Chapter 10

A SURVEY OF ALGORITHMS FOR DENSE SUBGRAPH DISCOVERY

Victor E. Lee
Department of Computer Science
Kent State University
Kent, OH 44242
vlee@cs.kent.edu

Ning Ruan
Department of Computer Science
Kent State University
Kent, OH 44242
nruan@cs.kent.edu

Ruoming Jin
Department of Computer Science
Kent State University
Kent, OH 44242
jin@cs.kent.edu

Charu Aggarwal
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598
charu@us.ibm.com

Abstract In this chapter, we present a survey of algorithms for dense subgraph discovery. The problem of dense subgraph discovery is closely related to clustering though the two problems also have a number of differences. For example, the problem of clustering is largely concerned with that of finding a fixed partition in the data, whereas the problem of dense subgraph discovery defines these dense components in a much more flexible way. The problem of dense subgraph discovery may wither be defined over single or multiple graphs. We explore both cases. In the latter case, the problem is also closely related to the problem of the frequent subgraph discovery. This chapter will discuss and organize the literature on this topic effectively in order to make it much more accessible to the reader.

Keywords: Dense subgraph discovery, graph clustering

1. Introduction

In almost any network, density is an indication of importance. Just as someone reading a road map is interesting in knowing the location of the larger cities and towns, investigators who seek information from abstract graphs are often interested in the dense components of the graph. Depending on what properties are being modeled by the graph's vertices and edges, dense regions may indicate high degrees of interaction, mutual similarity and hence collective characteristics, attractive forces, favorable environments, or critical mass.

From a theoretical perspective, dense regions have many interesting properties. Dense components naturally have small diameters (worst case shortest path between any two members). Routing within these components is rapid. A simple strategy also exists for global routing. If most vertices belong to a dense component, only a few selected inter-hub links are needed to have a short average distance between any two arbitrary vertices in the entire network. Commercial airlines employ this hub-based routing scheme. Dense regions are also robust, in the sense that many connections can be broken without splitting the component. A less well-known but equally important property of dense subgraphs comes from percolation theory. If a graph is sufficiently dense, or equivalently, if messages are forwarded from one node to its neighbors with higher than a certain probability, then there is very high probability of propagating a message across the diameter of the graph [20]. This fact is useful in everything from epidemiology to marketing.

Not all graphs have dense components, however. A sparse graph may have few or none. In order to understand this issue, we first need to define a formal notion of the words 'dense' and 'sparse'. We will address this issue shortly. A uniform graph is either entirely dense or not dense at all. Uniform graphs, however, are rare, usually limited to either small or artificially created ones. Due to the usefulness of dense components, it is generally accepted that their existence is the rule rather than the exception in nature and in human-planned networks [39].

Dense components have been identified in and have enhanced understanding of many types of networks; among the best-known are social networks [53, 44], the World Wide Web [30, 17, 11], financial markets [5], and biological systems [26]. Much of the early motivation, research, and nomenclature regarding dense components was in the field of social network analysis. Even before the advent of computers, sociologists turned to graph theory to formulate models for the concept of social cohesion. Clique, K-core, K-plex, and K-club are metrics originally devised to measure social cohesiveness [53]. It is not surprising that we also see dense components in the World Wide Web. In many ways, the Web is simply a virtual implementation of traditional direct human-human social networks.

Today, the natural sciences, the social sciences, and technological fields are all using network and graph analysis methods to better understand complex systems. Dense component discovery and analysis is one important aspect of network analysis. Therefore, readers from many different backgrounds will benefit from understanding more about the characteristics of dense components and some of the methods used to uncover them.

In the next section, we outline the graph terminology and define the fundamental measures of density to be used in the rest of the chapter. Section 3 categorizes the algorithmic approaches and presents representative implementations in more detail. Section 4 expands the topic to consider frequently-occurring dense components in a set of graphs. Section 5 provides examples of how these techniques have been applied in various scientific fields. Section 6 concludes the chapter with a look to the future.

2. Types of Dense Components

Different applications find different definitions of *dense component* to be useful. In this section, we outline the many ways to define a dense component, categorizing them by their important features. Understanding these features of the various types of components are valuable for deciding which type of component to pursue.

2.1 Absolute vs. Relative Density

We can divide density definitions into two classes, absolute density and relative density. An absolute density measure establishes rules and parameter values for what constitutes a dense component, independent of what is outside the component. For example, we could say that we are only interested in cliques, fully-connected subgraphs of maximum density. Absolute density measures take the form of relaxations of the pure clique measure.

On the other hand, a relative density measure has no preset level for what is sufficiently dense. It compares the density of one region to another, with the goal of finding the densest regions. To establish the boundaries of components, a metric typically looks to maximize the difference between intra-component connectedness and inter-component connectedness. Often but not necessarily, relative density techniques look for a user-defined number k densest regions. The alert reader may have noticed that relative density discovery is closely related to clustering and in fact shares many features with it.

Since this book contains another chapter dedicated to graph clustering, we will focus our attention on absolute density measures. However, we will have more so say about the relationship between clustering and density at the end of this section.

2.2 Graph Terminology

Let $G(V, E)$ be a graph with $|V|$ vertices and $|E|$ edges. If the edges are weighted, then $w(u)$ is the weight of edge u. We treat unweighted graphs as the special case where all weights are equal to 1. Let S and T be subsets of V. For an undirected graph, $E(S)$ is the set of induced edges on S: $E(S) = \{(u, v) \in E \,|u, v \in S\}$. Then, H_S is the induced subgraph $(S, E(S))$. Similarly, $E(S, T)$ designates the set of edges from S to T. $H_{S,T}$ is the induced subgraph $(S, T, E(S, T))$. Note that S and T are not necessarily disjoint from each other. If $S \cap T = \emptyset$, $H_{S,T}$ is a bipartite graph. If S and T are not disjoint (possibly $S = T = V$), this notation can be used to represent a directed graph.

A dense component is a maximal induced subgraph which also satisfies some density constraint. A component H_S is maximal if no other subgraph of G which is a superset of H_S would satisfy the density constraints. Table 10.1 defines some basic graph concepts and measures that we will use to define density metrics.

Table 10.1. Graph Terminology

Symbol	Description
$G(V, E)$	graph with vertex set V and edge set E
H_S	subgraph with vertex set S and edge set $E(S)$
$H_{S,T}$	subgraph with vertex set $S \cup T$ and edge set $E(S, T)$
$w(u)$	weight of edge u
$N_G(u)$	neighbor set of vertex u in G: $\{v\| \ (u, v) \in E\}$
$N_S(u)$	only those neighbors of vertex u that are in S: $\{v\| \ (u, v) \in S\}$
$\delta_G(u)$	(weighted) degree of u in $G : \sum_{v \in N_G(u)} w(v)$
$\delta_S(u)$	(weighted) degree of u in $S : \sum_{v \in N_S(u)} w(v)$
$d_G(u, v)$	shortest (weighted) path from u to v traversing any edges in G
$d_S(u, v)$	shortest (weighted) path from u to v traversing only edges in $E(S)$

We now formally define the **density of S**, $den(S)$, as the ratio of the total weight of edges in $E(S)$ to the number of possible edges among $|S|$ vertices. 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. The number of possible edges in an undirected graph of size n is $\binom{n}{2} = n(n-1)/2$. We give the formulas for an undirected graph; the formulas for a directed graph lack the factor of 2.

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

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

Some authors define density as the ratio of the number of edges to the number of *vertices*: $\frac{|E|}{|V|}$. We will refer to this as **average degree of S**.

Another important metric is the **diameter of S**, $diam(S)$. Since we have given two different distance measures, d_S and d_G, we accordingly offer two different diameter measures. The first is the standard one, in which we consider only paths within S. The second permits paths to stray outside S, if it offers a shorter path.

$$diam(S) = max\{d_S(u,v)|\ u,v \in S\}$$

$$diam_G(S) = max\{d_G(u,v)|\ u,v \in S\}$$

2.3 Definitions of Dense Components

We now present a collection of measures that have been used to define dense components in the literature (Table 10.2). To focus on the fundamentals, we assume unweighted graphs. In a sense, all dense components are either cliques, which represent the ideal, or some relaxation of the ideal. There relaxations fall into three categories: density, degree, and distance. Each relaxation can be quantified as either a percentage factor or a subtractive amount. While most of there definitions are widely-recognized standards, the name *quasi-clique* has been applied to any relaxation, with different authors giving different formal definitions. Abello [1] defined the term in terms of overall edge density, without any constraint on individual vertices. This offers considerable flexibility in the component topology. Several other authors [36, 32, 33] have opted to define quasi-clique in terms of minimum degree of each vertex. Li et al. [32] provide a brief overview and comparison of quasi-cliques. In our table, when the authorship of a specific metric can be traced, it is given. Our list is not exhaustive; however, the majority of definitions can be reduced to some combination of density, degree, and diameter.

Note that in unweighted graphs, cliques have a density of 1. Density-based quasi-cliques are only defined for unweighted graphs. We use the term *Kd-clique* instead of Mokken's original name *K-clique*, because K-clique is already defined in the mathematics and computer science communities to mean a clique with k vertices.

Table 10.2. 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.

Figure 10.1, a superset of an illustration from Wasserman and Faust [53], demonstrates each of the dense components that we have defined above.

- Cliques: {1,2,3} and {2,3,4}
- 0.8-Quasi-clique: {1,2,3,4} (includes $5/6 > 0.83$ of possible edges)
- 2-Core: {1,2,3,4,5,6,7}
- 3-Core: none
- 2-Plex: {1,2,3,4} (vertices 1 and 3 are missing one edge each)
- 2d-Cliques: {1,2,3,4,5,6} and {2,3,4,5,6,7} (In the first component, 5 connects to 6 via 7, which need not be a member of the component)
- 2-Clubs: {1,2,3,4,5}, {1,2,3,4,6}, and {2,3,5,6,7}

2.4 Dense Component Selection

When mining for dense components in a graph, a few additional questions must be addressed:

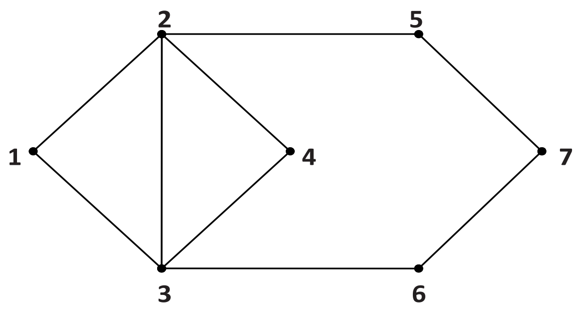

Figure 10.1. Example Graph to Illustrate Component Types

1 **Minimum size** σ: What is the minimum number of vertices in a dense component S? I.e., $|S| \geq \sigma$.

2 **All or top-N?**: One of the following criteria should be applied.

- Select all components which meet the size, density, degree, and distance constraints.
- Select the N highest ranking components that meet the minimum constraints. A ranking function must be established. This can be as simple as one of the same metrics used for minimum constraints (size, density, degree, distance, etc.) or a linear combination of them.
- Select the N highest ranking components, with no minimum constraints.

3 **Overlap**: May two components share vertices?

2.5 Relationship between Clusters and Dense Components

The measures described above set an absolute standard for what constitutes a dense component. Another approach is to find the most dense components on a relative basis. This is the domain of clustering. It may seem that clustering, a thoroughly-studied topic in data mining with many excellent methodologies, would provide a solution to dense component discovery. However, clustering is a very broad term. Readers interested in a survey on clustering may wish to consult either Jain, Murty, and Flynn [24] or Berkhin [8]. In the data mining community, *clustering* refers to the task of assigning similar or nearby items to the same group while assigning dissimilar/distant items to different groups. In most clustering algorithms, similarity is a relative concept; therefore it is potentially suitable for relative density measures. However, not all clustering algorithms are based on density, and not all types of dense components can be discovered with clustering algorithms.

Partitioning refers to one class of clustering problem, where the objective is to assign every item to exactly one group. A k-partitioning requires the result to have k groups. K-partitioning is not a good approach for identifying absolute dense components, because the objectives are at odds. Consider the well-known k-Means algorithm applied to a uniform graph. It will generate k partitions, because it must. However, the partitioning is arbitrary, changing as the seed centroids change.

In *hierarchical clustering*, we construct a tree of clusters. Conceptually, as well as in actual implementation, this can be either agglomerative (bottom-up), where the closest clusters are merged together to form a parent cluster, or divisive (top-down), where a cluster is subdivided into relatively distant child clusters. In basic greedy agglomerative clustering, the process starts by grouping together the two closest items. The pair are now treated as a single item, and the process is repeated. Here, pairwise distance is the density measure, and the algorithm seeks to group together the densest pair. If we use divisive clustering, we can choose to stop subdividing after finding k leaf clusters. A drawback of both hierarchical clustering and partitioning is that they do not allow for a separate "non-dense" partition. Even sparse regions are forced to belong to some cluster, so they are lumped together with their closest denser cores.

Spectral clustering describes a graph as a adjacency matrix W, from which is derived the Laplacian matrix $L = D - W$(unnormalized) or $L = I - D^{1/2}WD^{-1/2}$(normalized), where D is the diagonal matrix featuring each vertex's degree. The eigenvectors of L can be used as cluster centroids, with the corresponding eigenvalues giving an indication of the cut size between clusters. Since we want minimum cut size, the smallest eigenvalues are chosen first. This ranking of clusters is an appealing feature for dense component discovery.

None of these clustering methods, however, are suited for an absolute density criterion. Nor can they handle overlapping clusters. Therefore, some but not all clustering criteria are dense component criteria. Most clustering methods are suitable for relative dense component discovery, excluding k-partitioning methods.

3. Algorithms for Detecting Dense Components in a Single Graph

In this section, we explore algorithmic approaches for finding dense components. First we look at basic exact algorithms for finding cliques and quasi-cliques and comment on their time complexity. Because the clique problem is NP-hard, we then consider some more time efficient solutions. The algorithms can be categorized as follows: Exact enumeration (Section 3.1), Fast Heuristic Enumeration (Section 3.2), and Bounded Approximation Algorithms (Section 3.3). We review some recent works related to dense component discovery, concentrating on the details of several well-received algorithms.

The following table (Table 10.3) gives an overview of the major algorithmic approaches and lists the representative examples we consider in this chapter.

Table 10.3. Overview of Dense Component Algorithms

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 $\Omega(\theta/\log\Delta)$

3.1 Exact Enumeration Approach

The most natural way to discover dense components in a graph is to enumerate all possible subsets of vertices and to check if some of them satisfy the definition of dense components. In the following, we investigate some algorithms for discovering dense components by explicit enumeration.

Enumeration Approach. Finding maximal cliques in a graph may be straightforward, but it is time-consuming. The clique decision problem, deciding whether a graph of size n has a clique of size at least k, is one of Karp's 21 NP-Complete problems [28]. It is easy to show that the clique optimization problem, finding a largest clique in a graph, is also NP-Complete, because the optimization and decision problems each can be reduced in polynomial time to the other. Our goal is to enumerate all cliques. Moon and Moser showed that a graph may contain up to $3^{n/3}$ maximal cliques [38]. Therefore, even for modest-sized graphs, it is important to find the most effective algorithm.

One well-known enumeration algorithm for generating cliques was proposed by Bron and Kerbosch [12]. This algorithm utilizes the branch-and-bound technique in order to prune branches which are unable to generate a clique. The basic idea is to extend a subset of vertices, until the clique is maximal, by adding a vertex from a candidate set but not in a exclusion set. Let C be the set of vertices which already form a clique, $Cand$ be the set of vertices which may potentially be used for extending C, and $NCand$ be the set of vertices which are not allowed to be candidates for C. $N(v)$ are the neighbors of vertex v. Initially, C and $NCand$ are empty, and $Cand$ contains all vertices in the graph. Given C, $Cand$ and $NCand$, we describe the Bron-Kerbosch algorithm below. The authors experimentally observed $O(3.14^n)$, but did not prove their theoretical performance.

Algorithm 6 CliqueEnumeration(C,$Cand$,$NCand$)

```
if $Cand = \emptyset$ and $NCand = \emptyset$ then
    output the clique induced by vertices $C$;
else
    for all $v_i \in Cand$ do
        $Cand \leftarrow Cand \setminus \{v_i\}$;
        call $CliqueEnumeration(C \cup \{v_i\}, Cand \cap N(v_i), NCand \cap N(v_i))$;
        $NCand \leftarrow NCand \cup \{v_i\}$;
    end for
end if
```

Makino et al. [35] proposed new algorithms making full use of efficient matrix multiplication to enumerate all maximal cliques in a general graph or bicliques in a bipartite graph. They developed different algorithms for different types of graphs (general graph, bipartite, dense, and sparse). In particular, for a sparse graph such that the degree of each vertex is bounded by $\Delta \ll |V|$, an algorithm with $O(|V||E|)$ preprocessing time, $O(\Delta^4)$ time delay (i.e, the bound of running time between two consecutive outputs) and $O(|V| + |E|)$ space is developed to enumerate all maximal cliques. Experimental results demonstrate good performance for sparse graphs.

Quasi-clique Enumeration. Compared to exact cliques, quasi-cliques provide both more flexibility of the components being sought as well as more opportunities for pruning the search space. However, the time complexity generally remains NP-complete. The $Quick$ algorithm, introduced in [33], provided an illustrative example. The authors studied the problem of mining maximal degree-based quasi-cliques with size at least min_size and degree of each vertex at least $\lceil\gamma(|V|-1)\rceil$. The $Quick$ algorithm integrates some novel pruning techniques based on degree of vertices with a traditional depth-first search framework to prune the unqualified vertices as soon as possible. Those pruning techniques also can be combined with other existing algorithms to achieve the goal of mining maximal quasi-cliques.

They employ these established pruning techniques based on diameter, minimum size threshold, and vertex degree. Let $N_k^G(v) = \{u|dist^G(u,v) \leq k\}$ be the set of vertices that are within a distance of k from vertex v, $indeg^X(u)$ denotes the number of vertices in X that are adjacent to u, and $exdeg^X(u)$ represents the number of vertices in $cand\_exts(X)$ that are adjacent to u. All vertices are sorted in lexicographic order, then $cand\_exts(X)$ is the set of vertices after the last vertex in X which can be used to extend X. For the pruning technique based on graph diameter, the vertices which are not in $\cap_{v\in X}N_k^G(v)$ can be removed from $cand\_exts(X)$. Considering the minimum size threshold, the vertices whose degree is less than $\lceil\gamma(min\_size-1)\rceil$ should be removed.

In addition, they introduce five new pruning techniques. The first two techniques consider the lower and upper bound of the number of vertices that can be used to extend current X. The first pruning technique is based on the upper bound of the number of vertices that can be added to X concurrently to form a γ-quasi-clique. In other words, given a vertex set X, the maximum number of vertices in $cand\_exts(X)$ that can be added into X is bounded by the minimal degree of the vertices in X; The second one is based on the lower bound of the number of vertices that can be added to X concurrently to form a γ-quasi-clique. The third technique is based on critical vertices. If we can find some critical vertices of X, then all vertices in $cand\_exts(X)$ and adjacent to critical vertices are added into X. Technique 4 is based on cover vertex u which maximizes the size of $C_X(u) = cand\_exts(X) \cap N^G(u) \cap (\cap_{v\in X\wedge(u,v)\ni E}N^G(v))$.

Lemma 10.1. *[33] Let X be a vertex set and u be a vertex in $cand\_exts(X)$ such that $indeg^X(u) \geq \lceil\gamma \times |X|\rceil$. If for any vertex $v \in X$ such that $(u,v) \in E$, we have $indeg^X(v) \geq \lceil\gamma \times |X|\rceil$, then for any vertex set Y such that $G(Y)$ is a γ-quasi-clique and $Y \subseteq (X \cup (cand\_exts(X) \cap N^G(u) \cap (\cap_{v\in X\wedge(u,v)\ni E}N^G(v))))$, $G(Y)$ cannot be a maximal γ-quasi-clique.*

From the above lemma, we can prune the $C_X(u)$ of cover vertex u from $cand\_exts(X)$ to reduce the search space. The last technique, the so-called lookahead technique, is to check if $X \cup cand\_exts(X)$ is γ-quasi-clique. If so, we do not need to extend X anymore and reduce some computational cost. See Algorithm $Quick$ above.

Algorithm 7 Quick($X, cand\_exts(X), \gamma, min\_size$)

```
find the cover vertex $u$ of $X$ and sort vertices in $cand\_exts(X)$;
for all $v \in cand\_exts(X) - C_X(u)$ do
    apply minimum size constraint on $|X| + |cand\_exts(X)|$;
    apply lookahead technique (technique 5) to prune search space;
    remove the vertices that are not in $N_k^G(v)$;
    $Y \leftarrow X \cup \{u\}$;
    calculate the upper bound and lower bound of the number vertices to be
    added to $Y$ in order to form $\gamma$-quasi-clique;
    recursively prune unqualified vertices (techniques 1,2);
    identify critical vertices of $Y$ and apply pruning (technique 3);
    apply existing pruning techniques to further reduce the search space;
end for
return $\gamma$-quasi-cliques;
```

***K*-Core Enumeration.** For k-cores, we are happily able to escape NP-complete time complexity; greedy algorithms with polynomial time exist. Batagelj et al. [7] developed a efficient algorithm running in $O(m)$ time, based on the following observation: given a graph $G = (V, E)$, if we recursively eliminate the vertices with degree less than k and their incident edges, the resulting graph is a k-core. The algorithm is quite simple and can be considered as a variant of [29]. This algorithm attempts to assign each vertex with a core number to which it belongs. At the beginning, the algorithm places all vertices in a priority queue based on minimim degree. For each iteration, we eliminate the first vertex v (i.e, the vertex with lowest degree) from the queue. After then, we assign the degree of v as its core number. Considering v's neighbors whose degrees are greater than that of v, we decrease their degrees by one and reorder the remaining vertices in the queue. We repeat such procedure until the queue is empty. Finally, we output the k-cores based on their assigned core numbers.

3.2 Heuristic Approach

As mentioned before, it is impractical to exactly enumerate all maximal cliques, especially for some real applications like protein-protein interaction networks which have a very large number of vertices. In this case, fast heuristic methods are available to address this problem. These methods are able to efficiently identify some dense components, but they cannot guarantee to discover all dense components.

Shingling Technique. Gibson et al. [18] propose an new algorithm based on shingling for discovering large dense bipartite subgraphs in massive graphs. In this paper, a dense bipartite subgraph is considered a cohesive group of vertices which share many common neighbors. Since this algorithm utilizes the shingling technique to convert each dense component with arbitrary size into shingles with constant size, it is very efficient and practical for single large graphs and can be easily extended for streaming graph data.

We first provide some basic knowledge related to the shingling technique. Shingling was firstly introduced in [11] and has been widely used to estimate the similarity of web pages, as defined by a particular feature extraction scheme. In this work, shingling is applied to generate two constant-size fingerprints for two different subsets A and B from set S of a universe U of elements, such that the similarity of A and B can be computed easily by comparing fingerprints of A and B, respectively. Assuming π is a random permutation of the elements in the ordered universe U which contains A and B, the probability that the smallest element of A and B is the same, is equal to the Jaccard coefficient. That is,

$$Pr[\pi^{-1}(min_{a\in A}\{\pi(a)\}) = \pi^{-1}(min_{b\in B}\{\pi(b)\})] = \frac{|A\cap B|}{|A\cup B|}$$

Given a constant number c of permutations $\pi_1, \cdots, \pi_c$ of U, we generate a fingerprinting vector whose i-th element is $min_{a\in A}\pi_i(a)$. The similarity between A and B is estimated by the number of positions which have the same element with respect to their corresponding fingerprint vectors. Furthermore, we can generalize this approach by considering every s-element subset of entire set instead of the subset with only one element. Then the similarity of two sets A and B can be measured by the fraction of these s-element subsets that appear in both. This actually is an agreement measure used in information retrieval. We say each s-element subset is a *shingle*. Thus this feature extraction approach is named the (s, c) shingling algorithm. Given a n-element set $A = \{a_i, 0 \le i \le n\}$ where each element a_i is a string, the (s, c) shingling algorithm tries to extract c shingles such that the length of each shingle is exact s. We start from converting each string a_i into a integer x_i by a hashing function. Following that, given two random integer vectors R, S with size c, we generate a n-element temporary set $Y = \{y_i, 0 \le i \le n\}$ where each element $y_i = R_j \times x_i + S_j$. Then the s smallest elements of Y are selected and concatenated together to form a new string y. Finally, we apply a hash function on string y to get one shingle. We repeat such procedure c times in order to generate c shingles.

Remember that our goal is to discover dense bipartite subgraphs such that vertices in one side share some common neighbors in another side. Figure 10.2 illustrates a simple scenario in a web community where each web page

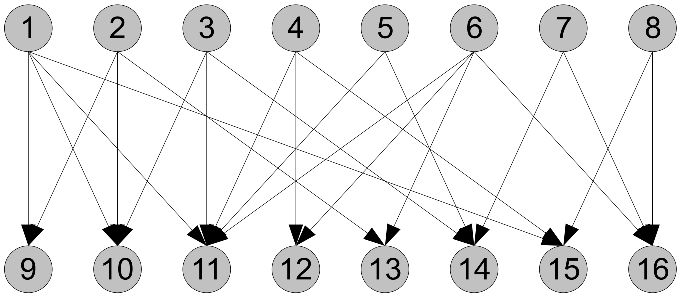

Figure 10.2. Simple example of web graph

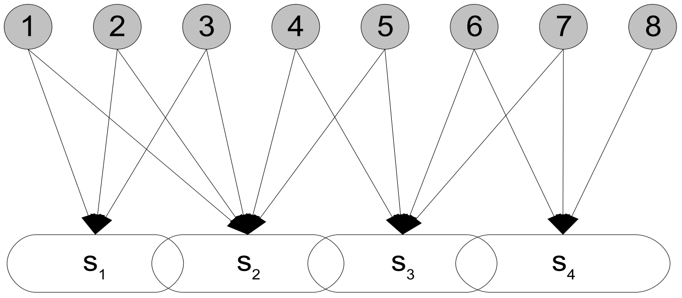

Figure 10.3. Illustrative example of shingles

in the upper part links to some other web pages in the lower part. We can describe each upper web page (vertex) by the list of lower web pages to which it links. In order to put some vertices into the same group, we have to measure the similarity of the vertices which denotes to what extent they share common neighbors. With the help of shingling, for each vertex in the upper part, we can generate constant-size shingles to describe its outlinks (i.e, its neighbors in the lower part). As shown in Figure 10.3, the outlinks to the lower part are converted to shingles s_1, s_2, s_3, s_4. Since the size of shingles can be significantly smaller than the original data, much computational cost can be saved in terms of time and space.

In the paper, Gibson et al. repeatedly employ the shingling algorithm for converting dense component into constant-size shingles. The algorithm is a two-step procedure. Step 1 is recursive shingling, where the goal is to exact some subsets of vertices where the vertices in each subset share many common neighbors. Figure 10.4 illustrates the recursive shingling process for a graph ($\Gamma(V)$ is the outlinks of vertices V). After the first shingling process, for each vertex $v \in V$, its outlinks $\Gamma(v)$ are converted into a constant size of first-level shingles v'. Then we can transpose the mapping relation E_0 to E_1 so that each shingle in v' corresponds to a set of vertices which share this shingle. In other words, a new bipartite graph is constructed where each vertex in one

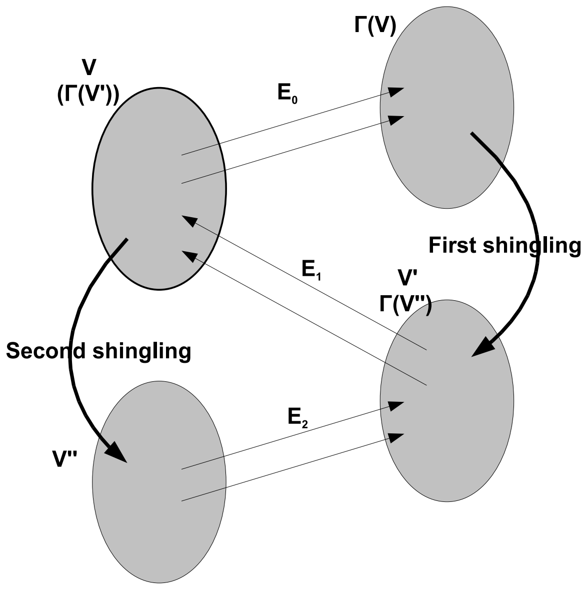

Figure 10.4. Recursive Shingling Step

part represents one shingle, and each vertex in another part is the original vertex. If there is a edge from shingle v' to vertex v, v' is one of the shingles for v's outlinks generated by shingling. From now on, V is considered as $\Gamma(V')$. Following the same procedure, we apply shingling on V' and $\Gamma(V')$. After the second shingling process, V is converted into a constant-size $V^{''}$, so-called second-level shingles. Similar to the transposition in the first shingling process, we transpose E_1 to E_2 and obtain many pairs $< v'', \Gamma(v'') >$ where v'' is second-level shingles and $\Gamma(v'')$ are all the first-level shingles that share a second-level shingle. Step 2 is clustering, where the aim is to merge first-level shingles which share some second-level shingles. Essentially, merges a number of biclique subsets into one dense component. Specifically, given all pairs $< v'', \Gamma(v'') >$, a traditional algorithm, namely $UnionFind$, is used to merge some first-level shingles in $\Gamma(V'')$ such that any two first-level shingles at least share one second-level shingle. To the end, we map the clustering results back to the vertices of the original graph and generate one dense bipartite subgraph for each cluster. The entire algorithm is presented in Algorithm ***DiscoverDenseSubgraph***.

GRASP Algorithm. As mentioned in Table 10.2, Abello et al. [1] were one of the first to formally define quasi-dense components, namely γ-cliques, and to investigate their discovery. They utilize a existing framework known as a Greedy Randomized Adaptive Search Procedure (GRASP). Their paper makes two major contributions. First, they propose a novel evaluation measure

Algorithm 8 DiscoverDenseSubgraph(c_1, s_1, c_2, s_2)

```
apply recursive shingling algorithms to obtain first- and second-level shingles;
let $S =< s, \Gamma(s) >$ be first-level shingles;
let $T =< t, \Gamma(t) >$ be second-level shingles;
apply clustering approach to get the clustering result $\mathcal{C}$ in terms of first-level shingles;
for all $C \in \mathcal{C}$ do
    output $\cup_{s \in C} \Gamma(s)$ as a dense subgraph;
end for
```

on potential improvement of adding a new vertex to a current quasi-clique. This measure enables the construction of quasi-cliques incrementally. Second, a semi-external memory algorithm incorporating edge pruning and external breath first search traversal is introduced to handle very large graphs. The basic idea is to decompose a large graph into several small components, then process each of them using GRASP. In the following, we concentrate our efforts on discussing the first point and its usage in GRASP. Interested readers can refer to [1] for the details of the second algorithm.

GRASP is a multi-start iterative process, with two steps per iteration, initial construction and local optimization. The initial construction step aims to produce a feasible solution for subsequent processing. For local optimization, we examine the neighborhood of the current solution in terms of the solution space, and try to find a better local solution. A comprehensive survey of the GRASP approach can be found in [41]. In this paper, Abello et al. proposed a incremental algorithm to build a maximal γ-clique, which serves as the initial feasible solution in GRASP. Before we move to the algorithm, we first define the potential of a vertex set R as

$$\phi(R) = |E(R)| - \gamma \left(\begin{array}{c} |R| \\ 2 \end{array} \right)$$

and the potential of R with respect to a disjoint vertices set S to be

$$\phi_S(R) = \phi(S \cup R)$$

Furthermore, considering a graph $G = (V, E)$ and a γ-clique induced by vertices set $S \subset V$, we call a vertex $x \in (V \setminus S)$ a δ-vertex with respect to S if and only if the graph induced by $S \cup \{x\}$ is a γ-clique. Then, the set of γ-vertices with respect to S is denoted as $\mathcal{N}_\gamma(S)$. Given this, the incremental algorithm tries to add a *good* vertex in $\mathcal{N}_\gamma(S)$ into S. To facilitate our discussion, a potential difference of a vertex $y \in \mathcal{N}_\gamma(S) \setminus \{x\}$ is defined to be

$$\delta_{S,x}(y) = \phi_{S\cup\{x\}}(\{y\}) - \phi_S(\{y\})$$

The above equation can also expressed as

$$\delta_{S,x}(y) = deg(x)|_S + deg(y)|_{\{x\}} - \gamma(|S| + 1)$$

where $deg(x)|_S$ is the degree of x in the graph induced by vertex set S. This equation implies that the potential of y which is a γ-neighbor of x does not decrease when x is included in S. Here the γ-neighbors of vertex x are the neighbors of x with $deg(x)|_S$ greater than $\gamma|S|$. The total effect caused by adding vertex x to current γ-clique S is

$$\Delta_{S,x} = \sum_{y \in \mathcal{N}_\gamma(S) \setminus \{x\}} \delta_{S,x}(y) = |\mathcal{N}_\gamma(\{x\})| + |\mathcal{N}_\gamma(S)|(deg(x)|_S - \gamma(|S| + 1))$$

We see that the vertices with a large number of γ-neighbors and high degree with respect to S are preferred to be selected. A greedy algorithm to build a maximal γ-clique is outlined in Algorithm *DiscoverMaximalQuasi-Clique*. The time complexity of this algorithm is $O(|S||V|^2)$, where S the vertex set used to induce a maximal γ-clique.

Algorithm 9 DiscoverMaximalQuasi-clique(V, E, γ)

```
$\gamma^* \leftarrow 1, S^* \leftarrow \emptyset$;
select a vertex $x \in V$ and add into $S^*$;
**while** $\gamma^* \geq \gamma$ **do**
    $S \leftarrow S^*$;
    **if** $\mathcal{N}_{\gamma^*}(S) \neq \emptyset$ **then**
        select $x \in \mathcal{N}_{\gamma^*}(S)$;
    **else**
        **if** $\mathcal{N}(S) \setminus S = \emptyset$ **then**
            **return** $S$;
        **end if**
        select $x \in \mathcal{N}(S) \setminus S$;
    **end if**
    $S^* \leftarrow S \cup \{x\}$;
    $\gamma^* \leftarrow 2|E(S^*)|/(|S^*|(|S^*| - 1))$;
**end while**
**return** $S$;
```

Then applying GRASP, a local search procedure tries to improve the generated maximal γ-clique. Generally speaking, given current γ-clique induced by vertex set S, this procedure attempts to substitute two vertices within S with one vertex outside S in order to improve aforementioned $\Delta_{S,x}$. GRASP guarantees to obtain a local optimum.

Visualization of Dense Components. Wang et al. [52] combine theoretical bounds, a greedy heuristic for graph traversal, and visual cues to develop a mining technique for clique, quasi-clique, and k-core components. Their approach is named CSV for Cohesive Subgraph Visualization. Figure 10.5 shows a representative plot and how it is interpreted.

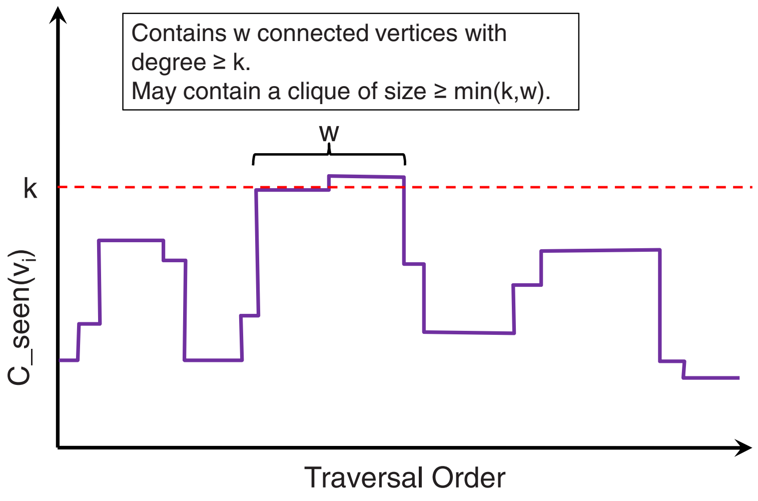

Figure 10.5. Example of CSV Plot

A key measure in CSV is co-cluster size $CC(v, x)$, meaning the (estimated) size of the largest clique containing both vertices v and x. Then, $C(v) = max\{CC(v, x), \forall x \in N(v)\}$.

At the top level of abstraction, the algorithm is not difficult. We maintain a priority queue of vertices observed so far, sorted by $C(v)$ value. We traverse the graph and draw a density plot by iterating the following steps:

1 Remove the top vertex from the queue, making this the current vertex v.
2 Plot v.
3 Add v's neighbors to the priority queue.

Now for some details. If this is the i-th iteration, plot the point $(i, C_{seen}(v_i))$, where $C_{seen}(v_i)$ is the largest value of $C(v_i)$ observed so far. We say "seen so far" because we may not have observed all of v neighbors yet, and even when we have, we are only estimating clique sizes. Next, some neighbors of v may already be in the queue. In this case, update their C values and reprioritize. Due to the estimation method described below, the new estimate is no worse that the previous one.

Since an exact determination of $CC(v,x)$ is computationally expensive, CSV takes several steps to efficiently find a good estimate of the actual clique size. First, to reduce the clique search space, the graph's vertices and edges are pre-processed to map them to a multi-dimensional space. A certain number of vertices are selected as pivot points. Then each vertex is mapped to a vector: $v \rightarrow M(v) = \{d(v,p_1), \cdots, d(v,p_p)\}$, where $d(v,p_i)$ is the shortest distance in the graph from v to pivot p_i. The authors prove that all the vertices of a clique map to the same unit cell, so we can search for cliques by searching individual cells.

Second, CSV further prunes the vertices within each occupied cell. Do the following for each vertex v in each occupied cell: For each neighbor x of v, identify the set of vertices Y which connect to both v and x. Construct the induced subgraph $S(v,x,Y)$. If there is a clique, it must be a subgraph of S. Sort Y by decreasing order of degree in S. To be in a k-clique, a vertex must have degree $\geq k-1$. Consequently, we step through the sorted Y list and eliminate the remainder when the threshold $\delta_S(y_i) < i-1$ is reached. The size of the remaining list is an upper bound estimate for $C(v)$ and $CC(v,x)$. With relatively minor modification, the same general approach can be used for quasi-cliques and k-cores.

The slowest step in CSV is searching the cells for pseudo-cliques, with overall time complexity $O(|V|^2 log|V|2^d)$. This becomes exponential when the graph is a single large clique. However, when tested on two real-life datasets, DBLP co-authorship and SMD stock market networks, $d << |V|$, so performance is polynomial.

Other Heuristic Approaches. We give a brief overview of three additional heuristic approaches. Li et al. [32] studied the problem of discovering dense bipartite subgraphs with so-called balanced noise tolerance, meaning that each vertex in one part is allowed no more than a certain number or a certain percentage of missing edges to the other part. This definition can avoid the density skew found within density-based quasi-cliques. Li et al. observed that their type of maximal quasi-biclique cannot be trivially expanded from traditional maximal bicliques. Some useful properties such as bounded closure and the fixed point property are utilized to develop an efficient algorithm, $\mu - CompleteQB$, for discovering maximal quasi-bicliques with balanced noise tolerance. Given a bipartite graph, the algorithm looks for maximal quasi-bicliques where the number of vertices in each part exceeds a specified value $ms \geq \mu$. Two cases are considered. If $ms \geq 2\mu$, the problem is converted into the problem to find exact maximal μ-quasi bicliques that has been well discussed in [47]. On the other hand, if $ms < 2\mu$, a depth-first search for μ-tolerance maximal quasi-bicliques whose vertex size is between ms and 2μ is conducted to achieve the goal.

A spectral analysis method [13] is used to uncover the functionality of a certain dense component. To begin, the similarity matrix for a protein-protein interaction network is defined, and the corresponding eigenvalues and eigenvectors are calculated. In particular, each eigenvector with positive eigenvalue is identified as a quasi-clique, while each eigenvector with negative eigenvalue is considered a quasi-biclique. Given these dense components, a statistical test based on p-value is applied to measure whether a dense component is enriched with proteins from a particular category more than would be expected by chance. Simply speaking, the statistical test ensures that the existence of each dense component is significant with respect to a specific protein category. If so, that dense component annotated with the corresponding protein functionality.

Kumar et al. [30] focus on enumerating emerging communities which have little or no representation in newsgroups or commercial web directories. They define an (i, j) biclique, where the number of vertices in each part are i and j, respectively, to be the $core$ of interested communities. Therefore, this paper aims to extract a non-overlapping maximal set of $cores$ for interested communities. A stream-based algorithm combining a set of pruning techniques is presented to process huge raw web data and eventually generate the appropriate cores. Some open problems like how to automatically extract semantic information and organize them into a useful structure are also discussed.

3.3 Exact and Approximation Algorithms for Discovering Densest Components

In this section, we focus on the problem of finding the densest components, i.e., the quasi-cliques with the highest values of $gamma$. We first look at exact solutions, utilizing max-flow/min-cut related algorithms. To reach faster performance, we then consider several greedy approximation algorithms that guarantee. These bounded-approximation algorithms are able to efficiently handle the large graphs and obtain guaranteed reasonable results.

Exact Solution for Discovering Densest Subgraph. We first consider density of a graph defined as its average degree. Using this definition, Goldberg [19] showed that the problem of finding the densest subgraph can be exactly reduced to a sequence of max-flow/min-cut problems. Given a value g, algorithm constructs a network and finds a min-cut on it. The resulting sizes tell us whether there is a subgraph with density at least g. Given a graph G

with n vertices and m edges, the construction of its corresponding cut network are as follows:

1 Add two vertices source s and sink t to undirected G;

2 Replace each undirected edge with two directed edges with capacity 1 such that each endpoint is the source and target of those two edges, respectively;

3 Add directed edges with capacity m from s to all vertices in G, and add directed edges with capacity $m + 2g - d_i$ from all vertices in G to t, where d_i is the degree of vertex v_i in the original graph.

We apply the max-flow/min-cut algorithm to decompose the vertices of the new network into two non-overlapping sets S and T, such that $s \in S$ and $t \in T$. Let $V_s = S \setminus \{s\}$. Goldberg proved that there exists a subgraph with density at least g if $V_s \neq \emptyset$. The following theorem formally presents this result:

Theorem 10.2. *Given S and T which are generated by the algorithm for max-flow min-cut problem, if $V_s \neq \emptyset$, then there is no subgraph with density D such that $g \leq D$. If $V_s = \emptyset$, then there exists a subgraph with density D such that $g \geq D$.*

The remaining issue is to enumerate all possible values of density and apply the max-flow/min-cut algorithm for each value. Goldberg observed that the difference between any two subgraphs is no more than $\frac{1}{n(n-1)}$. Combined with binary search, this observation provides a effective stop criteria to reduce the search space. The sketch of the entire algorithm is outlined in Algorithm *FindDensestSubgraph*.

Greedy Approximation Algorithm with Bound. In [14], Charikar describes exact and greedy approximation algorithms to discover subgraphs which can maximize two different notions of density, one for undirected graphs and one for directed graphs. The density notion utilized for undirected graphs is the average degree of the subgraph, such that density $f(S)$ of the subset S is $\frac{|E(S)|}{|S|}$. For directed graphs, the criteria first proposed by Kannan and Vinay [27] is applied. That is, given two subsets of vertices $S \subseteq V$ and $T \subseteq V$, the density of subgraph $H_{S,T}$ is defined as $d(S,T) = \frac{|E(S,T)|}{\sqrt{|S||T|}}$. Here, S and T are not necessarily disjoint. This paper studies the optimization problem of discovering a subgraph H_s induced by a subset S with maximum $f(S)$ or $H_{S,T}$ induced by two subsets S and T with maximum $d(S,T)$, respectively.

The author shows that finding a subgraph H_S in undirected graph with maximum $f(S)$ is equivalent to solving the following linear programming (LP) problem:

Algorithm 10 FindDensestSubgraph(G)

```
$mind \leftarrow 0; maxd \leftarrow m;$
$V_s \leftarrow \emptyset;$
**while** $maxd - mind \geq \frac{1}{n(n-1)}$ **do**
    $g \leftarrow \frac{maxd+mind}{2};$
    Construct new network as we have mentioned;
    Generate $S$ and $T$ utilizing max-flow min-cut algorithm;
    **if** $S = \{s\}$ **then**
        $maxd \leftarrow g;$
    **else**
        $mind \leftarrow g;$
        $V_s \leftarrow S - \{s\};$
    **end if**
**end while**
**return** subgraph induced by $V_s$;
```

(1) $max \sum_{ij} x_{ij}$
(2) $\forall ij \in E \; x_{ij} \leq y_i$
(3) $\forall ij \in E \; x_{ij} \leq y_j$
(4) $\sum_i y_i \leq 1$
(5) $x_{ij}, y_i \geq 0$

From a graph viewpoint, we assign each vertex v_i with weight $\sum_j x_{ij}$, and $min(y_i, y_j)$ is the threshold for the weight of all edges (v_i, v_j) incident to vertex v_i. Then x_{ij} can be considered as the weight of edge (v_i, v_j) which vertex v_i distributes. Weights are normalized so that the sum of threshold for edges incident to vertex v_i, $\sum_i y_i$, is bounded by 1. In this sense, finding the optimal solution of $\sum_{ij} x_{ij}$ is equivalent to finding a set of edges such that the weights of their incident vertices mostly distribute to them. Charikar shows that the optimality of the above LP problem is exactly equivalent to discovering the densest subgraph in a undirected graph.

Intuitively, the complexity of this LP problem depends highly on the number of edges and vertices in the graph (i.e., the number of inequality constraints in LP). It is impractical for large graphs. Therefore, Charikar proposes an efficient greedy algorithm and proves that this algorithm produces a 2-approximation for $f(G)$. This greedy algorithm is a simple variant of [29]. Let S is a subset of V and H_S is its induced subgraph with density $f(H_S)$. Given this, we outline this greedy algorithm as follows:

1 Let S be the subset of vertices, initialized as V;

2 Let H_S be the subgraph induced by vertices S;

3 For each iteration, eliminate the vertex with lowest degree in H_S from S and recompute its density;

4 For each iteration, measure the density of H_S and record it as a candidate for densest component

Similar techniques are also applied to finding the densest subgraph in a directed graph. The greedy algorithm for directed graphs takes $O(m + n)$ time. According to the analysis, Charikar claims that we have to run the greedy algorithm for $O(\frac{\log n}{\epsilon})$ values of c in order to get a $2 + \epsilon$ approximation, where $c = |S|/|T|$ and S, T are two subset of vertices in the graph.

A variant of this approach is presented in [25]. Jin et al. developed an approximation algorithm for discovering the densest subgraph by introducing a new notion of *rank subgraph*. The rank subgraph can be defined as follows:

Definition 10.3. *(Rank Subgraph) [25]. Given an undirected graph $G = (V, E)$ and a positive integer d, we remove all vertices with degree less than d and their incident edges from G. Repeat this procedure until no vertex can be eliminated and form a new graph G_d. Each vertex in G_d is adjacent to at least d vertices in G_d. If G_d has no vertices, it is denoted $G_\emptyset$. Given this, construct a subgraph sequence $G \supseteq G_1 \supseteq G_2 \cdots \supseteq G_l \supset G_{l+1} = G_\emptyset$, where $G_l \neq G_\emptyset$ and contains at least $l + 1$ vertices. Define l as the rank of the graph G, and G_l as the rank subgraph of G.*

Lemma 10.4. *Given an undirected graph G, let G_s be the densest subgraph of G with density $d(G_s)$ and G_l be its rank subgraph with density $d(G_l)$. Then, the density of G_l is no less than half of the density of G_s:*

$$d(G_l) \geq \frac{d(G_s)}{2}$$

The above lemma implies that we can use the rank subgraph G_l with highest rank of G to approximate its densest subgraph. This technique is utilized to derive a efficient search algorithm for finding densest subgraphs from a sequence of bipartite graphs. The interested reader can refer to [25] for details.

Other Approximation Algorithms. Anderson et al. [4] consider the problem of discovering dense subgraphs with lower bound or upper bound of size. Three problems including **dalks**, **damks** and **dks** are formulated. In detail, **dalks** is the abbreviation for Densest-At-Least-K subgraph problem aiming at extracting an induced subgraph with highest average degree among all subgraphs with at least k vertices. Similarly, **damks** looks for the Densest At-Most-K subgraph and **dks** seeks the densest subgraph with exactly k vertices. Clearly, both **dalks** and **damks** are relaxed versions of **dks**. Anderson et al. show that **daks** is approximately as hard as **dks** which has been proven to be NP-Complete. More importantly, an effective 1/3-approximation algorithm based on core decomposition of a graph is proposed for **dalks**. This algorithm runs in $O(m + n)$ and $O(m + n \log n)$ time for unweighted and weighted graphs, respectively.

We describe the algorithm for **dalks** as follows. Given a graph $G = (V, E)$ with n vertices and a lower bound of size k, let H_i be the subgraph induced by i vertices. At the beginning, i is initialized with n and H_i is the original graph G. Then, we remove the vertex v_i with minimum weighted degree from H_i to form H_{i-1}. Next, we update its corresponding total weight $W(H_{i-1})$ and density $d(H_{i-1})$. We repeat this procedure and get a sequence of subgraphs $H_n, H_{n-1}, \cdots, H_1$. Finally, we choose the subgraph H_k with maximal density $d(H_k)$ as the resulting dense component.

Anderson [3] develops a local search algorithm to find a dense bipartite subgraph near a specified starting vertex in a bipartite graph. Specifically, for any bipartite subgraph with K vertices and density θ (the definition of density is identical to the definition in [27]), the proposed algorithm guarantees to generate a subgraph with density $\Omega(\theta/\log\Delta)$ near any starting vertex v where Δ is the maximum degree in the graph. The time complexity of this algorithm is $O(\Delta K^2)$ which is independent of the size of graph, and thus has potential to be scaled for large graphs.

4. Frequent Dense Components

The dense component discovery problem can be extended to consider a dataset consisting of a set of graphs $D = \{G_1, \cdots, G_n\}$. In this case, we have two criteria for components: they must be dense and they must occur frequently. The density requirement can be any of our earlier criteria. The frequency requirement says that a component satisfies a minumum *support* threshold; that is, it appears in at least a certain number of graphs. Obviously, if we say that we find the same component in different graphs, there must be a correspondence of vertices from one graph to another. If the graphs have exactly the same vertex sets, then we call this a relation graph set.

Many authors have considered the broader problem of frequent pattern mining in graphs [50, 23, 31]; however, not until recently has there been a clear focus on patterns defined and restricting by density. Several recent papers have looked into discovery methods for frequent dense subgraphs. We take a more detailed look at some of these papers.

4.1 Frequent Patterns with Density Constraints

One approach is to impose a density constraint on the patterns discovered by frequent pattern mining. In [55], Yan et al. use the minumum cut clustering criterion: a component must have an edge cut less than or equal to k. Note that this is equivalent to a k-core criterion. Furthermore, each frequent pattern must be closed, meaning it does not have any supergraph with the same support level. They develop two approaches, pattern growth and pattern reduction. In pattern growth, begin with a small subgraph (possibly a single vertex) that satisfies both the frequency and density requirements but may not be closed. The algorithm incrementally adds adjacent edges until the pattern is closed. In pattern reduction, initialize the working set P_1 to be the first graph G_1. Update the working set by intersecting its edge set with the edges of the next graph:

$$P_i = P_{i-1} \cap G_I = (V, E(P_{i-1}) \cap E(G_I))$$

This removes any edges that do not appear in both input graphs. Decompose P_i into k-core subgraphs. Recursively call pattern reduction for each dense subgraph. Record the dense subgraphs that survive enough intersections to be considered frequent.

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. The consequence of edge intersection is that we only find components whose edges happen to appear in the first $min\_support$ graphs. Therefore, a useful heuristic would be to order the graphs by decreasing overall density. In [55], they find that pattern reduction works better when targeting high connectivity but a low support threshold. Conversely, pattern growth works better when targeting high support but only modest connectivity.

4.2 Dense Components with Frequency Constraint

Hu et al. [22] take a different perspective, providing a simple meta-algorithm on top of an existing dense component algorithm. From the input graphs, which must be a relation graph set, they derive two new graphs, the Summary Graph and the Second-Order Graph. The Summary Graph is $\hat{G} = (V, \hat{E})$, where an edge exists if it appears in at least k graphs in D. For the Second-Order Graph, we transform each edge in D into a vertex, giving us $F = (V \times V, E_F)$. An edge joins two vertices in F (equivalent to two edges in G) if they have similar support patterns in D. An edge's support pattern is represented as the n-dimensional vector of weights in each graph: $\boldsymbol{w}(e) = \{w_{G_1}(e), \cdots, w_{G_n}(e)\}$. Then, a similarity measure such as Euclidean distance can be used to determine whether two vertices in F should be connected.

Given these two secondary graphs, the problem is quite simple to state: find *coherent dense subgraphs*, where a subgraph S qualifies if its vertices form a dense component in $\hat{G}$ and if its edges form a dense component in F. Density in $\hat{G}$ means that the component's edges occur frequently, when considering the whole relation graph set D. Density in F ensures that these frequent edges are coherent, that is, they tend to appear in the same graphs.

To efficiently find dense subgraphs, Hu uses a modified version of Hartuv and Shamir's HCS mincut algorithm [21]. Because Hu's approach converts any n graphs into only 2 graphs, it scales well with the number of graphs. A drawback, however, is the potentially large size of the second-order graph. The worst case would occur when all n graphs are identical. Since all edge support vectors would be identical, the second order graph would become a clique of size $|E|$ with $O(|E|^2)$ edges.

4.3 Enumerating Cross-Graph Quasi-Cliques

Pei et al. [40] consider the problem of finding so-called cross-graph quasi-cliques, CGQC for short. They use the balanced quasi-clique definition. Given a set of graphs $D = \{G_1, \cdots, G_n\}$ on the same set of vertices U, corresponding parameters $\gamma_1, \cdots, \gamma_n$ for the completeness of vertex connectivity, and a minimum component size min_S, they seek to find all subsets of vertices of cardinality $\geq min_S$ such that when each subset is induced upon graph G_i, it will form a maximal γ_i-quasi-clique.

A complete enumeration is $\#P$-Complete. Therefore, they derive several graph-theoretical pruning methods that will typically reduce the execution time. They employ a *set enumeration tree* [43] to list all possible subsets of

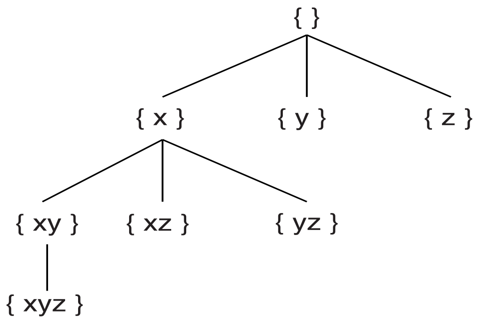

Figure 10.6. The Set Enumeration Tree for {x,y,z}

vertices, while taking advantage of some tree-based concepts, such as depth-first search and sub-tree pruning. An example of a set enumeration tree is shown in Figure 10.6. Below is a brief listing of some of the graph and tree properties they utilize to prune the set of candidate components, followed by the main algorithm, called *Crochet*.

1 Given γ and graph size n, there exist upper bounds on the graph diameter $diam(G)$. For example, $diam(G) \leq n-1$ if $\gamma > \frac{1}{n-1}$.

2 Define $N^k(u) =$ vertices within a distance k of u.

3 Reducing vertices: If $\delta(u) < \gamma_i(min_S - 1)$ or $|N^k(u)| < (min_S - 1)$, 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 $N^{k_i}_{G_i}$.

5 Subtree pruning: apply various rules on min_S, redundancy, monotonicity.

5. Applications of Dense Component Analysis

In financial and economic analysis, dense components represent entities that are highly correlated. For example, Boginski et al. define a market graph, where each vertex is a financial instrument, and two vertices are connected if their behaviors (say, price change over time) are highly correlated [9, 10]. A dense component then indicates a set of instruments whose members are well-correlated to one another. This information is valuable both for understanding market dynamics and for predicting the behavior of individual instruments. Density can also indicate strength and robustness. Du et al. [15] identify cliques in a financial grid space to assist in discovering price-value motifs. Some researchers have employed bipartite and multipartite networks. Sim et al. [47] correlates stocks to financial ratios using quasi-bicliques. Alkemade

Algorithm 11 Crochet$(G_1, G_2, \gamma_1, \gamma_2, min_s)$

```
 1: for all graph $G_i$ do
 2:     construct set enumeration tree for all possible vertex subsets of $G_i$;
 3:     $k_i \leftarrow$ upper bound diameter of complete $\gamma_i$-quasi-complete graph in $G_i$;
 4: end for
 5: apply Vertex and Edge Reduction to $G_1$ and $G_2$;
 6: for all $v \in V(G_1)$, using DFS and highest-degree-child-first order do
 7:     recursive-mine $(\{v\}, G_1, G_2)$;
 8: end for
 9:
10: Function recursive-mine$(X, G_1, G_2)$; {returns TRUE if still seeking
    quasi-cliques in this branch}
11: $G_i \leftarrow G_i(P), P = \{u|u \in \cap_{v \in X, i=1,2} N^{k_i}_{G_i}(v)\}$ {Candidate Projection}
12: $G_i \leftarrow G_i(P(X))$;
13: apply Vertex Reduction;
14: if a Subtree Pruning condition applies then return FALSE;
15: $continue \leftarrow$ FALSE;
16: for all $v \in P(X) \backslash X$, using DFS and highest-degree-child-first order do
17:     $continue \leftarrow continue \vee$ recursive-mine $(X \cup \{v\}, G_1, G_2)$;
18: end for
19: if (not $continue$) $\wedge$ ($G_i(X)$ is a $\gamma_i$-quasi-complete graph) then
20:     output $X$;
21:     return TRUE;
22: else
23:     return $continue$;
24: end if
```

et al. [2] finds edge density in a tripartite graph of producers, consumers, and intermediaries to be an important factor in the dynamics of commerce.

In the first decade of the 21st century, the field that perhaps has shown the greatest interest and benefitted the most from dense component analysis is biology. Molecular and systems biologists have formulated many types of networks: signal transduction and gene regulation networks, protein interaction networks, metabolic networks, phylogenetic networks, and ecological networks. [26].

Proteins are so numerous that even simple organisms such as *Saccharomyces cerevisiae*, a budding yeast, are believed to have over 6000 [51]. Understanding the function and interrelationships of each one is a daunting task. Fortunately, there is some organization among the proteins. Dense components in protein-protein interaction networks have been shown to correlate to functional units [49, 42, 54, 13, 6]. Finding these modules and complexes helps to explain metabolic processes and to annotate proteins whose functions are as yet unknown.

Gene expression faces similar challenges. Microarray experiments can record which of the thousands of genes in a genome are expressed under a set of test conditions and over time. By compiling the expression results from several trials and experiments, a network can be constructed. Clustering the genes into dense groups can be used to identify not only healthy functional classes, but also the expression pattern for genetic diseases [48].

Proteins interact with genes by activating and regulating gene transcription and translation. Density in a protein-gene bipartite graph suggests which protein groups or complexes operate on which genes. Everett et al. [16] have extended this to a tripartite protein-gene-tissue graph.

Other biological systems are also being modeled as networks. Ecological networks, famous for food chains and food webs, are receiving new attention as more data becomes available for analysis and as the effects of climate change become more apparent.

Today, the natural sciences, the social sciences, and technological fields are all using network and graph analysis methods to better understand complex systems. Dense component discovery and analysis is one important aspect of network analysis. Therefore, readers from many different backgrounds will benefit from understanding more about the characteristics of dense components and some of the methods used to uncover them.

6. Conclusions and Future Research

In this chapter, we presented a survey of algorithms for dense subgraph discovery. This problem has been studied in the classical literature in the context of the problem of graph partitioning. Subsequently, a number of techniques have been designed for quasi-clique detection, as well as shingling approaches for dense subgraph discovery. Many of the recent applications are designed in the contexts of the web, social, communication and biological networks. These networks have a number of properties, in that they are *massive* and often *dynamic* in nature. This leads to a number of interesting problems for future research:

- In many large scale applications, the data is often disk-resident. This leads to issues involving efficient processing of the underlying network. This is because it is not possible to perform random access of the edges in a disk-resident networks.

- In applications such as the web and social networks, the *domain* of the underlying graph may be massive. In many web, telecommunication, biological and social networks, we may have millions of nodes in the underlying graph. Consequently, the number of edges may range in the trillions. This may lead to storage issues, since the number of distinct edges may not even be possible to store effectively on many desktop machines.

- A number of recent applications may lead to the streaming scenario in which the edges in the graph are received incrementally over time at a fast speed. This is the case in many large telecommunication and social networks. In such cases, it may be extremely challenging to analyze the underlying graph in real time to determine dense patterns.

The area of dense graph mining in massive graphs is still relatively unexplored and represents a fertile area of future research for a number of different applications.

References

[1] J. Abello, M. G. C. Resende, and S. Sudarsky. Massive quasi-clique detection. In *LATIN '02: Proc. 5th Latin American Symposium on Theoretical Informatics*, pages 598–612. Springer-Verlag, 2002.

[2] F. Alkemade, H. A. La Poutré, and H. A. Amman. An agent-based evolutionary trade network simulation. In A. Nagurney, editor, *Innovations in Financial and Economic Networks (New Dimensions in Networks)*, chapter 11, pages 237–255. Edward Elgar Publishing, 2004.

[3] R. Andersen. A local algorithm for finding dense subgraphs. In *SODA '08: Proc. 19th ACM-SIAM Symp. on Discrete Algorithms*, pages 1003–1009. Society for Industrial and Applied Mathematics, 2008.

[4] R. Andersen and K. Chellapilla. Finding dense subgraphs with size bounds. In *WAW '09: Proc. 6th Intl. Workshop on Algorithms and Models for the Web-Graph*, pages 25–37. Springer-Verlag, 2009.

[5] Anna Nagurney, ed. *Innovations in Financial and Economic Networks (New Dimensions in Networks).* Edward Elgar Publishing, 2004.

[6] G. Bader and C. Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4(1):2, 2003.

[7] V. Batagelj and M. Zaversnik. An o(m) algorithm for cores decomposition of networks. *CoRR (Computing Research Repository)*, cs.DS/0310049, 2003.

[8] P. Berkhin. Survey of clustering data mining techniques. In C. N. Jacob Kogan and M. Teboulle, editors, *Grouping Multidimensional Data*, chapter 2, pages 25–71. Springer Berlin Heidelberg, 2006.

[9] V. Boginski, S. Butenko, and P. M. Pardalos. On structural properties of the market graph. In A. Nagurney, editor, *Innovations in Financial and Economic Networks (New Dimensions in Networks)*, chapter 2, pages 29–45. Edward Elgar Publishing, 2004.

[10] V. Boginski, S. Butenko, and P. M. Pardalos. Mining market data: A network approach. *Computers and Operations Research*, 33(11):3171–3184, 2006.

[11] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Comput. Netw. ISDN Syst.*, 29(8-13):1157–1166, 1997.

[12] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM*, 16(9):575–577, 1973.

[13] D. Bu, Y. Zhao, L. Cai, H. Xue, and X. Z. andH. Lu. Topological structure analysis of the protein-protein interaction network in budding yeast. *Nucl. Acids Res.*, 31(9):2443–2450, 2003.

[14] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *APPROX '00: Proc. 3rd Intl. Workshop on Approximation Algoritms for Combinatorial Optimization*, volume 1913, pages 84–95. Springer, 2000.

[15] X. Du, J. H. Thornton, R. Jin, L. Ding, and V. E. Lee. Migration motif: A spatial-temporal pattern mining approach for financial markets. In *KDD '09: Proc. 15th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*. ACM, 2009.

[16] L. Everett, L.-S. Wang, and S. Hannenhalli. Dense subgraph computation via stochastic search: application to detect transcriptional modules. *Bioinformatics*, 22(14), July 2006.

[17] G. W. Flake, S. Lawrence, and C. L. Giles. Efficient identification of web communities. In *KDD'00: Proc. 6th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 150 – 160, 2000.

[18] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *VLDB '05: Proc. 31st Intl. Conf. on Very Large Data Bases*, pages 721–732. ACM, 2005.

[19] A. V. Goldberg. Finding a maximum density subgraph. Technical report, UC Berkeley, 1984.

[20] G. Grimmett. *Precolation*. Springer Verlag, 2nd edition, 1999.

[21] E. Hartuv and R. Shamir. A clustering algorithm based on graph connectivity. *Inf. Process. Lett.*, 76(4-6):175–181, 2000.

[22] H. Hu, X. Yan, Y. H. 0003, J. Han, and X. J. Zhou. Mining coherent dense subgraphs across massive biological networks for functional discovery. In *ISMB (Supplement of Bioinformatics)*, pages 213–221, 2005.

[23] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *PKDD '00: Proc. 4th European Conf. on Principles of Data Mining and Knowledge Discovery*, pages 13–23, 2000.

[24] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31(3):264–323, 1999.

[25] R. Jin, Y. Xiang, N. Ruan, and D. Fuhry. 3-hop: A high-compression indexing scheme for reachability query. In *SIGMOD '09: Proc. ACM SIGMOD Intl. Conf. on Management of Data*. ACM, 2009.

[26] B. H. Junker and F. Schreiber. *Analysis of Biological Networks*. Wiley-Interscience, 2008.

[27] R. Kannan and V. Vinay. Analyzing the structure of large graphs. manuscript, August 1999.

[28] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum, New York, 1972.

[29] G. Kortsarz and D. Peleg. Generating sparse 2-spanners. *J. Algorithms*, 17(2):222–236, 1994.

[30] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the web for emerging cyber-communities. *Computer Networks*, 31(11-16):1481–1493, 1999.

[31] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *ICDM '01: Proc. IEEE Intl. Conf. on Data Mining*, pages 313–320. IEEE Computer Society, 2001.

[32] J. Li, K. Sim, G. Liu, and L. Wong. Maximal quasi-bicliques with balanced noise tolerance: Concepts and co-clustering applications. In *SDM '08: Proc. SIAM Intl. Conf. on Data Mining*, pages 72–83. SIAM, 2008.

[33] G. Liu and L. Wong. Effective pruning techniques for mining quasi-cliques. In W. Daelemans, B. Goethals, and K. Morik, editors, *ECML/PKDD (2)*, volume 5212 of *Lecture Notes in Computer Science*, pages 33–49. Springer, 2008.

[34] R. Luce. Connectivity and generalized cliques in sociometric group structure. *Psychometrika*, 15(2):169–190, 1950.

[35] K. Makino and T. Uno. New algorithms for enumerating all maximal cliques. *Algorithm Theory - SWAT 2004*, pages 260–272, 2004.

[36] H. Matsuda, T. Ishihara, and A. Hashimoto. Classifying molecular sequences using a linkage graph with their pairwise similarities. *Theor. Comput. Sci.*, 210(2):305–325, 1999.

[37] R. Mokken. Cliques, clubs and clans. *Quality and Quantity*, 13(2):161–173, 1979.

[38] J. W. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3:23–28, 1965.

[39] M. E. J. Newman. The structure and function of complex networks. *SIAM REVIEW*, 45:167–256, 2003.

[40] J. Pei, D. Jiang, and A. Zhang. On mining cross-graph quasi-cliques. In *KDD'05: Proc. 11th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 228–238. ACM, 2005.

[41] L. Pitsoulis and M. Resende. Greedy randomized adaptive search procedures. In P. Pardalos and M. Resende, editors, *Handbook of Applied Optimization*, pages 168–181. Oxford University Press, 2002.

[42] N. Pržulj, D. Wigle, and I. Jurisica. Functional topology in a network of protein interactions. *Bioinformatics*, 20(3):340–348, 2004.

[43] R. Rymon. Search through systematic set enumeration. In *Proc. Third Intl. Conf. on Knowledge Representation and Reasoning*, 1992.

[44] J. P. Scott. *Social Network Analysis: A Handbook.* Sage Publications Ltd., 2nd edition, 2000.

[45] S. B. Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269–287, 1983.

[46] S. B. Seidman and B. Foster. A graph theoretic generalization of the clique concept. *J. Math. Soc.*, 6(1):139–154, 1978.

[47] K. Sim, J. Li, V. Gopalkrishnan, and G. Liu. Mining maximal quasi-bicliques to co-cluster stocks and financial ratios for value investment. In *ICDM '06: Proc. 6th Intl. Conf. on Data Mining*, pages 1059–1063. IEEE Computer Society, 2006.

[48] D. K. Slonim. From patterns to pathways: gene expression data analysis comes of age. *Nature Genetics*, 32:502–508, 2002.

[49] V. Spirin and L. Mirny. Protein complexes and functional modules in molecular networks. *Proc. Natl. Academy of Sci.*, 100(21):1123–1128, 2003.

[50] Y. Takahashi, Y. Sato, H. Suzuki, and S.-i. Sasaki. Recognition of largest common structural fragment among a variety of chemical structures. *Analytical Sciences*, 3(1):23–28, 1987.

[51] P. Uetz, L. Giot, G. Cagney, T. A. Mansfield, R. S. Judson, J. R. Knight, D. Lockshon, V. Narayan, M. Srinivasan, P. Pochart, A. Qureshi-Emili, Y. Li, B. Godwin, D. Conover, T. Kalbfleisch, G. Vijayadamodar, M. Yang, M. Johnston, S. Fields, and J. M. Rothberg. A comprehensive analysis of protein-protein interactions in saccharomyces cerevisiae. *Nature*, 403:623–631, 2000.

[52] N. Wang, S. Parthasarathy, K.-L. Tan, and A. K. H. Tung. Csv: visualizing and mining cohesive subgraphs. In *SIGMOD '08: Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pages 445–458. ACM, 2008.

[53] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications.* Cambridge University Press, 1994.

[54] S. Wuchty and E. Almaas. Peeling the yeast interaction network. *Proteomics*, 5(2):444–449, 2205.

[55] X. Yan, X. J. Zhou, and J. Han. Mining closed relational graphs with connectivity constraints. In *KDD '05: Proc. 11th ACM SIGKDD Intl. Conf. on Knowledge Discovery in Data Mining*, pages 324–333. ACM, 2005.