On Triangulation-based Dense Neighborhood Graph Discovery

Nan Wang, Jingbo Zhang, Kian-Lee Tan, Anthony K. H. Tung *
School of Computing, National University of Singapore, Singapore
{wangnan, jingbo, tankl, atung}@comp.nus.edu.sg

ABSTRACT

This paper introduces a new definition of dense subgraph pattern, the DN-graph. DN-graph considers both the size of the substructure and the minimum level of interactions between any pair of the vertices.

The mining of DN-graphs inherits the difficulty of finding clique, the fully-connected subgraphs. We thus opt for approximately locating the DN-graphs using the state-of-the-art graph triangulation methods. Our solution consists of a family of algorithms, each of which targets a different problem setting. These algorithms are iterative, and utilize repeated scans through the triangles in the graph to approximately locate the DN-graphs. Each scan on the graph triangles improves the results. Since the triangles are not physically materialized, the algorithms have small memory footprint.

With our solution, the users can adopt a "pay as you go" approach. They have the flexibility to terminate the mining process once they are satisfied with the quality of the results. As a result, our algorithms can cope with semi-streaming environment where the graph edges cannot fit into main memory. Results of extensive performance study confirmed our claims.

1. INTRODUCTION

Graphs are the most pervasive model of entity interactions as it concisely captures the interactions among entities. However, for large graphs (which are becoming increasingly common in many applications), it becomes too complicated for human beings to find key information without the help of suitable graph mining technology. Graph mining refers to the process of discovering designated subgraphs from a target graph, in the hope of uncovering unknown knowledge about the graph. When facing unsolvable resource constraint, how to answer the mining question to the best, becomes more challenging.

Most recent works on graph mining [4, 2, 3, 6, 10] believe that dense patterns are prominent. They capture the most active involvement of entity interactions. Subsequently, researchers propose various definitions of dense substructures. In section 2, we review some of these patterns.

Intuitively, a dense pattern contains a set of highly relevant vertices. They usually share large number of common neighbors (two vertices are neighbors if they connect to each other by an edge). The definition of DN-graph (a.k.a. **Dense Neighborhood graph**) follows this intuition.

This paper provides a set of algorithms to mine DN-graphs from large scaled graphs. The problem of mining DN-graphs is an NP-complete problem (due to the close relationship between DN-graphs and cliques, lemma 3.1 will cover this in detail). As such, in this paper, we opt to design approximate solutions. In our solutions, the local neighborhood size is the most important, but difficult, quantity to be computed. We associate this quantity with local triangle counting in order to approximate it efficiently.

Graph triangulation refers to the process of generating all triangles in a graph. Our approach locates DN-graph by using the state-of-the-art triangulation algorithm [14, 5]. As the storage of triangles can be expensive, we do not store these triangles. Instead, we design our approach to operate iteratively. In each iteration, our scheme dynamically regenerates all triangles and improve the connectivity estimation between vertices in each round.

Such an iterative, triangulation-based approach has three advantages. Firstly, most of the details involved in efficient processing, such as minimizing I/Os, are abstracted within the triangulation algorithm. The abstraction ensures our approach's extensibility to different input settings, e.g. when the target graph is too large to fit into memory, our approach only needs to change the access method of the graph links. In addition, the estimation of the local neighborhood is encapsulated within the triangulation algorithm. Secondly, as the estimation of the local density value improves with each additional iteration, users can adopt a "pay as you go" approach and obtain the most updated results on demand. Finally, when the graph is too large to fit into the main memory, we can collect statistics in the first iteration to support effective buffer management, should there be a need to store the local density value on a disk, since the triangles are generated in the same ordering in every iteration

In this paper, we present triangulation based dense graph mining algorithms. Together they form an algorithm family. Their key features are compared in Table 1. For brevity, we name them respectively as 1) TriDN, 2) BiTriDN and 3) StreamDN.

Algorithms TriDN and BiTriDN are two variances that handle in-memory graphs. Both algorithms iteratively generate triangles to refine the λ value. These two processes reach convergence when all λ values remain the same as previous iteration.

The third algorithm, StreamDN, is for semi-streaming graph setting. In section 4.2, we introduce the model of semi-streaming

*Support in part by NUS FRC Grant R-252-000-370-112 as part of the research project entitled "Visual Exploration and Mining of Cohesive Subgraphs in Complex Relational Graph"(http://nusdm.comp.nus.edu.sg/).

Proceedings of the VLDB Endowment, Vol. 4, No. 2

	In Memory	Time	Space
TriDN	Yes	$O(klog\|V\|\|E\|^{\frac{3}{2}})$	$O(\|V\|log\|V\| +\|E\|)$
BiTriDN (Binary Bounding)	Yes	$O(klog\|V\|\|E\|^{\frac{3}{2}})$	$O(\|V\|log\|V\| +\|E\|)$
StreamDN (Semi-Stream)	No	$O(k\|E\|)$	$O(\|V\|)$

Table 1: A Family of DN-Graph Mining Algorithms

graph. To mine semi-streaming graphs, algorithm StreamDN applies the min-wise independent set property, which provides an approximation for triangulation using sequentially scan of graph edges, with bounded error.

The rest of the paper is organized as follows: In Section 2, we review related works. The definition of DN-graph is formally presented in Section 3. We present algorithms for finding DN graphs in Section 4. Experimental studies are then described in Section 5. Section 6 concludes our work.

2. RELATED WORK

A dense graph pattern is a connected subgraph that has significant more internal connections with respect to the surrounding vertices. Depending on the semantic meanings of the graph data, various forms of dense patterns have been investigated in the literature.

(1) Clique/Quasi-Clique. A clique represents the highest level of internal interactions. In graph theory, a clique is a fully connected subgraph. Each pair of vertices are connected by an edge. A quasi-clique, on the other hand, is an "almost" clique with a few missing edges. If a clique is not a proper subgraph of any larger clique, we call it a "closed" clique. **(2) High Degree Patterns.** This pattern requires the average vertex degree to be above certain level or outstanding among surrounding vertices. Here a vertex's degree is the number of edges intercepting the vertex. Unlike cliques, a high degree pattern do not require high interconnection within the pattern [8]. **(3) Dense Bipartite Patterns.** If the involved entities belong to two distinctive classes, and only entities from different classes have associations, the dense bipartite patterns are bipartite graphs with outstandingly many edges. **(4) Heavy Patterns.** Previous patterns emphasize on the topological features. The heavy patterns, however, aim at maximizing edge weights [9]. If the weights on the edges of a weighted graph follow the triangle inequality, the heavy pattern is also a dense pattern in the un-weighted graph. Even though this type of pattern is not our preliminary target for this paper, it is presented here for completeness purpose.

In this paper, we view a **dense subgraph** as a set of vertices sharing many common neighbors. If two connected vertices share one common neighbor, they form a triangle together with their common neighbor. In view of the association between dense patterns and triangles, we further study the problem of triangle counting.

Triangle counting and listing have been well studied in the literature. Given a graph G with $|V|$ vertices and $|E|$ edges, [14] proposed a triangle-listing algorithm with time complexity $O(|E|^{\frac{3}{2}})$ and with O($3|E|+3|V|$) space. Further work [13] improves the performance of the algorithm by separating the vertices into two types, dense and sparse. The improved technique has the same time complexity as the work in [14], while it reduces the space complexity to $O(|E| + |V|)$. The above ideas count triangles by scanning graph edges, and join adjacency list of the two vertices. The scanning of graphs makes these techniques highly adaptable to streaming environment (In section 4.2, we discuss the graph streaming model.).

Other research works on mining dense subgraphs can be classified according to their counterparts in item-set mining approaches. The most relevant work is the density based solution [15]. This work provides not only a way to find the closed cliques (biggest clique among the neighborhood) but to order all graph vertices into a linear fashion for visualization purpose. One of the leading approaches in [8] adopts two-level-shingling method. Although the work only demonstrates its power in collecting statistics from extremely large graph, its performance is impressive and this approach can be employed into graph mining domain to handle large scale graphs.

3. DN-GRAPH AS A DENSITY INDICATOR

A graph $G(V, E)$ consists of a set of vertices V and set of interactions E over $V \times V$. The size of G, denoted as $|V|$, is the number of vertices in V. The neighborhood of a graph vertex v, is the set of vertices directly connecting to v. We use $N(v)$ to represent it. If vertex u and v share some common neighbors, we use $N_\cap(u, v)$ to represent the joint neighborhood. The neighborhood of e is the joint neighborhood of its two end vertices. We denote the joint neighborhood as N_e. For a subgraph G' of G, the neighborhood of G', $N(G')$, is the set of vertices $u \in G \setminus G'$, which immediately connect with vertices in G'. Inside a graph, the measurement of minimal joint neighborhood size between any connected vertex pair is denoted as λ. We use the notation $\lambda(G)$/$\lambda(V)$ to refer to the measurement of a graph G with vertex set V. For brevity, we omit the content inside bracket and use λ when the context is clear. We also use $\tilde{a}$ to represent an upperbound of quantity a. The upperbound of λ is thus written as $\tilde{\lambda}$.

In this paper, a clique is a fully connected graph, in which every pair of vertices are connected by an edge. If the size of a clique is c, we call the clique a c-clique. When compared with clique of the same size, a quasi-clique has only a fraction (say δ) of edges in the graph, it is a δ quasi-clique.Conventionally δ is in the interval $(0.5, 1]$.

DEFINITION 1. *DN-Graph*
A DN-graph with parameter λ, denoted $G'(V', E', \lambda)$, is a connected subgraph $G'(V', E')$ of graph $G(V, E)$ that satisfies the following conditions: (1) Every connected pair of vertices in G' share at least λ common neighbors.
(2) for any $v \in V \setminus V'$, $\lambda(V' \cup \{v\}) < \lambda$; and for any $v \in V'$, $\lambda(V' - \{v\}) \leq \lambda$.

As the definition states, a DN-graph should be a connected subgraph in which the lower bound of shared neighborhood between any connected vertices, λ, is locally maximized. Being a DN-graph, it has local maximal λ value and the size of the DN-graph is maximized. This ensures that the DN-graph has more distinguishing power and maximal coverage. Similar with the graph's diameter and minimum cut, λ is an indicator of the graphs' underlying density. As proven in the appendix (Proposition 8.3), it is a local maximum graph. For example, in figure 1, subgraph $ABCDEF$ is a DN-graph of λ value 3. If we include one more vertex A', the λ value of the graph $A'ABCDEF$'s drops significantly to 0. Similarly, taking away any vertex, say A, leads to a lower value λ.

DN-graph is designed to represent dense patterns, as it captures subgraphs with more internal associations. It is thus not surprising to see the correlation between DN-graph's λ value and maximum clique size, which is another popular dense indicator.

Figure 2 illustrates the relationship between λ and the size of the maximum clique within a dynamic graph. The graph consists

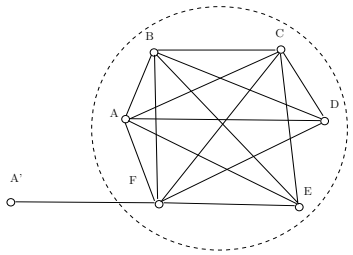


Figure 1: A DN-Graph

of 20 separated vertices and initially has no edges. The topology of the dynamic graph is varied by adding edge to the graph one at a time. For comparison purpose, the vertices are deliberately separated into two groups. Each group consists of 10 vertices. One group ("dense") has much higher probability of being connected to each other with a new edge. The probability of new edge appearing between the vertices of the other group is significantly lower. As the edges are added, the whole graph becomes a 20-clique. Unsurprisingly, the "dense" group becomes a 10-clique much faster compared with the "sparse" subgraph. Correspondingly, the λ value of the "dense" group grows substantially faster than the "sparse" one. From this example, we can see that the growth in λ is a good indication that a subgraph will eventually form a dense clique.

Besides the level of connectivity, a DN-graph also imposes restrictions on the minimal size of the shared neighborhood. This restriction is especially useful when predicting protein complexes via densely connected proteins within a protein-protein interaction (PPI) graph. A protein complex's formation often serves to activate or inhibit one or more of the complex members[11]. In a PPI network, we can observe the phenomenon that members of a protein complex share (significantly many) neighbors. The DN-graph definition reconciles the sharing of neighborhood.

Based on DN-graph, this paper provides effective solutions towards mining DN-graphs within a massive graph, Formally:

DEFINITION 2. *DN-graph mining problem*
Given a graph $G(V, E)$, we want to find all DN-graphs $g(v, e, \lambda)$ in G.

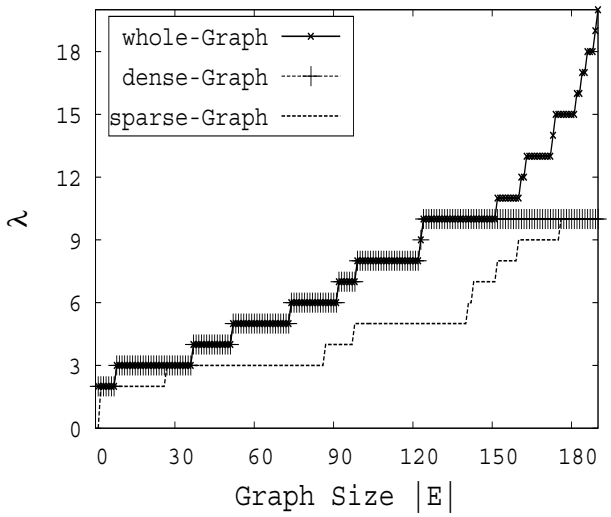


Figure 2: The Growth in λ Value of a 20-Vertex Dynamic Graph

Generally speaking, the level of interactions among entities determines the density of the substructures. From this point of view, it is not surprising to see that some patterns are transformable to others. For example, a DN-graph is a more general case of a closed clique (Recall that a clique is a fully connected graph, while the closed clique is the local maximal clique). In fact, a DN-graph is a relaxation of a clique, with less rigid size constraints. Lemma 3.1 states the relationship formally:

LEMMA 3.1. *DN-Graph and Closed Clique*
A graph contains a closed clique of size d if and only if the graph contains a DN-graph g with $\lambda = d - 2$ and $|g| = d$.

The proof of above lemma is explained in detail in appendix 8.2. Here we omit it for brevity. Using Lemma 3.1, we are able to reduce the close clique mining problem to DN-graph mining problem. The reduction signifies that DN-graph mining is NP-complete (detail please refer to appendix 8.4). Prompted by this result, we seek to develop heuristical solutions instead.

Like the closed clique mining problem, the computational bottleneck for DN-graph mining is on counting degrees within a subgraph. In fact, the counting of local degrees relies heavily on the multiple joins of neighbors, which are computationally expensive. To avoid the complexity of multiple joins, we next introduce the concept of $\lambda(e)$.

3.1 DN-Graph and $\lambda(e)$

As discussed previously, the bottleneck of DN-graph mining is excessive number of neighborhood joins required. This is because we have to test combinatorial number of subgraphs for their λ value and most subgraphs tested are not DN-graphs.

Most of λ value testings however are unnecessary. Due to the local maximality feature of a DN-graph, it is impossible for any two different DN-graphs to share any common vertices or edges. Once we verify that a graph, g_{dn} is a DN-graph, we need not consider other subgraphs that intercept with g_{dn}. In fact, by computing the λ value of edges, we can locate DN-graphs. If we assign the λ value of g_{dn} as the density value of its edges, a DN-graph becomes a set of edges with local maximal λ.

Before explaining the process of locating DN-graph using edge density, let us first define edge density, $\lambda(e)$, formally:

DEFINITION 3. *$\lambda(e)$*
Given a graph $G(V, E)$ and an edge $e \in E$, $\lambda(e)$ is the maximal $\lambda(G')$ value where $e \in E(G')$ and $G' \subseteq (G)$.

The value $\lambda(e)$ indicates quantitatively, the most prominent relationships between two linked vertices. With the definition of local density, we next prove that using $\lambda(e)$, we are able to find all DN-graphs.

THEOREM 3.1. *Locating DN-Graph Using $\lambda(e)$*
A graph G' is a DN-graph if and only if

- *all edges e within G' have equal $\lambda(e)$ value, represented as λ_{max} and,*
- *for all $u \in N(G')$ and $v \in G'$, $\lambda(u, v) \leq \lambda_{max}$.*

For a proof of Theorem 3.1, readers are referred to Appendix 8.3. Based on Theorem 3.1,we can locate the DN-graph by connecting edges with local maximal $\lambda(e)$.

Computing $\lambda(e)$ for all edges is however computationally prohibitive, as discussed in section 3. To facilitate approximation efficiently, we first find an upper bound value for $\lambda(e)$, the $\tilde{\lambda}(e)$, and

then iteratively refine $\tilde{\lambda}(e)$ to capture the actual $\lambda(e)$ as accurately as possible.

The approximation is based on the fact that for an edge e, its $\lambda(e)$ value is upper bounded by the joint neighborhood size of the end vertices of e. This joint neighborhood size is in fact the number of triangles e participates in a graph. Thus we are inspired to use triangulation to approximate $\lambda(e)$ for every graph edge.

4. LOCAL TRIANGULATION AND ITS APPLICATION IN DN-GRAPH MINING

A triangle consists of a vertex triple (u, v, w) and three edges (u, v), (v, w) and (u, w). The problem of counting or listing all triangles within a graph is referred as **Graph Triangulation** in this paper:

DEFINITION 4. *Graph Triangulation*
Given a graph $G(V, E)$, Graph Triangulation finds all vertex triples (u, v, w), where every vertex pair inside the triple are connected by an edge, denoted as $e(u, v)$, $e(v, w)$ and $e(u, w)$ respectively.

The joint neighborhood of an edge $e(u, v)$ upper-bounds $\lambda(e)$, while the number of triangles $e(u, v)$ participates in is equal to the joint neighborhood size. This indicates that graph triangulation provides an upper bound $\lambda(e)$ for every edge e. Here we use $\tilde{\lambda}(u, v)$ to represent the current upper bound of edge (u, v). What's more, given a graph triangle, the $\tilde{\lambda}(u, v)$ can tighten the other two edges' density upper bound. The following proposition gives the relationship between an edge e's ($\tilde{\lambda}(e)$) and its neighbors':

PROPOSITION 4.1. *Neighbor Bounding of $\tilde{\lambda}(e)$*
Inside a triangle (u, v, w), if $\tilde{\lambda}(u, v) \leq min(\tilde{\lambda}(u, w), \tilde{\lambda}(v, w))$, we say w supports $\tilde{\lambda}(u, v)$. $\tilde{\lambda}(u, v)$ is valid if and only if $|\{w|w$ supports $\tilde{\lambda}(u, v)\}| \geq \tilde{\lambda}(u, v)$

PROOF. The proof of necessary condition for proposition 4.1 follows the definition of the $\tilde{\lambda}$ value and is omitted for brevity. Now we prove the sufficient condition: If the number of supporting vertices is greater or equal to $\tilde{\lambda}(e)$, then $\tilde{\lambda}(e)$ is an upper bound for $\lambda(e)$. We prove this by contradiction. Suppose there are fewer than $\tilde{\lambda}(e)$ supporting vertices for $\tilde{\lambda}(e)$, according to the definition of $\lambda(e)$, $\lambda(e) < \tilde{\lambda}(e)$, which means $\tilde{\lambda}(e)$ is larger than $\lambda(e)$. In that case, $\tilde{\lambda}(e)$ is not a valid upper bound of $\lambda(e)$ This contradicts with earlier assumption. With the above reasoning, we complete the proof of proposition 4.1. □

4.1 Triangulation Based DN-Graph Mining

The elementary operation behind local triangulation is the joining of vertex neighborhoods. As studied in [13], the performance of a local triangulation algorithm heavily depends on the order of those join operations. In fact, it is a necessary preprocessing step to sort vertices according to their degrees for effective triangulation (Appendix 8.5 will explains the algorithm in detail).

4.1.1 Generate Triangles to Refine Local Density

We adopt the graph triangulation algorithm in [13]. The algorithm generates triangles systematically for each edge of the graph. The generation of the triangles is a sequence of join operations between the neighbors of two connected vertices. Based on a special order of joining operations, the triangles are generated in a streaming fashion. The DN-graph mining algorithm thus obtains the local density information gradually along the triangle streams. Based on proposition 4.1, we can use the number of triangles an edge participates in ($TC(e)$) as the initial upper bound of the $\lambda(e)$, the $\tilde{\lambda}(e)$. To give an even more accurate bound for $\lambda(e)$, the algorithm uses the density value of e's neighbors' to validate the current upper bound $\lambda(e)$. Figure 3 shows how this process works graphically.

In the first round of graph triangulation, we are aware of the triangular count of $e(a, b)$ (which is in fact $\tilde{\lambda}(e)$), and nothing about its neighbors. However, the triangular counts of the neighbors (a.k.a local density estimation) are available once the first round of graph triangulation is completed. To compute a more accurate $\tilde{\lambda}(e)$ for each edge, we will simply go through more rounds of triangulation and make use of the density information of the neighbors to further validate a new estimation of $\tilde{\lambda}(e)$ for each edge.

For a triangle $(a, b, n1)$, the algorithm checks whether the triangles $(a, b, n1)$ can possibly be a supporting evidence that edge $e(a, b)$ is in a DN-graph, with $\tilde{\lambda}(e)$. This is done by checking whether both the other two edges of triangle $(a, b, n1)$ (i.e. $e(a, n1)$ and $e(b, n1)$) have $\tilde{\lambda}$ greater or equal to $\tilde{\lambda}(e)$. If this is the case, this means that $n1$ is such a supporting vertex.

The triangle is then represented as a solid line indicating that $e(a, b)$ finds a new supporting vertex $n1$ in DN-graph with $\tilde{\lambda}(e)$. As new triangles approach, the algorithm counts the number of supporting vertices for edge (a, b) to form DN-graph, with current value of $\tilde{\lambda}(e)$. After one pass of all triangles, the number of vertices that support each edge's density upper bound $\tilde{\lambda}(e)$ are available for further computation.

Algorithm 1 Triangulation based DN-Graph Mining

Require: Graph $G(V, E)$
1: Triangles = Triangulation(G), $k(e)$=Triangle_count(e)
2: **while** converge AND iteration!=MAX_ITR **do**
3: $sc = 0$, converge=TRUE
4: **for all** Triangles $(a, b, c) \in G$ **do**
5: Increment corresponding $sc(e)$ if e is supported
6: **end for**
7: **for all** edges $e \in G$ **do**
8: **if** $(sc(e) < \tilde{\lambda}(e))$ **then**
9: Find next possible value $\tilde{\lambda}(e)$ for e
10: converge = FALSE
11: **end if**
12: **end for**
13: Increment iteration by 1
14: **end while**
15: **return** $\tilde{\lambda}(e)$ for each $e \in E$

With the supporting neighbors' information, the algorithm is able to determine the upper bound of λ for each graph edge (the upper bound is denoted as $\tilde{\lambda}(e)$). If sufficient supporting vertices are found for $\tilde{\lambda}(e)$ for an edge $e(a, b)$, $\tilde{\lambda}(e)$ is a valid upper bound of $e(a, b)$'s λ value. If there is not enough supporting vertices for $e(a, b)$, the algorithm finds the next possible $\tilde{\lambda}(e)$ value and tests it in the next round of triangulation. The algorithmic description is given in Algorithm 1. Within the algorithm, $sc(e)$ records the number of vertices supporting current $\tilde{\lambda}(e)$ value.

4.1.2 $\lambda(e)$ Bounding Choice

We can derive two variants of DN-graph mining algorithms from Algorithm 1, namely algorithms TriDN and BiTriDN. The two algorithms have different ways to decide the next possible $\tilde{\lambda}(e)$ value. The first variant, called **Tri**DN, decreases $\tilde{\lambda}(e)$ by one (Line 9 in Algorithm 1 becomes $\tilde{\lambda}(e) = \tilde{\lambda}(e) - 1$), if current $\tilde{\lambda}(e)$ cannot obtain sufficient supporting vertices count. This strategy is useful when the triangle counts are close to the actual $\lambda(e)$ values (qualitatively, when $|TC(e) + 2 - \lambda(e)| \leq log\lambda(e)$).

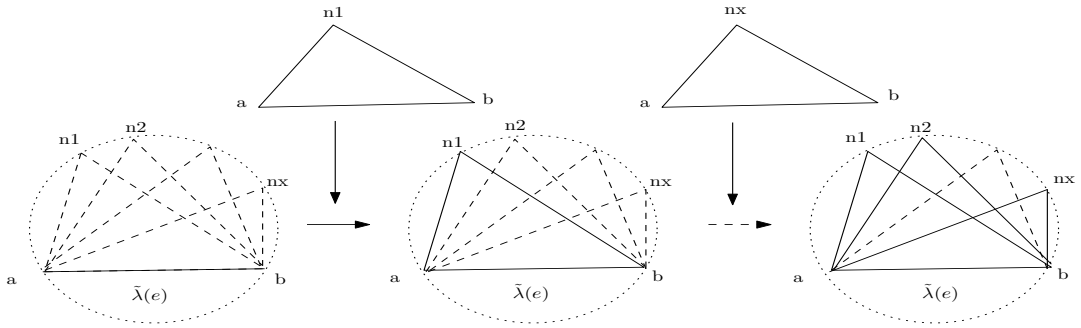


Figure 3: Use Triangle to Refine $\tilde{\lambda}(e)$

When the triangulation results are far above actual $\lambda(e)$, we can employ the second variant, called **BiTri**DN, which adopts a binary search strategy for the next possible value of $DN(e)$. BiTriDN requires additional information of possible $DN(e)$'s range. We use two numbers $lbk(e)$ and $\tilde{\lambda}(e)$ to record the lower bound and upper bound of $\lambda(e)$ value, and $mk(e)$ denotes the medium of range [$lbk(e)$, $\tilde{\lambda}(e)$]. For completeness, we rewrite Line 7 onwards in Algorithm 1. BiTriDN has the advantage of fast convergence if the graph to be mined has many high degree vertices (qualitatively, when $|TC(e) + 2 - \lambda(e)| \geq log\lambda(e)$). Appendix 8.7 gives the proof of the correctness for both bounding choices. Interested readers are referred to Appendix 8.8 for the complexity analysis of both algorithms.

Algorithm 2 Binary DN-Graph Mining Variance "**BiTri**DN"

Require: Graph $G(V, E)$
1: $mk(e) = k(e) = TC(e) + 2, lbk(e) = 2$
2: Get support count $sc_{mk}(e)$ for all edges' $\tilde{\lambda}(e)$ {This part is the same as in Algorithm 1}
3: **for all** edge $e \in G$ **do**
4: **if** ($sc_{mk}(e) < mk(e)$ AND $lbk(e) < \tilde{\lambda}(e)$) **then**
5: $\tilde{\lambda}(e) = mk(e) - 1$, converge = FALSE
6: **else**
7: $lbk(e) = mk(e)$
8: **end if**
9: $mk(e) = \frac{\tilde{\lambda}(e)+lbk(e)}{2}$
10: **end for**
11: **return** $\tilde{\lambda}(e)$ for each $e \in E$

4.2 Extension of DN-Graph Mining to Semi-Streaming Graph

The semi-streaming graph model assumes the vertices of the graph can be fitted into main memory, and the interactions among vertices are stored in an ordered manner within the secondary storage. While this assumption may not hold for arbitrarily large graphs, we can still handle up to Giga scale vertices (assume $|V|$ vertices require $|V|log|V|$ bits storage) with today's main memory capacities. Following the nature of physical storage devices, our streaming model assumes random access in primary storage (i.e. memory) and only sequential access in secondary storage. In the secondary storage, graph interactions are stored in the form of adjacency list. As a feasible solution towards a streaming graph $G(V, E)$, it should not exceed $log|V|$ scans of G's adjacency list.

In the semi-streaming graph setting, the exact triangulation algorithm proposed in [13] cannot be directly applied in the DN-graph mining solutions. The information of the neighbors are stored in secondary storage and may not be immediately available when the algorithm retrieves it.

In view of above difficulty, our streaming solution first performs a semi-streaming triangulation, followed by the complete DN-graph mining solution in semi-streaming setting.

The neighborhoods join operations are in fact the process of determining the similarity between two sets. The most well-adapted measurement for set similarity is Jaccard coefficient. For two sets A and B, Jaccard coefficient is calculated as $J(A, B) = \frac{|A \cap B|}{A \cup B}$.

In the semi-streaming graph setting, it is however expensive to calculated Jaccard coefficient between two neighborhoods. Since the operation of set joining requires expensive pre-processing of sets such as sorting or heap building.

In view of above difficulty, we use the property of min-wise independent set to approximate Jaccard coefficient. When dealing with large sets, min-wise independent property approximate set intersection size using sequential scan only.

Suppose A and B are defined on the set universe X, and π is a permutation over universe X, the min-wise independent property states: If $\pi[X]$ is a uniformly chosen random permutation over X, and $W \subset [X]$ is any subset over the universe, and $\pi[W]$ is the projection of W by permutation π, then the probability that two subsets' minimal projected images are equal is the same as the Jaccard coefficient. Formally, $P[min(\pi[A]) == min(\pi[B])] = J(A, B)$. [5] proposes a streaming local triangle counting algorithm based on min-wise independent property(Appendix 8.6 outlines the technique).

The algorithm proposed in [5] estimates local triangulation using edge scans. It forms the first step of algorithm StreamDN. The next step is to calculate each edge's λ value using only edge scans [1]. StreamDN, as presented in Algorithm 3, adopts the bounding process as algorithm BiTriDN. That is:

The only difference between the streaming version of the algorithm and BiTriDN is when counting the supporting vertices. In StreamDN, we can only access the graph edges sequentially. In view of the restriction, proposition 4.1 is relaxed to as follows:

PROPOSITION 4.2. ***Relaxed*** *Neighbor Bounding of* $\lambda(e)$
Given a graph edge $e(u, v)$ *and the joint neighbor set* $N_\cap(u, v)$, *we say a vertex* $w \in N_\cap(u, v)$ *is a supporting vertex of* $\tilde{\lambda}(e)$ ***if*** $\lambda(u, w) \geq \tilde{\lambda}(e)$. *An integer* k *is a valid upper bound of* $\tilde{\lambda}(e)$ *if and only if there are at least* k *of such supporting vertices in* $N_\cap(u, v)$

The proof for proposition 4.2 is omitted for brevity.

[1]For brevity, in following parts of the paper, we use streaming DN-graph mining algorithm instead of explicitly stating "semi-streaming"

Algorithm 3 Streaming DN-Graph Mining Algorithm "StreamDN"

Require: Graph $G(V, E)$, r : # of scans of graph links k : # of bits for hash values

1: $mk(e) = \tilde{\lambda}(e) = TC(e), lbk(e) = 0$
2: Triangulation and store triangle count $TC(v, u)$ for all $e \in E$ as in algorithm 5 in appendix.
3: **while** !converge AND iteration!=MAX_ITR **do**
4: $sc_k = 0$ $ubk(e) = \tilde{\lambda}(e) = TC(e), lbk(e) = 0$
5: **for all** edge $(u, v) \in G$ **do**
6: $sc_k(u, v)$=number of u's neighbor with $\tilde{\lambda}(u, v)$
7: Bound $\tilde{\lambda}(u, v)$ using $ubk(u, v)$/$lbk(u, v)$/$sc_k(u, v)$ {the same as Algorithm 2}
8: **end for**
9: **end while**
10: **return** $\tilde{\lambda}(e)$ value for every graph edge e

4.2.1 Error-Bound on Streaming DN-Graph Mining

As mentioned in the previous subsections, the number of permutations adopted determines the estimation accuracy of min-wise independent property. The error, however, is bounded. If we denote the joint size as $X = |A \cap B|$ and the estimated value $\overline{X} = TC(A, B)$, the error bound is:

$$P[|\overline{X} - X| > \epsilon X] \geq 2e^{-\frac{\epsilon^2}{3} r J(A,B)} + \frac{m|A \cup B|}{2^k - 1} [5]$$

The complexity analysis for streaming DN-graph mining is presented in the appendix. In fact, the triangle based approach algorithm can also be applied to dynamic graphs and we attach the adapted algorithm for dynamic setting in Appendix 8.9.

5. EXPERIMENTAL STUDY

In this section, we study the performance of the DN-graph mining algorithms. Experimental data come from both theoretically proven data generators ([7]), as well as domain datasets. All the experiments are conducted on a workstation with a Quad-Core AMD Opteron(tm) processor 8356, 128GB RAM and 700GB hard disk. The operating system is Windows server 2003, Enterprize x64 edition.

Synthetic Graph Generators (G_{EC}). We use the *clique hiding graph generator* developed by M. Brockington and J. Culberson [7]. This graph generator randomly embeds a fixed size clique (c) into a graph of ($|V|$) vertices. The graph density, p, is calculated as $p = \frac{2|E|}{|V|(|V|-1)}$. The resulting graphs are random graphs with one known fixed size clique embedded. Table 2 summarizes the key parameters, with the default values highlighted in **bold**.

Parameters	Experimental Range
c: clique size	[20, **40**, 60, 80, 100]
$\|V\|$: # of vertices	[1000,2000, **3000**, 4000, 5000]
p: edge density(%)	[4, 8, **12**, 16, 20]

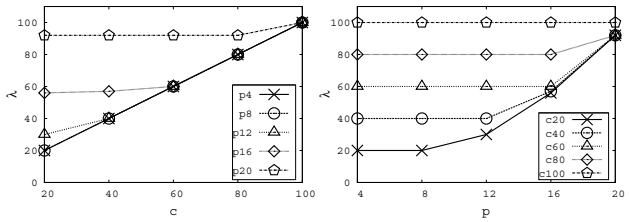
Table 2: Parameter Table

Domain Graph Datasets We also employ 3 real life datasets in our study. These datasets are either collected by domain experts or extracted from well-known public databases. 1) Protein Protein Interaction (PPI) dataset: This dataset[16] contains 17203 interactions among 4930 proteins.2) Netflix dataset: It is compiled from Netflix raw data consisting of 480,000 customers and 17,000 movies records [1]. 3) Flickr dataset: This dataset is derived from the well-known photo sharing network Flickr with 1,715,255 vertices and 22,613,982 edges. Each vertex represents a person.

5.1 Performance Evaluation

5.1.1 DN-Graph Mining Accuracy

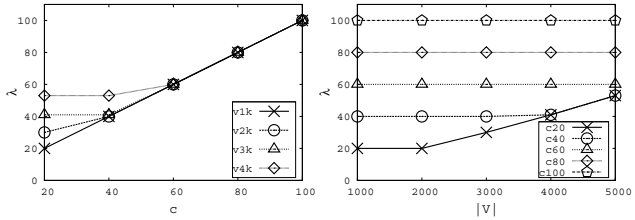
The first set of experiments evaluate the accuracy of DN-Graph algorithms on the synthetic data generated by G_{EC}. We focused on two algorithms - TriDN and BiTriDN. We compared the results and observed similar behaviors between the two algorithms. Due to space limitation, we only present the results of algorithm BiTriDN.



(a) Vary c (b) Vary p

Figure 4: λ Value for Fixed $|V| = 3000$

There are three groups of experiments, each of which fixes a G_{EC} parameter to the default value. Group 1: When we fixed $|V| = 300$, Figure 4(a) and figure 4(a) show the calculated λ values of different datasets. From figure 4(a), the algorithm accurately reports the DN-graph size as c, when the embedded clique is in fact the dense area of the dataset. When the graph is denser ($p \geq \%12$), the clique becomes a less dense area. In these datasets, BiTriDN reports higher $\tilde{\lambda}$ value (up to $\tilde{\lambda} = 90$). We examined the dataset and verified that BiTriDN did find DN-graph with higher $\tilde{\lambda}$ value ($> c$), and confirmed the correctness of BiTriDN's results. Similarly, the results in figure 4(b) shows the $\tilde{\lambda}$ value over different densities, which once more confirms BiTriDN's accuracy. Group 2: When we fixed $p = 12\%$, the results in figures 5(a) and 5(b) show that BiTriDN identifies the λ value accurately. Similar observations are made for experiments in Group 3 when we fixed $c = 40$ (see figures 6(a) and 6(b)).

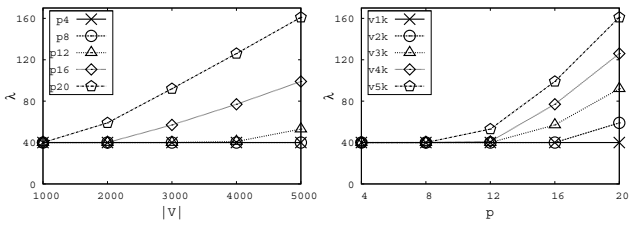


(a) Vary c (b) Vary $|V|$

Figure 5: $\tilde{\lambda}$ Value for Fixed $p = 12\%$

5.1.2 Convergence of DN-Graph Mining Algorithms

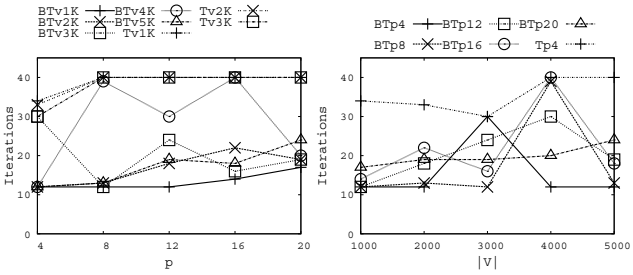
In this set of experiments, we compare the pace of convergence between two algorithms: TriDN and BiTriDN. The synthetic datasets are generated from G_{EC} as well. The maximal iteration is set to be 40 rounds to avoid pro-long running of the experiments. Plots in figure 7(a) and 7(b) show the number of iterations to reach convergence with different graph density p and graph size $|V|$. The



(a) Vary $|V|$ (b) Vary p

Figure 6: $\tilde{\lambda}$ Value for Fixed $c = 40$

results from both plots show that algorithm BiTriDN can converge within 10 to 25 rounds on most parameter settings. However, in most of the experiments, TriDN has not reached convergence after the preset maximal iteration (in figure 7(a) and 7(b), we omitted parameter settings which do not converge). This provides strong support on the claim that BiTriDN converges significantly faster than TriDN.

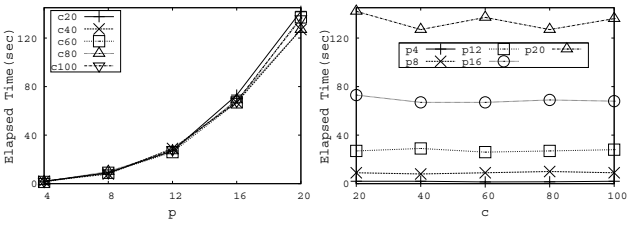


(a) Vary p, fixed c (b) Vary $|V|$, fixed $c = 40$

Figure 7: Convergence of TriDN and BiTriDN

5.1.3 Time Performance in Memory

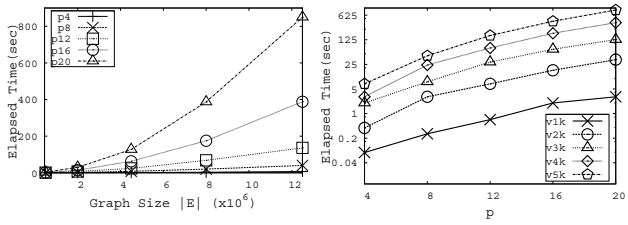
In this study, we evaluate the efficiency (i.e., running time) of the two algorithms TriDN and BiTriDN over the synthetic data generated by G_{EC}. For a fixed parameter setting, the two algorithms converge at different iteration. To remove the effect of different convergence speed towards time performance, all time are measured for only 1 iteration in this study. Both algorithms have almost same behavior for 1 iteration. Figure 8(a) and 8(b) show the



(a) Vary p (b) Vary c

Figure 8: BiTriDN One Iteration Running Time $|V| = 3000$

running time when $|V|$ is fixed. The results match the complexity analysis in section 4.1. The effects of edges distribution change are shown in figure 8(a) and 8(b). The synthetic graph generator G_{EC} varies the edge distribution by varying the embedded clique size c. Experiments on these data always indicate that only $|V|$ and p affect the running time. Figure 9(a) and 9(b) present the effect of different graph size $|E|$ over the running time. The trend over time roughly follows complexity $O(E^{\frac{3}{2}})$.



(a) Vary $|V|$, $c = 40$ (b) Vary p, fix $c = 40$

Figure 9: BiTriDN Iteration Running Time $|c| = 40$

5.1.4 Memory Usage

We also monitor the peak memory usage of both TriDN and BiTriDN on the synthetic data. Figure 10 shows that both algorithms increase their memory usage when the graph size/density increase. Meanwhile, the results tell us that the memory usage of TriDN is always slightly less than that of BiTriDN under the same parameter setting. This is because in BiTriDN, additional memory is used to store both the upper bound and lower bound of the λ value.

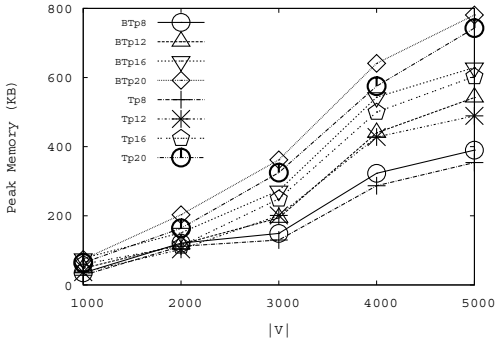
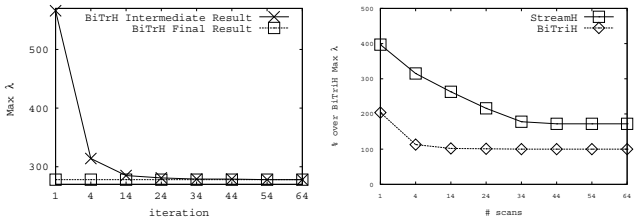


Figure 10: Memory Usage of TriDN and BiTriDN

5.1.5 Test on Very Large Graph

In this experiment, we applied BiTriDN on the Flickr dataset. The running time per iteration is between 55 minutes to 1 hour. The stable memory usage is less than 1G. The program converges after 66 iterations. Each iteration's λ values are recorded for each vertex. When algorithm converges, the largest DN-graph's highest λ is 278. We note that at the 35th iteration, the largest λ value already reaches 279. Figure 11(a) plots the trend of maximal vertex λ value's change. From this experiment, the DN-graph mining results have high availability as the results are updated at every iteration. What's more, if allowing small errors, the program converges very fast. These fast convergence feature is observed for all real datasets we tested on. We did not reports all results due to space limitation.



(a) Convergence (b) StreamDN Accuracy

Figure 11: Performance on Flickr Dataset

5.1.6 StreamDN Performance on Flickr Dataset

In this set of experiments, the StreamDN algorithm is ran on the Flickr dataset. The implementation mimics the behavior of disk scan on main memory as our experiment machine has large enough memory. The results in figure 11(b) shows StreamDN over-estimates with respect to BiTriDN algorithm's results by 72% during the first 66 scans of the whole Flickr dataset. With the analysis of StreamDN's running time complexity, we confirmed that the triangulation based DN-graph mining algorithms can handle streaming setting with reasonable accuracy.

5.1.7 DN-graph Semantics in Various Domain

In the movie co-comments network, two movies are connected by a weighted edge if these two movies share enough commenters such that the Jaccard Coefficient of the two movies' commenters sets is above a certain threshold. Figure 12 shows a set of movies and their interactions found in 100 movies co-comments network with a threshold of 0.5. These movies are reported with $\lambda = 9$. All of the 9 movies have exceptional high IMDB scores (> 8 out of 10, which means they are exceptionally popular). Besides the popularity, we found 7 of them are from USA, while the remaining 2 are from France and Japan respectively. The 9 movies all belongs to genre Violence/Fantasy.

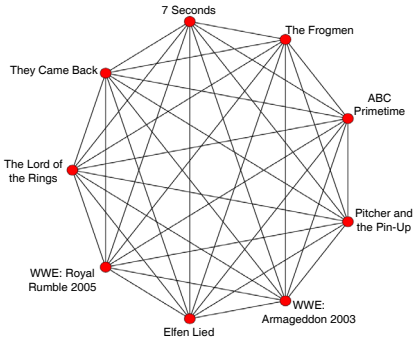


Figure 12: Patterns Discovered in NetFlix

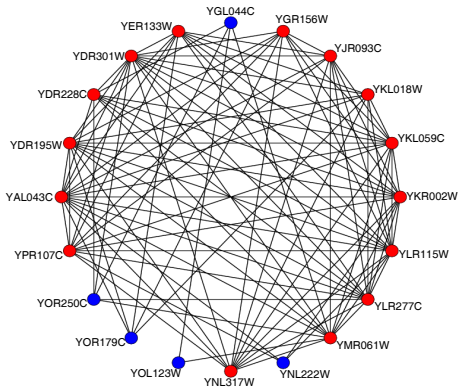


Figure 13: A 20-Protein Complex in Form of DN-Graph

Many proteins are functional only when they are assembled into a protein complex. Our DN-Graph mining algorithm can detect important protein complexes out of large amount of protein-protein interaction (PPI) data. For example, mRNA_CF_Complex in figure 13 is a 20 protein complex confirmed by the benchmark. The red colored nodes are active proteins that functionally interact with other proteins within the complex. These proteins can be successfully detected as an DN-Graph by our algorithm, while others (blue-colored nodes) are missed out in our results. The reason for missing out those proteins is because these proteins have fewer interactions with the rest of the proteins. Even with those missing points, our results are already a significant improvement over known results (For further results, please refer to Appendix 8.10).

6. CONCLUSION

In this paper, we present a new graph dense structure DN-graph. The DN-graph complements popular dense structures by imposing both size and degree constraints. We then discuss the graph local triangulation problem and its connection with DN-graph mining problem. Based on that, we propose solutions to effectively locate DN-graphs. The solutions are set of algorithms catering for different problem settings from in memory to streaming. The iterative, triangulation based solution has the advantages that the details can be abstracted within the triangulation algorithm. Since the algorithm improves the result at every iteration, users can stop the algorithm at any time and get the best results within the time limit. Our experimental study shows our solutions are both time and space efficient.

7. REFERENCES

[1] *Netflix Prize Data Set*, 2009 (accessed July 23, 2009).

[2] J. Abello, M. Resende, and R. Sudarsky. Massive quasi-clique detection. In *Proc. 5th Latin American Symposium on Theoretical Informatics*, pages 598–612, 2002.

[3] I. Akihiro, W. Takashi, and M. Hiroshi. Complete mining of frequent patterns from graphs: Mining graph data. *Machine Learning*, 50(3):321–354, 2003.

[4] P. Aloy, B. BäPttcher, H. Ceulemans, C. Leutwein, C. Mellwig, S. Fischer, A. C. Gavin, P. Bork, S. G. Furga, L. Serrano, and R. D. Russell. Structure-based assembly of protein complexes in yeast. *Science*, 303(5666):2026–2029, 2004.

[5] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *ACM KDD '08*, pages 16–24, New York, NY, USA, 2008.

[6] V. Boginski, S. Butenko, and P. Pardalos. Mining market data: a network approach. *Computer Operational Research*, 33(11):3171–3184, 2006.

[7] M. Brockington and J. Culberson. Camouflaging independent sets in quasi-random graphs. *DIMACS Series*, 26:75–88, 1994.

[8] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *VLDB'05*, pages 721–732, 2005.

[9] J. Han, N. Stefanovic, and K. Koperski. Selective materialization: An efficient method for spatial data cube construction. In *PAKDD'98 [Lecture Notes in Artificial Intelligence, 1394, Springer Verlag, 1998]*, pages 144–158, 1998.

[10] H. Hu, X. Yan, Y. Huang, J. Han, and X. Zhou. Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics*, 21:213–221, 2005.

[11] C. F. J. Rivas. Proteincprotein interactions essentials: Key concepts to building and analyzing interactome networks. *PLoS Computational Biology*, 6:6, 2010.

[12] R. Karp. Reducibility among combinatorial problems. *The Journal of Symbolic Logic*, 40:618–619, 1975.

[13] M. Latapy. Practical algorithms for triangle computations in very large (sparse (power-law)) graphs. *Journal of Theoretical Computer Science*, 407:458–473, 2008.

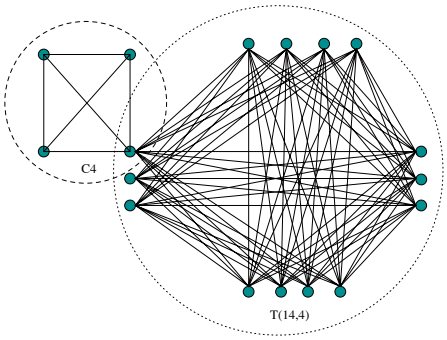
[14] T. Schank and D. Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In *WEA*, pages 606–609, 2005.

[15] N. Wang, S. Parthasarathy, K. L. Tan, and A. Tung. Csv: visualizing and mining cohesive subgraphs. In *SIGMOD'08*, pages 445–458, 2008.

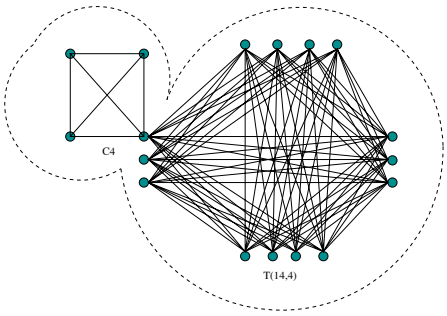
[16] I. Xenarios and L. S. etc. DIP, the database of interacting proteins: A research tool for studying cellular networks of protein interactions. *Nucleic Acids Research*, 30(1):303–305, 2002.

8. APPENDIX

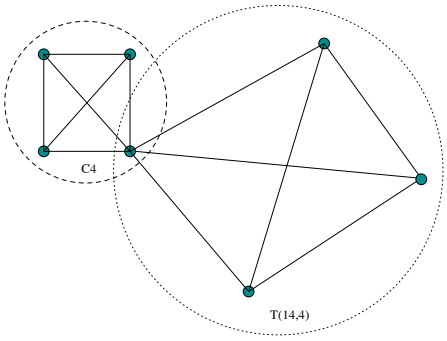
8.1 An Illustrative Example on Different Density Patterns



(a) C4,T(14,4)



(b) δ Quasi clique



(c) Max Clique

Figure 14: Comparison between Different Dense Substructures

Figure 14(a) to 14(c) present the mining results of two different density criteria on an illustrative graph. The graph in Figure 14(a) contains a 4-clique loosely attached to a Turan's graph. A Turan's graph $T(M, N)$ is a special class of graph, in which N graph vertices are divided equally (or as equal as possible) into N groups. Every pair of vertices from two different groups has edge connecting them, while members in the same group are not connected. In fact, Figure 14(a) embeds a Turan's graph $T(14, 4)$. Judging by the interactions, there are two interesting substructures in Figure 14(a). One is the 4-clique and the other is the Turan's graph. If we apply δ quasi-clique mining, when setting $\delta = 0.8$, the mining result includes the whole example graph (figure 14(b)). The mining results are so relaxed that two dense substructures cannot be distinguished. If we insist on regularity of the pattern (e.g. a clique), we can only find subgraphs of size 4 (as indicated in 14(c)) while missing the Turan's graph.

8.2 Proof of Lemma 3.1

First we prove the necessary condition. If a graph contains a DN-graph g with $\lambda = d - 2$ and $|g| = d$, according to the definition of DN-graph, g has d vertices, which shares $(d - 2)$ common neighbors with every connected vertices in g. For any two distinct vertices v and u in g, if they are connected, they Both connect to every other vertices inside g. This holds for every vertex pair, which indicates that if g has an edge, it is a clique. While g is a DN-graph, it does not contain a proper super graph which is also a DN-graph, it thus does not have a proper super graph with $\lambda = d - 1$ and $|g| = d + 1$. It does not contain a clique of size $d + 1$.

The sufficient conditions proof is: if a graph contains a closed clique of size d (*), this clique has d vertices and each pair of vertices share $d - 2$ common neighbors. $\lambda = d - 2$ in this case according to λ definition. Being a closed clique, it does not have a proper super graph which is also a clique. This indicates that it does not have a proper super graph with .$\lambda \geq d - 1$ and $size = \lambda + 2$. Suppose there is a super graph with $\lambda = d-2+\delta$ and $size = d+\epsilon$, there must be some vertices which are not inside g connecting to at least $d - 2 + \epsilon - \delta$ d-clique vertices, while $d - 2 + \epsilon - \delta \leq d$, indicating $\epsilon - \delta \leq 2$, which means $size \leq \lambda$. This is either impossible or the super graph is a clique, which contradicts with the assumption (*). So there is not possible such super graph exist. So the d-clique is an DN-graph. This complete the proof.

8.3 Proof of Theorem 3.1

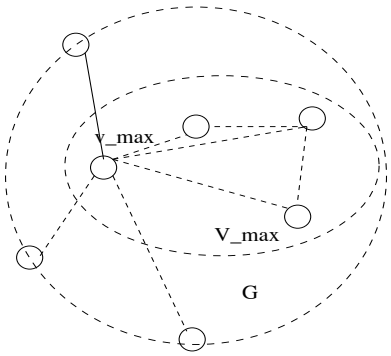


Figure 15: Proof of Theorem 3.1

To prove the correctness of theorem 3.1, we use the abstract graph in figure 15. The complete proof consists of two steps. Firstly, G' must exist. Secondly, G' must contain some max-min DN graph. To prove the existence of G', we construct G' using graph vertices/edges and their λ values. First pick a vertex v with $\lambda(v) \geq \lambda(u)$ for all $(u \in N(v))$. Denote $\lambda(v)$ as λ_{max}. By the definition of local λ value, $\lambda(v)$ participates in a connected graph G' with $\lambda(G') = \lambda_{max}$. From v, we find all its immediately connected neighbors that have $\lambda(u) = \lambda_{max}$. From each u, we find u's immediately connected neighbors with local λ value λ_{max}. This process propagates until no such neighbor exists. The collection of discovered vertices form a connected subgraph G' with λ value λ_{max}.

Next, we show that G' contains a DN-graph. By first part of the proof, G' contains all vertices and edges with λ value λ_{max}. For a vertex $v' \in G'$, it only can form DN-graph of $\lambda = \lambda_{max}$ with vertices inside G'. If denoting the minimal set of vertices from G' that form an DN-graph with v' as V_{min}, the subgraph $V_{min} \cup v'$ is also a DN-graph. This proves that a graph G' containing the set of vertices with $\lambda(v) = \lambda_{max} > \lambda(u)$ where $u \in N(G')$ must participate in a DN-graph. The condition that $\lambda(v) = \lambda_{max}$ and

$\lambda_{max} > \lambda(u)$, where u is the neighbor vertices of G', means the graph G' contains vertices with local maximal λ value. Since graph G' is always a super graph of some DN-graph, If a solution can find G', the DN graph can be located within G'.

With above two steps, we prove the correctness of theorem 3.1.

8.4 Closed Clique Finding is NP-Complete

A closed clique is the local maximal clique, where no proper super graph of it is also a clique. The problem of detecting cliques is a well known NP-complete problem, which is first discussed in the landmark paper [12]. As a clique possessing certain property (here, local maximality), a closed clique detection problem is also an NP problem.

8.5 Graph Triangulation Algorithms

The exact graph triangulation algorithm was first proposed in [13].

Algorithm 4 Local Triangulation Algorithm

Require: Graph $G(V, E)$
1: $mk(e) = k(e) = TC(e) + 2$, $lbk(e) = 2$
2: Order vertices and edges according to degrees
3: **for all** dense vertex $v \in G$ **do**
4: Retrieve all v's dense neighbors u
5: Joint v and u's neighborhoods to find triangle $< v, u, w >$
6: **end for**
7: **for all** sparse vertex $e(v, u) \in G$ **do**
8: Join v and u's neighbor lists to find triangles containing edge triangle $< v, u, w >$
9: **end for**
10: **return** All triangles in G

Algorithm 4 separates the vertices in $G(V, E)$ into two classes: dense and sparse. Vertices with degree greater or equal to $\sqrt{|V|}$ are dense vertices, the remaining are sparse vertices. An edge with both end vertices being sparse vertices is a sparse edge. For a dense vertex v, the local triangulation algorithm perform a join on v's immediate neighbors and the neighborhood list v's neighbor, u. The size of the join set is the triangle count of edge (v, u). Similarly, for sparse edge (v', u'), algorithm join the neighborhood list of v' and u'.

8.6 Approximate Graph Triangulation

The approximate graph triangulation algorithm was first proposed in [5].

The basic idea of the streaming triangulation algorithm in Figure 5[5] is to generate r times permutation over vertices set V. For each permutation, algorithm records every vertex' minimal neighbors within the permutation. After obtaining the minimal neighbors for each vertex, the algorithm scans the graph once to check whether the minimal neighbors of two connected vertices are equal. If they are equal, algorithm increments count Y for the connected vertices. The estimator for vertex triangle count is:

$$\sum_{u \in N(v)} \frac{Y(v,u)}{Y(v,u)+r}(|N(v)| + |N(u)|)[5]$$

This estimator is derived from min-wise independent property. We use small number of permutation to estimate all permutations over graph vertices.

Algorithm 5 Streaming Triangulation Algorithm

Require: Graph $G(V, E)$, r : # of scans of graph links, k : # of bits for hash values
1: $mk(e) = k(e) = TC(e) + 2$, $lbk(e) = 2$
2: $Y = 0$, $min(V) = 0$
3: **for** $s = 1$ TO r **do**
4: Hash every vertex label $h_s(v)$ to any random k bits
5: **for all** vertex $v \in V$ and its neighbor u **do**
6: min(v) = min(min(v),$h_s(u)$)
7: **end for**
8: **for all** vertex $v \in V$ and its neighbor u **do**
9: **if** (min(u)==min(v)) **then**
10: Increment $Y(u, v)$ by 1
11: **end if**
12: **end for**
13: **end for**
14: **for all** every vertex v in G **do**
15: $TC(v) = \sum_{u \in N(v)} \frac{Y(v,u)}{Y(v,u)+r}(|N(v)| + |N(u)|)$
16: **end for**
17: **return** All triangles in G

8.7 Correctness of λ Bounding

Let us denote the actual λ value for an edge as $\lambda(e)$, and the exact supporting neighbor count as sc. The upper bound of λ value is denoted as $k(e)$ and the supporting neighbor count of $k(e)$ is denoted as $sc_k(e)$. To prove the correctness of Algorithm 1, we need to show that 1) $k(e)$ is always an upper bound of $\lambda(e)$. 2) $k(e)$ converges.

PROOF. 1) For the first bounding choice, At the beginning of the algorithm 1, $k(e)$ is equal to triangle count for e, $TC(e)$. since $k(e) \geq \lambda(e)$. If the algorithm stops, the upper bound invariance holds. Suppose the invariance holds for iteration $i > 0$, $k(e) \geq \lambda(e)$, at iteration $i+1$, $k(e)$ is updated to $k(e)-1$ when this condition holds: the number of neighbors having λ values greater or equal to $k(e)$ is less than $k(e)$. This condition uses the value of k from the neighborhood vertices to verify whether the current iteration's $k(e)$ value is valid. Since the neighbor's k values are the upper bounds their actual λ values, the number of qualified candidates $sc(e) \geq sc_k(e)$. Thus when $sc_k(e)$ is less than $k(e)$, $k(e)$ is definitely greater than the real $\lambda(e)$ value (as in 4.1). In that case, $k(e)$ is reduced by 1. And the new $k(e)$ value is still an upper bound. The upper bounds invariance is proven to hold. 2) For the second bounding choice (as in algorithm 2), the proof of upper bound invariance follows with the exceptions that this bounding choices test on the median of possible λ range instead of $k(e) - 1$.

The convergence of $k(e)$ is due to the monotonic decreasing of $k(e)$. The algorithm initializes $k(e)$ as triangle counts. This is an upper bound of $\lambda(e)$. After that, algorithm only decreases $k(e)$. As $\lambda_a(e)$ is always an upper bounds of the actual value $\lambda(e)$, $\lambda_a(e)$ value will converge. □

8.8 Complexity Analysis

The triangulation algorithm (in Appendix 8.5) sorts vertices and adjacency list into descending order of degrees. The operations require $O(|V|log|V|)$ time complexity. After that, it counts triangles within the graphs for each vertex. To count triangles, the algorithm separates vertices into dense vertices and sparse ones according to vertices' degrees. For dense vertex, the algorithm lists the number of triangles $|V|$ participating in $O(|E|)$ time. The total complexity for counting dense vertices is $O(|V||E|)$. For sparse vertices, the algorithm counts sparse edges that intersects with sparse vertices.

For a sparse vertex/edge, its neighborhood size is at most constant (say S). The counting over sparse edges requires $O(S|E|)$ time. If setting $S = \sqrt{|E|}$, taking into consideration of the complexity of counting on dense vertices, The time complexity for triangle counting procedure is $O(|E|^{\frac{3}{2}})$.

TriDN in Algorithm 1 iterates on all triangles that form the graph. Each iteration also requires $O(|E|^{\frac{3}{2}})$ time. For a fixed number of iteration k, the algorithm needs $O(k|E|^{\frac{3}{2}})$ time in total. If insisting on convergence, the algorithm may need up to $O(k|V||E|^{\frac{3}{2}})$. As we may need to test local λ value $\lambda(e)$ from $v-2$ down to 3. The iterative version of the algorithm for λ mining reduces time complexity to $O(klog|V||E|^{\frac{3}{2}})$ since it employs the binary search paradigm to test possible $\lambda(e)$ for every e. The space complexity of both DN graph mining algorithms are similar. The triangulation stage requires $O(|V|log|V|+|E|)$ while the iteration process requires $O(|E|)$.

For streaming DN graph mining, Algorithm 3 first performs semi-streaming triangulation following the idea proposed in [5]. While the semi-streaming triangulation scans the graph r passes to apply min-wise independent set principle and an additional pass to calculate the estimator of triangle counts, its complexity is of $O(r|E|)$. The remaining steps of streaming DN graph mining requires $O(|E|)$ time for every iteration. For a fixed number of iteration w, the algorithm needs $O(w|E|)$ time. In summary, the time complexity of streaming DN graph mining is of $O(k|E|)$, where k is a constant.

8.9 Dynamic DN-Graph Mining

We have discussed solutions for finding handling static graphs. These solutions can be extended to graphs with dynamic topology with minimal modification.

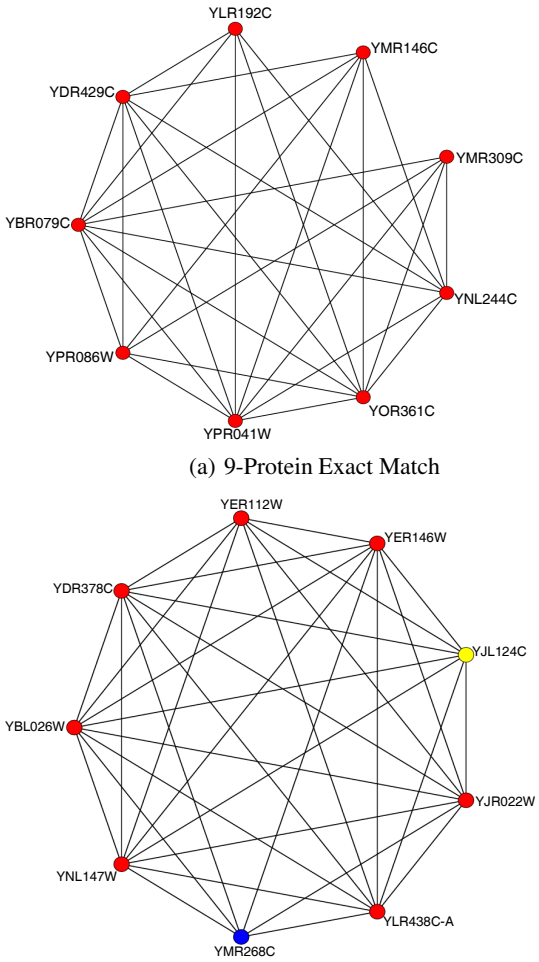
Recall that the triangulation based DN-graph mining solution consists of two stages. In the first stage, algorithm in figure 1 performs local triangulation on the whole graph. The next step is to iterate on each discovered triangle. For each triangle, the algorithm uses it to verify whether it can support its edges' $\lambda_a(e)$ value. After scanning all triangle once, the $\lambda_a(e)$ values for each edge are updated accordingly.

A dynamic graph $\mathcal{G}(\mathcal{V},\mathcal{E}) = \{G^t(V^t,E^t)\}$ changes its topology over time. Such dynamics can also be modelled as the emerging and disappearing of triangles inside $\mathcal{G}$. At each discrete time t, the instance of a streamed graph is the set of vertices and edges presented at t, we use $G^t(V^t,E^t)$ to represent it. Without loss of generally, this chapter only concerns the addition of edges. When a new edge e appears between vertex a and b, this edge's initial $\lambda_a(e)$ is set to be the size of $N(a) \cap N(b)$, while edge $\lambda_a(a,n)$ and $\lambda_a(b,n)$ increase by 1 if they share neighbor vertex n before new edge comes. After the process of adjusting $\lambda_a(e)$ values for dynamically affected edges, we then iterate on the new set of triangles following the same way as in algorithm in Figure 1.

8.9.1 Complexity of Dynamic DN-Graph Mining

Denoting the total number of edges of a dynamic graph $\mathcal{G}$ as $|E^*|$. Followed previous discussion, we only process each edge once when it first appears or its neighborhood information changes. If the graph is a sparse graph, the neighborhood size of any vertex is a constant. Thus in sparse graph, the complexity for dynamic DN-graph mining is of $O(k|E^*|)$ where k is the number of iterations. The space complexity is $O(E^*)$ as we need to store DN information for each edge appeared.

8.10 More Semantic Findings



(a) 9-Protein Exact Match

(b) Protein Complex snRNP

Figure 16: Additional Interesting Subgraphs

Figure 16(a) and figure 16(b) show two additional DN graphs identified by our mining algorithm. Figure 16(a) is an exact 9-protein complex matched by our algorithm from PPI dataset. Besides identifying known protein complex, DN-graph mining results can also predict unknown proteins' functionality. These proteins are not included in the existing protein complexes benchmark due to systematic experimental constraints. However, by constructing the PPI across different experimental sources, this unknown protein may be included into a synergetic DN graph. In figure 16(b), protein YJL124C was predicted to have the same functionality as the rest inside the graph. We could not find supporting evidence in known snRNP protein complex benchmark. However, by checking with domain experts, we confirmed the correctness of our finding. Note that certain protein (such as YMR268C) could not be detected due to low connectivity with other members in the complex. With the observation of high connectivity among the rest of the proteins, we strongly urge biologists to experimentally search for the "missing"connections between missed proteins and the rest of the protein members. (For clarity, matches are marked red; missing proteins are marked as blue points. Proteins that are not present in known benchmark pattern but discovered as members of an DN-Graph are marked as yellow.)