_itemsets):
        candidates = []
        n = len(prev_level_itemsets)
        seen = set()
        for i in range(n):
            for j in range(i + 1, n):
                a, _ = prev_level_itemsets[i]
                b, _ = prev_level_itemsets[j]
                union = a | b
                if len(union) == len(a) + 1 and union not in seen:
                    seen.add(union)
                    count = self.support(union)
                    if count >= self.min_support:
                        candidates.append((union, count))
        return candidates

    def run(self):
        start_time = time.time()
        level = 1
        current_itemsets = self.get_frequent_1_itemsets()

        while current_itemsets:
            next_level_itemsets = []
            for itemset, count in current_itemsets:
                cov = self.coverage(itemset)
                ovl = self.overlap(itemset)
                if cov >= self.min_coverage and ovl <= self.max_overlap:
                    self.L.append((itemset, cov))
                    self.prev_patterns.append((itemset, cov))
                else:
                    next_level_itemsets.append((itemset, count))
```

```
    current_itemsets = self.join_step(next_level_itemsets)
    level += 1
end_time = time.time()
process = psutil.Process(os.getpid())
self.execution_time = end_time - start_time
self.memory_uss = process.memory_full_info().uss
self.memory_rss = process.memory_info().rss
# Sort patterns by coverage and then by length descending
self.L = sorted(self.L, key=lambda x: (x[1], len(x[0])), reverse=True)

return self.L
```