Managing and Mining Graph Data

中国科学技术大学
University of Science and Technology of China

QiuWentao

Content

Section 7

- Exact Graph Matching.
- Inexact Graph Matching
- Graph Matching for Data Mining and Information Retrieval
  - Graph Edit Distance
  - Other Inexact Graph Matching Techniques
- Vector Space Embeddings of Graphs via Graph Matching

Yellow Highlights Are What I Consider To Be Relatively Important Content, While Red Highlights Are What I Still Need To Further Understand.

Section 10

- Types of Dense Components
  - Absolute Vs. Relative Density
  - Graph Terminology
  - Definitions of Dense Components
  - Dense Component Selection
  - Relationship between Clusters and Dense Components
- Algorithms for Detecting Dense Components in a Single Graph
  - Exact Enumeration Approach
  - Heuristic Approach
  - Exact and Approximation Algorithms for Discovering Densest Components
- Frequent Dense Components
  - Frequent Patterns with Density Constraints
  - Dense Components with Frequency Constraint
  - Enumerating Cross-Graph Quasi-Cliques Applications of Dense Component Analysis

Section 12

- Frequent Subgraph Mining
  - Problem Definition
  - Apriori-based Approach
  - Pattern-Growth Approach
  - Closed and Maximal Subgraphs
  - Mining Subgraphs in a SingleGraph
- Mining Sigifcant Graph Ptterns
  - Problem Definition
  - gboost:ABranch-and-Bound Approach
  - gPLS: APartial Least Squares Regression Approach
  - LEAP:AStructural Leap Serch Approach
  - GraphSig: AFeature Representation Approach
- Mining Rpresentative Orthogonal Graphs
  - Problem Definition
  - Randomized Maximal Subgraph Mining
  - Orthogonal Representative Set Generation

SECTION7
EXACT AND INEXACT GRAPHMATCHING METHODOLOGY AND APPLICATIONS

Exact Graph Matching

The aim in exact graph matching is to determine whether two graphs, or at least part of them, are identical in terms of structure and labels.

Graph dissimilarity measures can be derived from the maximum common subgraph of two graphs. mcs is Maximum common subgraph. MCS is Minimum common subgraph.

Distance measure	Value
$d_{MCS}(g_1, g_2) = 1 - \frac{\lvert mcs(g_1, g_2)\rvert}{\max\{\lvert g_1\rvert, \lvert g_2\rvert\}}$	Isomorphic: 0 No Part In Common: 1
$d_{WGU}(g_1, g_2) = 1 - \frac{\lvert mcs(g_1, g_2)\rvert}{\lvert g_1\rvert + \lvert g_2\rvert - \lvert mcs(g_1, g_2)\rvert}$	Behaves similarly to d_{MCS}, Allow for changes in the smaller graph to exert some influence on the distance measure
$d_{UGU}(g_1, g_2) = \lvert g_1\rvert + \lvert g_2\rvert - 2 \cdot \lvert mcs(g_1, g_2)\rvert$	Similar to d_{WGU}. But not normalized to the interval [0, 1]
$d_{MMCS}(g_1, g_2) = \lvert MCS(g_1, g_2)\rvert - \lvert mcs(g_1, g_2)\rvert$	Based on both the maximum and minimum common supergraph
$d_{MMCSN}(g_1, g_2) = 1 - \frac{\lvert mcs(g_1, g_2)\rvert}{\lvert MCS(g_1, g_2)\rvert}$	Normalized d_{MMCS}

Inexact Graph Matching — Graph Edit Distance

Definition:

1. A sequence of edit operations e1, . . . , ek that transform g1 into g2 is called an edit path between g1 and g2.
2. A standard set of edit operations is given by insertions, deletions, and substitutions of both nodes and edges.
3. The edit distance of two graphs is defined by the minimum cost edit path between two graphs.

Drawbacks:

1. Prior knowledge of the graphs' labels is often inevitable for graph edit distance to be a suitable proximity measure.
2. Computational complexity, which is exponential in the number of nodes of the involved graphs.

Limitation:

1. Application of optimal algorithms for edit distance computations is limited to graphs of rather small size in practice.

Other Inexact Graph Matching Techniques

Name	Idea	Algorithm	Advantage	Limitation
Artificial Neural Networks	Use of neural networks for error-tolerant graph matching	Utilizes Hopfield networks to minimize energy criteria and compatibility coefficients to evaluate matches. Optimization stabilized with Potts MFT or self-organizing Hopfield network.	Promising results in improving accuracy and efficiency of error-tolerant graph matching.	Potential challenges with training and interpreting neural networks. Limited applicability to certain types of graphs.
Relaxation Labeling	Use of relaxation labeling techniques for error-tolerant graph matching	Graph matching problem formulated as a labeling problem, with iterative refinement process using Gaussian probability distributions to model compatibility coefficients. Initial labeling based on node attributes and connectivity.	Extension of framework to improve accuracy and incorporate edge labels. Successful application to special kinds of graphs, such as trees .	Limited applicability to certain types of graphs. May require significant computation time for large graphs. Dependence on proper initialization of labeling process for accuracy.
Spectral Methods	Use of spectral methods for graph matching	Graphs represented by eigendecomposition of their structural matrix, allowing matching based on derived features. Eigendecomposition of adjacency or Laplacian matrix is invariant to node permutation.	Offers potential for efficient and accurate graph matching.	Sensitive to structural errors such as missing or spurious nodes. Limited applicability to labeled graphs or constrained label αbets.
Graph Kernel	Kernel methods for graph matching	Convolution, random walk, and diffusion kernels used. Convolution kernels infer similarity from parts in complex objects. Random walk kernels measure similarity based on matching walks in graphs. Diffusion kernels defined with respect to a base similarity measure.	Offers potential for efficient and accurate graph matching. Can handle continuous labels and non-identically labeled walks.	Kernel choice and parameter selection may impact performance. Limited applicability to certain types of graphs. Computationally intensive for large graphs.
Miscellaneous Methods	Other error-tolerant graph matching methods	EM algorithm, replicator equations, graduated assignment, random walks, least-squares and interpolation theory algorithms, random graphs	Offers potential for efficient and accurate graph matching.	Performance may vary depending on specific algorithm and graph characteristics. Limited applicability to certain types of graphs. Computationally intensive for large graphs.

Graph Matching for Data Mining and Information Retrieval

1. **What are the limitations of the conventional subgraph isomorphism algorithm in graph-based data mining?**
    1. The presence of a large number of irrelevant attributes in the underlying database graph
    2. A limited answer format
    3. The inability to impose constraints on query attributes to model dependencies or restrictions

2. **How does the generalized subgraph isomorphism retrieval procedure overcome these limitations?**
   By allowing for the masking out of irrelevant attributes, using variables to retrieve more specific information, and defining queries with constrained variables.

3. **How does the knowledge mining and information retrieval method based on the generalized subgraph isomorphism retrieval procedure work?**
    1. Extracts information from a large database graph by specifying a query graph.
    2. Unlike conventional methods, this method defines nodes and edges in the graph with type and attribute labels.
    3. Use variables to retrieve specific attribute values from the database graph, and allows for defining more specific queries that can model dependencies or restrictions.

Graph Matching for Data Mining and Information Retrieval

1. The approach proposed is based on specifying a query graph to extract information from a large database graph.
2. Nodes and edges in the graph are labeled by type and attributes.
3. Query graphs can include don't care symbols and variables to define attributes to be returned or express constraints.
4. Matches between query and database graphs require each edge in the query graph to be included in the database graph.
5. Variables indicate which attribute values are to be returned as an answer to a query.
6. An answer is generated for each match, with values of attributes in the database graph corresponding to the answer variables under the match.
7. The process involves finding subgraph isomorphisms between the query and database graphs.
8. Constraints on variables can be used to evaluate if a subgraph isomorphism exists.
9. Matches must satisfy the constraints on the variables in the query graph.
10. A match is a generalized subgraph isomorphism between the query and database graphs.

Graph Matching for Data Mining and Information Retrieval

11. The answer to a query can be yes or no, depending on whether the query graph exists as a substructure in the database graph and whether it contains any answer variables.
12. An individual answer is generated for each match when answer variables are defined in the query graph.
13. The proposed system currently returns no information when no match is found, which may be undesirable.
14. Graph edit distance can be used to allow for some tolerance to errors and find a minimally modified query that can be matched to the database graph.
15. An algorithmic procedure is described for finding matches between a query and a database graph by constructing all possible mappings, but it has exponential complexity.
16. For large query graphs, a novel approximate approach for querying graph databases is introduced.
17. The approach selects important nodes from the query graph, matches them against the database graph nodes, and applies a bipartite optimization procedure to establish a one-to-one correspondence between query and database nodes.
18. The initial graph match is iteratively extended by trying to map nearby nodes to database nodes.
19. This procedure is suboptimal but applicable to very large query graphs.

Vector Space Embeddings of Graphs via Graph Matching

1. Classification and Clustering in Intelligent Information Processing

1. Classification assigns unknown input objects to classes, while clustering divides given objects into homogeneous groups.
2. Most algorithms for classification and clustering are designed for object representations given in terms of feature vectors, resulting in a lack of algorithmic tools for graph classification and clustering.

2. Limitations of Graph Dissimilarity Measures

1. Graph dissimilarity measures can be defined via specific graph matching procedures, but this is often not sufficient for standard algorithms in intelligent information processing.
2. Graph distance based pattern recognition is basically limited to nearest-neighbor classification and k-medians clustering.

3. Graph Embedding into Vector Spaces

1. Graph embedding into vector spaces is a promising direction to overcome the limitation of graph dissimilarity measures and establish access to the rich repository of algorithmic tools developed for vectorial representations.
2. Graph embedding techniques include features derived from the eigendecomposition of graphs, string edit distance applied to the eigensystem of graphs, spectral decomposition of the Laplacian matrix of a graph, and using the relationship between the Laplace-Beltrami operator and the graph Laplacian to embed a graph in a Riemannian manifold.

Vector Space Embeddings of Graphs via Graph Matching

4. **Introduction to Dissimilarity-based Graph Embedding**
   1. Graph embedding procedures based on dissimilarity representation and graph matching were originally proposed to map feature vectors into dissimilarity spaces.
   2. Later, the idea was generalized to string-based object representation and to the domain of graphs.
   3. Graphs from a given problem domain are mapped to vector spaces by computing the distance to some predefined prototype graphs.

5. **Formal Definition of Graph Embedding**
   1. Let G be a graph domain, T = {g1, . . . , gn} be a training set of given graphs, and P = {p1, . . . , pn} be a set of prototype graphs.
   2. The mapping function φ_n^P : G → $\mathbb{R}$n is defined as φ_n^P (g) = (d(g, p1), . . . , d(g, pn)), where d(g, pi) is any graph dissimilarity measure between graph g and the i-th prototype graph.
   3. This procedure leads to an n-dimensional vector (d1, . . . , dn) for each embedded graph g, where each axis corresponds to a prototype graph pi and the coordinate values are the distances of g to the elements in P.

Vector Space Embeddings of Graphs via Graph Matching

6.Generalization of Graph Embedding

1. The procedure is further generalized towards Lipschitz embeddings by using sets of prototypes for embedding the graphs via dissimilarities.
2. Sets of prototypes P1, . . . , Pn are used instead of singleton reference sets (i.e. prototypes p1, . . . , pn) for embedding the graphs via dissimilarities.

7. Vector Space Representation of Graph Embedding

1. The definition of graph embedding leads to a vector space representation where each axis corresponds to a prototype graph pi $\in$ P and the coordinate values of an embedded graph g are the distances of g to the elements in P.
2. This allows any graph g from the training set T as well as any other graph set S to be transformed into a vector of real numbers.
3. In [65], the procedure is further generalized towards Lipschitz embeddings by using sets of prototypes P_1, . . . , Pn for embedding the graphs via dissimilarities.

Vector Space Embeddings of Graphs via Graph Matching

8. **Use of Graph Edit Distance for Dissimilarity Measure**
    1. The embedding procedure proposed in [62] uses graph edit distance as the graph dissimilarity measure.
    2. However, any other graph dissimilarity measure can be used as well.
    3. Using graph edit distance allows the method to deal with a large class of graphs and to be highly robust against various graph distortions.
    4. The approach is highly flexible in the graph definition, without imposing restrictions on the type of underlying graph, unlike other graph embedding techniques.

9. **Relationship to Kernel Methods**
    1. Dissimilarity embeddings are closely related to kernel methods.
    2. The difference is that in the kernel approach, objects are described by pairwise kernel functions, while in the dissimilarity approach, they are described by pairwise dissimilarities.
    3. The kernel values are interpreted as dot products in some implicitly existing feature space, while in the dissimilarity approach, the set of dissimilarities is interpreted as a novel vectorial description of the object under consideration.
    4. The dissimilarity approach obtains an explicit dissimilarity space, rather than an implicit feature space as in the kernel approach.

Section10
A Survey Of Algorithms For Dense Subgraph Discovery

Types of Dense Components Density definitions

Two classes of density definitions	Measures	Relationship
Absolute density	Relaxations of the pure clique measure	Only interested in cliques, fully-connected subgraphs of maximum density
Relative density	No preset level for what is sufficiently dense. Compares the density of one region to another, with the goal of finding the densest regions	Closely related to clustering and in fact shares many features with it

Types of Dense Components Density definitions

1. Definition of the density of S:

- The ratio of the total weight of edges in E(S) to the number of possible edges among ｜S｜ vertices.
- For an undirected graph, E(S) is the set of induced edges on S: $E(S) = \{(u,v) \in E | u,v \in S\}$.
- If the graph is unweighted, then the numerator is simply the number of actual edges, and the maximum possible density is 1.
- If the graph is weighted, the maximum density is unbounded.

1. Formulas for the density of a graph:

We give the formulas for an undirected graph:

$$den(S) = \frac{2|E(S)|}{|S|(|S|-1)}$$

The formulas for a directed graph lack the factor of 2

For the weighted graph:

$$den_W(S) = \frac{2\sum_{u,v \in S} w(u,v)}{|S|(|S|-1)}$$

Types of Dense Components Definitions of Dense Components

1. What is dense components ?

- All dense components are either cliques, which represent the ideal, or some relaxation of the ideal.
- Three relaxations categories: density, degree, and distance.
- Each relaxation can be quantified as either a percentage factor or a subtractive amount.

2.Types of Dense Components:

Component	Reference	Formal definition	Description
Clique		$\exists(i,j), i \neq j \in S$	Every vertex connects to every other vertex in S.
Quasi-Clique (density-based)	[1]	$den(S) \geq \gamma$	S has at least $\gamma\|S\|(\|S\|-1)/2$ edges. Density may be imbalanced within S.
Quasi-Clique (degree-based)	[36]	$\delta_S(u) \geq \gamma * (k-1)$	Each vertex has γ percent of the possible connections to other vertices. Local degree satisfies a minimum. Compare to K-core and K-plex.
K-core	[45]	$\delta_S(u) \geq k$	Every vertex connects to at least k other vertices in S. A clique is a (k-1)-core.
K-plex	[46]	$\delta_S(u) \geq \|S\| - k$	Each vertex is missing no more than $k-1$ edges to its neighbors. A clique is a 1-plex.
Kd-clique	[34]	$diam_G(S) \leq k$	The shortest path from any vertex to any other vertex is not more than k. An ordinary clique is a 1d-clique. Paths may go outside S.
K-club	[37]	$diam(S) \leq k$	The shortest path from any vertex to any other vertex is not more than k. Paths may not go outside S. Therefore, every K-club is a K-clique.

Algorithms for Detecting Dense Components in a Single Graph

Some more time efficient solutions were considered as for the clique problem is NP-hard

The table gives an overview of the major algorithmic approaches:

Algorithm Type	Component Type	Example	Comments
Enumeration	Clique	[12]	
	Biclique	[35]	
	Quasi-clique	[33]	min. degree for each vertex
	Quasi-biclique	[47]	
	k-core	[7]	
Fast Heuristic Enumeration	Maximal biclique	[30]	nonoverlapping
	Quasi-clique/biclique	[13]	spectral analysis
	Relative density	[18]	shingling
	Maximal quasi-biclique	[32]	balanced noise tolerance
	Quasi-clique, k-core	[52]	pruned search; visual results with upper-bounded estimates
Bounded Approximation	Max. average degree	[14]	undirected graph: 2-approx. directed graph: 2+ϵ-approx.
	Densest subgraph, $n \geq k$	[4]	1/3-approx.
	Subgraph of known density θ	[3]	finds subgraph with density $\Omega(\theta / \log \Delta)$

Exact Enumeration Approach

To enumerate all cliques, as even for modest-sized graphs, a graph may contain up to $3^{n/3}$maximal cliques.
So it is crucial to find the most effective algorithm.

Name	Idea	Algorithm	Advantage	Limitation
Quasi-clique Enumeration.	A graph mining technique used to find subgraphs that are not necessarily fully connected but have a high density of edges.	Quick algorithm: A quasi-clique enumeration algorithm that integrates novel pruning techniques based on the degree of vertices with a traditional depth-first search framework. The purpose of the Quick algorithm is to prune unqualified vertices as soon as possible.	Provide more flexibility in the components being sought and more opportunities for pruning the search space compared to exact cliques.	May not be sufficient for more complex graph structures and not be applicable to all types of graphs.
K-Core Enumeration	Greedy algorithms Assign each vertex with a core number to which it belongs	Repeatedly eliminates vertices with the lowest degree, decreases the degrees of their higher-degree neighbors, and reorders the remaining vertices in the queue.	The algorithm is efficient, easy to understand and implement, and can calculate the k-core of a graph quickly	May not be able to handle certain types of graph data, especially those with complex structures. and not find all k-cores if the graph is not connected. The results may not be unique due to multiple ways of eliminating vertices with degrees less than k.

Heuristic Approach

Name	Idea	Algorithm	Advantage	Limitation
Shingling Technique	Based on shingling for discovering large dense bipartite subgraphs in massive graphs. The algorithm converts each dense component with arbitrary size into shingles with constant size, making it efficient for single large graphs and easily extendable for streaming graph data.	The algorithm extracts shingles using the (s, c) shingling algorithm, which converts each string into an integer and generates a temporary set of integers, from which the s smallest elements are selected and concatenated together to form a new string that is hashed to get one shingle. The algorithm also uses a fingerprinting vector and similarity measure to estimate the similarity of two sets.	Efficient and practical for single large graphs and can be easily extended for streaming graph data. It can discover large dense bipartite subgraphs in massive graphs and estimate the similarity of sets using shingles and fingerprinting.	May not perform well on graphs with low density or small dense subgraphs. May also require significant computational resources when dealing with very large graphs. Additionally, the effectiveness of the algorithm may depend on the choice of parameters such as the size of the shingles and number of permutations used.
GRASP Algorithm	Utilize the Greedy Randomized Adaptive Search Procedure (GRASP) to produce feasible solutions and perform local optimization in a multi-start iterative process. Their algorithm aims to build a maximal γ-clique by selecting vertices with a large number of γ-neighbors and high degree with respect to the current solution.	Add vertices from the set of γ-vertices with respect to the current solution that have the potential to increase the solution's potential. The potential of a vertex set is defined as the number of edges minus a penalty term based on the size of the set and the desired clique size. The potential difference of a vertex is defined as the difference in potential when the vertex is added to the current solution. The algorithm selects vertices with a large potential difference and adds them to the current solution until no more vertices can be added.	Provides a way to incrementally construct quasi-dense components and build maximal γ-cliques efficiently using a multi-start iterative process. The algorithm also incorporates a novel evaluation measure that enables the selection of good vertices to add to the current solution.	Prohibitively expensive for very large graphs. The effectiveness of the algorithm may also depend on the choice of parameters such as the desired clique size and the penalty term used in the potential calculation.

Frequent Dense Components

The dense component discovery problem can be extended to consider a dataset consisting of a set of graphs $D = \{G1, \cdots, Gn\}$.

In this case, we have two criteria for components:

1.Must be dense and they must occur frequently.
2.The density requirement can be any of our earlier criteria.
3.The frequency requirement says that a component satisfies a minumum ***support*** threshold;

minumum *support* threshold : it appears in at least a certain number of graphs

Frequent Patterns with Density Constraints

1. **Approach:** Impose a density constraint on the patterns discovered by frequent pattern mining
2. **Criterion:** Use the minimum cut clustering criterion, where a component must have an edge cut less than or equal to k, which is equivalent to a k-core criterion
3. **Requirement:** Each frequent pattern must be closed, meaning it does not have any super-graph with the same support level
4. **Two approaches:** Pattern growth and pattern reduction
5. **Pattern growth:** Incrementally add adjacent edges to a small subgraph until it satisfies both the frequency and density requirements and becomes closed
6. **Pattern reduction:** Intersect the edge set of the working set with the edges of the next graph, decompose it into k-core subgraphs, and recursively call pattern reduction for each dense subgraph. The algorithm records the dense subgraphs that survive enough intersections to be considered frequent.
7. **Trade-off:** The greedy removal of edges at each iteration quickly reduces the working set size, leading to fast execution time. The trade-off is that we prune away edges that might have contributed to a frequent dense component.
8. **Heuristic:** Order the graphs by decreasing overall density as a useful heuristic.
9. **Performance:** Pattern reduction works better for high connectivity but low support threshold, while pattern growth works better for high support but only modest connectivity.

Dense Components with Frequency Constraint

1. **Perspective:** Provide a simple meta-algorithm on top of an existing dense component algorithm
2. **Input requirement:** A relation graph set
3. **Process:** Derive two new graphs from the input graphs: the Summary Graph and the Second-Order Graph
4. **Summary Graph:** Gˆ = (V, ˆE), where an edge exists if it appears in at least k graphs in D
5. **Second-Order Graph:** F = (V × V, EF), where each edge in D is transformed into a vertex in F, and two vertices in F are connected if they have similar support patterns in D
6. **Coherent dense subgraphs:** Find coherent dense subgraphs, where a subgraph S qualifies if its vertices form a dense component in ˆG and if its edges form a dense component in F
7. **Density in ˆG:** The component′ s edges occur frequently when considering the whole relation graph set D
8. **Density in F:** Ensures that these frequent edges are coherent, that is, they tend to appear in the same graphs
9. **Algorithm:** Hu uses a modified version of Hartuv and Shamir′ s HCS mincut algorithm to efficiently find dense subgraphs
10. **Scalability:** Hu's approach scales well with the number of graphs since it converts any n graphs into only 2 graphs
11. **Drawback:** The potentially large size of the second-order graph, which could become a clique of size ｜E｜ with O(｜E｜2) edges in the worst case where all n graphs are identical.

Enumerating Cross-Graph Quasi-Cliques

1. **Problem:** Finding cross-graph quasi-cliques (CGQC) using the balanced quasi-clique definition on a set of graphs D = {G1, ..., Gn} on the same set of vertices U, corresponding completeness parameters γ1, ..., γn, and a minimum component size minS
2. **Definition of CGQC:** All subsets of vertices of cardinality ≥ minS such that when each subset is induced upon graph Gi, it will form a maximal γi-quasi-clique
3. **Enumeration:** Complete enumeration is #P-Complete. They employ a set enumeration tree to list all possible subsets of vertices, while taking advantage of some tree-based concepts, such as depth-first search and sub-tree pruning
4. **Pruning methods:** Several graph-theoretical pruning methods are derived to typically reduce the execution time
5. **Graph and tree properties utilized for pruning:**
    1. Upper bounds on the graph diameter diam(G) given γ and graph size n: for example, diam(G) ≤ n 1 if γ > 1/(n-1)
    2. Nk(u) = vertices within a distance k of u
    3. Reducing vertices: If δ(u) < γi(minS 1) or |Nk(u)| < (minS 1), then u cannot be in a CGQC
    4. Candidate projection: when traversing the tree, a child cannot be in a CGQC if it does not satisfy its parent′ s neighbor distance bounds Nki(Gi)
6. **Subtree pruning:** apply various rules on minS, redundancy, monotonicity
7. **Main algorithm:** Crochet

Section12
Mining Graph Patterns

Frequent Subgraph Mining

Definition Of Frequent Graph:

Given a labeled graph dataset $D = \{G1, G2, \dots, Gn\}$and a subgraph g, the supporting graph set of g is $Dg = \{Gi \mid g \subseteq Gi, Gi \in D\}$. The support of g is $support(g) = \frac{|Dg|}{|D|}$.
A frequent graph is a graph whose support is no less than a minimum support threshold.

An important property, called *anti-monotonicity*, is crucial to confine the search space of frequent subgraph mining.

Definition Of Anti-Monotonicity:

Anti-monotonicity means that a size-k subgraph is frequent only if all of its subgraphs are frequent.

Apriori-based Approach

1. **Apriori-based algorithms:**
Share similar characteristics with Apriori-based frequent itemset mining algorithms, start with small-size subgraphs and proceed in a bottom-up manner by joining similar but slightly different frequent subgraphs to generate new subgraphs of increased size.

2. **Overhead of Apriori-based algorithms:**
Considerable overhead when two size-k frequent subgraphs are joined to generate size-(k+1) candidate patterns

3. **Typical examples:**
   1. AGM by Inokuchi: Uses vertex-based candidate generation, increases subgraph size by one vertex in each iteration, joins two size-(k+1) frequent subgraphs only if they have the same size-k subgraph
   2. FSG by Kuramochi and Karypis : Adopts edge-based candidate generation, increases subgraph size by one edge in each iteration, merges two size-(k+1) patterns only if they share the same subgraph having k edges
   3. An edge-disjoint path-join algorithm by Vanetik: Classifies graphs by the number of disjoint paths they have, generates subgraph pattern with k+1 disjoint paths by joining subgraphs with k disjoint paths

Non-Apriori-based

1. **Non-Apriori-based algorithms:**
Developed to avoid the overhead of Apriori-based algorithms, most of which adopt the pattern-growth methodology.

2. **Advantages of non-Apriori-based algorithms:**
Can handle larger datasets and more complex subgraph patterns compared to Apriori-based algorithms.

3. **Examples include:**
    1. GSPAN by Yan and Han [30]: Uses a depth-first search strategy to grow a subgraph pattern by adding one edge at a time, and employs minimum DFS codes to prune duplicate and non-frequent patterns
    2. Gaston by Nijssen and Kok [24]: Uses an incremental depth-first search strategy to grow a subgraph pattern, and employs a frequent substructure caching mechanism to speed up the mining process
    3. FFSM by Inokuchi et al. [15]: Uses a tree structure to represent frequent subgraphs, and employs a pattern-growth approach to construct the tree in a top-down manner

Pattern-Growth Approach

1. **Pattern-growth graph mining algorithms:**
    1. **Examples**: gSpan, MoFa, FFSM, SPIN, and Gaston.
    2. **Inspired by**: PrefixSpan, TreeMinerV, and FREQT.
    3. **Approach**: Extend a frequent graph directly by adding a new edge in every possible position, avoiding expensive join operations.

2. **Potential problem:** Discovering duplicates.

3. **Solution:** gSpan's right-most extension technique, where only extensions take place on the right-most path of a given graph (the straight path from the starting vertex to the last vertex, according to a depth-first search on the graph).

4. **Constraint-based subgraph mining algorithms**:
    1. **Examples**:
        1. Mining closed graph patterns by Yan and Han, mining coherent subgraphs by Huan .
        2. mining closed and maximal frequent subtrees by Chi.
        3. discovering exact dense frequent subgraphs in relational graphs by Yan .
        4. mining frequent large-scale structures by Jin.

5. **Large-scale graph database mining:**
    1. **Method**: Disk-based frequent graph mining introduced by Wang et al.

6. **Comprehensive introduction:**
    1. **Surveys**: Washio and Motoda and Yan and Han's surveys on basic graph pattern mining algorithms, including Apriori-based and pattern-growth approaches.

Mining Significant Graph Patterns

1. Problem definition:

Given a graph database $D = \{G1, \ldots, Gn\}$ and an objective function F, the problem is to find significant subgraphs g such that $F(g) \geq \delta$, where δ is a significance threshold, or to find a subgraph $g*$ such that $g* = argmax_g F(g)$.

The efficient mining algorithm should find significant patterns directly without exhaustively generating the whole set of graph patterns.

2. **Algorithms:**
   1. **gboost :** Uses a boosting algorithm to iteratively select the most informative subgraph features and build a classification model, which is then used to identify significant subgraphs.
   2. **gPLS :** Adapts partial least squares regression to identify significant subgraphs that are most correlated with the response variable in a regression model.
   3. **LEAP :** Uses a combination of a genetic algorithm and a support vector machine classifier to identify significant subgraphs based on their ability to classify the graphs into different classes.
   4. **GraphSig :** Uses a graph signature representation to measure the similarity between graphs, and identifies significant subgraphs that are most discriminative between different classes of graphs.
   5. **Pruning techniques:** Each algorithm uses different pruning techniques to reduce the search space and avoid enumerating all possible subgraphs.

Mining Representative Orthogonal Graphs

1. **ORIGAMI algorithm:** proposed by Hasan , mines a set of α-orthogonal, β-representative graph patterns.
2. **α-orthogonal:** two graph patterns are α-orthogonal if their similarity is bounded by a threshold α. This constraint ensures controlled redundancy in the resulting pattern set.
3. **β-representative:** a graph pattern is β-representative of another pattern if their similarity is at least β. This constraint ensures representativeness of frequent graph patterns not reported in the α-orthogonal set.
4. **Compact summary:** the set of representative orthogonal graph patterns is a compact summary of the complete set of frequent subgraphs.
5. **Goal:** given user-specified thresholds $\alpha, \beta \in [0, 1]$, the goal is to mine an α-orthogonal, β-representative graph pattern set that minimizes the set of unrepresented patterns.

Problem Definition

1. **Problem definition:**
   Given a collection of graphs D and a similarity threshold $\alpha \in [0,1]$, a subset of graphs $R \subseteq D$ is α-orthogonal with respect to D if for any $Ga, Gb \in R, sim(Ga, Gb) \leq \alpha$ and for any $Gi \in D \backslash$R there exists a $Gj \in R$, $sim(Gi, Gj) > \alpha$. Given D, an α-orthogonal set R, and a similarity threshold $\beta \in [0,1]$, R represents a graph $G \in D$ if there exists some $Ga \in R$, such that $sim(Ga, G) \geq \beta. Let\ \Upsilon(R, D) = \{G \mid G \in D\ s.t. \exists Ga \in R, sim(Ga, G) \geq \beta\}$, then R is a β-representative set for $\Upsilon(R, D)$. The problem is to find the α-orthogonal, β-representative set for the set of all maximal frequent subgraphs M which minimizes the residue set size

2. **Algorithm framework:**
   The mining problem can be decomposed into two subproblems of maximal subgraph mining and orthogonal representative set generation

3. **Residue set:**
   The residue set of R is the set of unrepresented patterns in D, denoted as $\triangle (R, D) = D\{R \cup \Upsilon(R, D)\}$.

Randomized Maximal Subgraph Mining

1. **First step:**
    1. ORIGAMI mines a set of maximal subgraphs, on which the α-orthogonal, β-representative graph pattern set is generated.
2. **Motivation:**
    1. The number of maximal frequent subgraphs is much fewer than that of frequent subgraphs
    2. The maximal subgraphs provide a synopsis of the frequent ones to some extent.
    3. Thus, it is reasonable to mine the representative orthogonal pattern set based on the maximal subgraphs rather than the frequent ones.
3. **Sampling:**
    1. To avoid the infeasibility of mining all maximal subgraphs, ORIGAMI first finds a sample of the complete set of maximal frequent subgraphs.
4. **Diverse set:**
    1. The goal is to find a set of maximal subgraphs, which is as diverse as possible.
    2. To achieve this goal, ORIGAMI avoids using combinatorial enumeration to mine maximal subgraph patterns.
    3. Instead, it adopts a random walk approach to enumerate a diverse set of maximal subgraphs from the positive border of such maximal patterns.
5. **Random walk:**
    1. The randomized mining algorithm starts with an empty pattern and iteratively adds a random edge during each extension until a maximal subgraph M is generated and no more edges can be added.
    2. This process walks a random chain in the partial order of frequent subgraphs.

Randomized Maximal Subgraph Mining

7. **Extension:**
   1. To extend an intermediate pattern, Sk ⊆ M, it chooses a random vertex v from which the extension will be attempted.
   2. Then a random edge e incident on v is selected for extension.
   3. If no such edge is found, no extension is possible from the vertex.
   4. When no vertices can have any further extension in Sk, the random walk terminates and Sk = M is the maximal graph.
   5. On the other hand, if a random edge e is found, the other endpoint v' of this edge is randomly selected.
   6. By adding the edge e and its endpoint v', a candidate subgraph pattern Sk+1 is generated, and its support is computed.

8. **Probability:**
   1. The probability of a particular edge sequence leading from an empty graph to a maximal graph is given by $P[(e1e2 \dots en)] = P(e1) \prod_{i=2}^{n} P(ei|e1 \dots ei-1)$.

9. **Duplicate subgraphs:**
   1. Duplicate maximal subgraphs can be generated through multiple iterations following overlapping chains
   2. Or multiple iterations following different chains but leading to the same maximal pattern.

Randomized Maximal Subgraph Mining

10. Collision rate:

1. To avoid generating duplicate maximal subgraphs, a termination condition is designed based on an estimate of the collision rate of the generated patterns.
2. The collision rate keeps track of the number of duplicate patterns seen within the same or across different random walks.
3. As a random walk chain is traversed, ORIGAMI maintains the signature of the intermediate patterns in a bounded size hash table.
4. As an intermediate or maximal subgraph is generated, its signature is added to the hash table, and the collision rate is updated.
5. If the collision rate exceeds a threshold, the method could
    1. abort further extension along the current path and randomly choose another path;
    2. or trigger the termination condition across different walks, since it implies that the same part of the search space is being revisited.

Orthogonal Representative Set Generation

1. Given a set of maximal subgraphs $\widehat{\mathcal{M}}$, the next step is to extract an α-orthogonal β-representative set from it.
2. **Meta-graph:**
    1. A meta-graph $\Gamma(\widehat{\mathcal{M}})$ is constructed to measure similarity between graph patterns in $\widehat{\mathcal{M}}$, in which each node represents a maximal subgraph pattern, and an edge exists between two nodes if their similarity is bounded by α.
3. **Problem modeling:**
    1. The problem of finding an α-orthogonal pattern set can be modeled as finding a maximal clique in the similarity graph $\Gamma(\widehat{\mathcal{M}})$
4. **Goodness measure:**
    1. The size of the residue set is used to measure the goodness of an α-orthogonal set.
    2. An optimal α-orthogonal β-representative set is the one which minimizes the size of the residue set.
5. **NP-hardness:**
    1. The problem of finding an optimal α-orthogonal β-representative set is NP-hard.

Orthogonal Representative Set Generation

1. **Approximate algorithm:**
    1. ORIGAMI uses an approximate algorithm to solve the problem which guarantees local optimality.
2. **State transition:**
    1. The algorithm starts with a random maximal clique in the similarity graph $\Gamma(\widehat{\mathcal{M}})$ and tries to improve it through state transitions.
3. **Local neighbor:**
    1. At each state transition, another maximal clique which is a local neighbor of the current maximal clique is chosen.
4. **Improvement:**
    1. If the new state has a better solution, the new state is accepted as the current state and the process continues.
5. **Termination:**
    1. The process terminates when all neighbors of the current state have equal or larger residue sizes.
6. **Neighborhood constraints:**
    1. Two maximal cliques of different sizes are considered neighbors if they share exactly one fewer vertex than their size difference.
7. **Selective removal:**
    1. The state transition procedure selectively removes one vertex from the maximal clique of the current state
    2. Then expands it to obtain another maximal clique which satisfies the neighborhood constraints.

Conclusion

THE END

中国科学技术大学

University of Science and Technology of China

QiuWentao