Understanding evolving group structures in time-varying networks

Paul Caravelli, Yifang Wei, Daniel Subak, Lisa Singh, and Janet Mann
Georgetown University
Washington, DC 20057

Abstract—**This paper presents a framework for identifying persistent groups and individuals across multiple time granularities in dynamic graphs. Understanding the longevity of groups and the relevance of individuals within a group is important in many fields, including sociology, biology, economics, psychology, and political science. Different clustering algorithms have been proposed for static and dynamic graphs. However, using the clustering results to understand the changing dynamics of groups can be difficult. In order to better understand how groups evolve and the level of cohesion within these groups, we propose a holistic dynamic clustering framework that allows the user to adjust the underlying algorithms for clustering nodes in a graph that changes over time and then use the final clusters to produce a time hierarchy that highlights the groups and individuals persistent during different time periods. We test our framework and algorithm both on synthetic and real world data. Our findings indicate that our approach not only yields highly accurate results, but also detects unexpected variations in group structure.**

I. Background and Motivation

Understanding of the stable and changing behavior of groups is important for many applications. When do groups form? How do they evolve or change over time? How stable or transient are different members of a group? Answering these questions offers insight into the nature of online social networks, citation networks, and animal communities, etc. To address these questions, we present a flexible framework for identifying and exploring these evolving group structures in social network graphs across different time granularities.

Many graph clustering/community detection algorithms have been proposed. While some find communities or groups in static graphs [1][2][3][4][5][6][7] and others find them in dynamic graphs [8][9][10][11][12][13], we identify structural clusters at different time granularities for visual analysis. To accomplish this, we propose a holistic framework that incorporates three complementary components: dynamic graph clustering, multi-granular dynamic alliance detection, and temporal visual exploration. Figure 1 shows the different components of our framework. Given a dynamic graph, the framework begins by computing clusters at each time point. Those clusters are then used to create a hierarchy that shows *group alliances* (see below) at different time resolutions. Both the clusters and the alliances can then be explored using an interactive visualization to analyze the stable and changing group structure at different time scales. We now describe and highlight the contributions of each component.

A. Dynamic Graph Clustering

Different dynamic graph clustering algorithms have been proposed in the literature [8][9][10][11][12][13]. Many of these algorithms begin with or embed an initial clustering based on a static clustering algorithm. Unfortunately, for different applications, it is not always obvious how to transition these static clustering algorithms through time or which static clustering algorithm will lead to meaningful clusters. To that end, we propose a dynamic graph clustering framework that (1) allows researchers to adjust the underlying static clustering algorithms within the framework of a single dynamic clustering algorithm; (2) preserves structural continuity over time; and (3) uses a greedy algorithm to identify important structural changes in clusters at a particular time point and focuses re-clustering to only those parts of the graph.

B. Multi-resolution Dynamic Alliance Detection

Dynamic graph clustering algorithms capture time at a single resolution. Having the ability to identify short-term and long-term clusters, i.e. clusters at different time resolutions, provides researchers with insight into the changing and stable social structures of a population. Individuals that are members of a group or community for a long period of time form a *group alliance*. Individuals that switch from group to group may be characterized as *social butterflies* or transients, who are members of different groups or communities over time, but do not exhibit a strong preference toward one particular group or community. To capture these different group dynamics, we propose detecting *multi-granular, dynamic alliances*. We accomplish this by using frequent pattern mining on base-resolution dynamic clusters to identify alliances and social butterflies at multiple time resolutions, where each resolution is represented as a different level of a time-hierarchy.

C. Temporal Visual Exploration

While researchers can investigate individuals in the different levels of the hierarchy textually, hierarchies naturally lend themselves to visual analytics. Therefore, we also present an implementation of an interactive visual application for time-based hierarchical analysis of clusters. By giving users the ability to visually interact with base-level clusters and multi-granular alliances, researchers can better understand temporal community dynamics. Because of space limitations. we will focus on the other two components of the framework.

The contributions of this work are as follows: (1) a holistic approach to multi-granular, dynamic network clustering; (2) a localized greedy algorithm for single resolution, dynamic network clustering; (3) a frequent graph mining algorithm for detecting multi-granular, time-based groups and alliances; (4) an empirical analysis of both algorithms on synthetic data and real world animal community.

The remainder of this paper is organized as follows. Section II presents related literature. We describe our dynamic

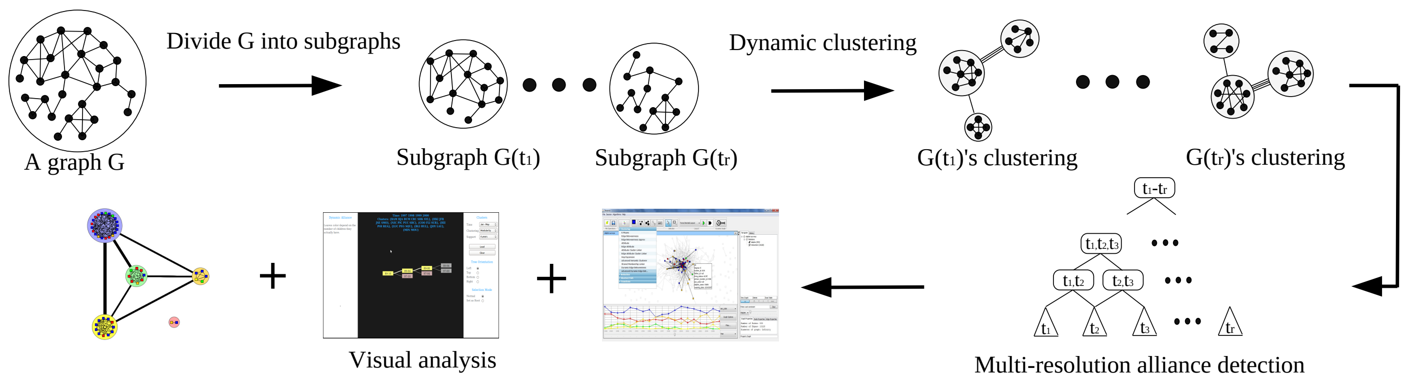


Fig. 1: A high level view of the proposed framework

clustering framework in section III. Our clustering evaluation on synthetic data and real world data, as well as an expert evaluation of the alliance detection algorithm are presented in IV. Finally, conclusions are presented in section V.

II. Related Literature

Clustering of time series and temporal data is an active area of research. We refer you to surveys by Liao [14] and Antunes and Oliveira [15] for an overview of the area. While the techniques presented in those surveys are very relevant to our approach, our work is focused on using graph structure at different time points to identify similarities, differences, and changes in clusters and groups over time.

Das et al. [16] present a dynamic clustering algorithm that clusters data at each time step. Their goal is to identify and group time steps with similar clusters. In contrast, our goal is to identify clusters and individuals that exist at different resolutions of time. Chakrabarti et al. [17] proposes an evolutionary clustering algorithm that uses k-means and agglomerative hierarchical clustering in conjunction with the history of previous time steps when clustering the current time step. Similar to our work, they look to maintain consistency among clusters across time steps. However, their approach uses the history of many time steps, while ours is a local greedy strategy that is more incremental.

Other relevant dynamic graph research includes the evolution of graphs based on various topological properties [18], group formation [19], group evolution [20], identification of behavioral events and key patterns [21], actors that change affiliations together [22], and identification of spread blockers [23], and prominent actors in dynamic affiliation networks [24], to name a few. None of these mentioned worked consider multi-resolution clusters.

Tantipathananandh et al. [8] treat cluster identification as a combinatorial optimization problem with two subproblems: identifying which groups at time t map to which groups at $t+1$ and assigning individuals to those groups. Palla et al. [9] also proposed a dynamic community detection algorithm that uses the clique percolation method at each time point to identify k-clique communities. While both these works have similar notions of capturing the dynamics between time steps, their formulations differ from ours since they are identifying cliques in overlapping communities. We do not require our communities to be cliques and at a single time point, our communities do not overlap. Similarly, Aggarwal et al. [10] also use clique percolation to investigate time dependencies of overlapping communities. Their focus is on estimating a community's lifetime.

Gorke et al. [11] propose a dynamic graph clustering algorithm that uses minimum-cut trees for determining clusters and maintaining temporal smoothness. Hopcroft et al. [12] present an approach for identifying natural or stable communities using an agglomerative clustering algorithm with a similarity distance based on a cosine function.

Both of these mentioned algorithms are complementary approaches for identifying dynamic clusters. An approach that considers communities across varying time intervals is proposed by Sun et al. [13]. Their method finds similar communities from consecutive time steps by minimizing an objective function based on encoding length. Our approach differs from both these works since it explicitly lets the user adjust the static component of the algorithm and incorporates multi-resolution groups.

III. Dynamic Clustering Framework

We represent a network as a graph $G=(V,E)$, where the graph is composed of a set of nodes $V(G)=\{v_1,\ldots,v_n\}$ and a set of edges $E(G)=\{e_1,\ldots,e_m\}$. We define a cluster as a set of nodes from our graph that have been grouped together and combined with associated edges. A cluster C_i, thus, has a set of nodes contained in the cluster, $V(C_i)$, and a set of *in-cluster edges*, $E(C_i)$, composed of all edges such that both endpoints of the edge are contained in $V(C_i)$. We further define *cross-cluster edges* $E(C_i,C_j)$ for clusters C_i and C_j as the set of edges such that each edge has one endpoint in C_i and the other endpoint in C_j. A *cluster set* $\mathbb{C}$ is defined as a group of clusters $\{C_1,\ldots,C_p\}$, where p is the number of clusters in the cluster set, $\bigcup_{i=1}^{p} V(C_i)=V(G)$, and $V(C_i)\cap V(C_j)=\emptyset$.

Since we are interested in dynamic networks, we assume each edge in the graph is associated with a time point t_k, and $T=\{t_1,t_2,...,t_r\}$ represents the set of time points. Then all the edges associated with time point t_k and all the incident nodes compose a graph $G(t_k)$, a sub-graph of $G(t)$ [1]. A node might occur in multiple subgraphs. Similar to our static clustering notation, let $C_i(t_k)$ represent a cluster at time point t_k and $\mathbb{C}(t_k)$ represent a cluster set at time point t_k, where

[1]We will use G and $G(t)$ interchangeably in this work

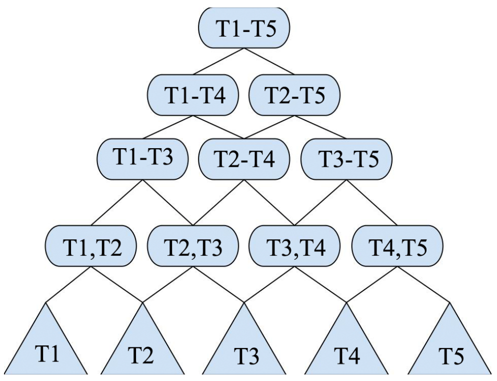


Fig. 2: An example time hierarchy

$\bigcup_{i=1}^{p} V(C_i(t_k)) = V(G(t_k))$, and $V(C_i(t_k)) \cap V(C_j(t_k)) = \emptyset$. Then the dynamic clustering results across T are represented as $\{\mathbb{C}(t_1), \ldots, \mathbb{C}(t_r)\}$.

Lastly, let $\mathbb{T}$ be a generalized time hierarchy of height h, where each level of $\mathbb{T}$ represents a different time resolution, and the nodes at each level map to a time window, $w(l)$, containing l time points. The resolution of a level of the time tree equals $1/l$. The leaf nodes of $\mathbb{T}$ (the highest resolution) map to one time point, t_i (see Figure 2 for an example). For each subsequent level of the tree, the number of time points that map to a node increases by one. The root node maps to a window containing all the time points (lowest resolution). Because two adjacent nodes at the same level differ in only one time point, each node in the hierarchy (except for the root node) has two parents, allowing for overlapping windows at a single resolution.

A. Dynamic Clustering Algorithm

Objective: Given a dynamic graph $G(t)$, find a set of clusters $\mathbb{C}(t_i)$ at each time point t_i where structural continuity is maintained, the number of nodes that need to be re-clustered in $G(t_i)$ is minimized, and the accuracy of the clusters is high.

One approach to dynamic clustering is to apply a static clustering algorithm to each subgraph $G(t_i)$. Unfortunately, this approach requires re-clustering all the nodes at each time point. It does not optimize to reduce the number of nodes that need to be re-clustered at each time point or maintain structural continuity, i.e. only re-clustering those parts of the graph that have a large number of changes.

To address these issues, we propose a localized, greedy, dynamic clustering algorithm. At a particular time point t_k, our algorithm clusters nodes that are not in the previous time point, t_{k-1} and re-clusters nodes that are in clusters having significant change from one time point to the next. This allows us to maintain structural continuity from one time point to the next and avoid recomputing all the clusters at each time step.

A high level view of this greedy algorithm is presented in Algorithm 1. We can use any static algorithm designed for graphs, e.g. Newman's modularity clustering [2], for the initial clustering at time t_1 (Line 2), while for every subsequent time step, we begin with the clusters from the previous time step and update them using our operations: add/remove (Line 5 - 6), split (Line 7), recluster (Line 8), and disband (Line 9). We pause to mention that reordering the operations can lead to different clustering results. This ordering was selected based

Algorithm 1: DYNAMIC CLUSTERER

```
Input:
  A series of dynamic subgraphs: $\{G(t_1), \ldots, G(t_r)\}$
  Splitting Threshold: $L$
  Reclustering Threshold: $H$
  Disbanding Threshold: $M$
Output: $\{\mathbb{C}(t_1), \ldots, \mathbb{C}(t_k)\}$
1 Create a set of cluster sets $\{\mathbb{C}(t_1), \ldots, \mathbb{C}(t_k)\}$ ;
2 $\mathbb{C}(t_1) \leftarrow$ STATIC CLUSTERING $(G(t_1))$ ;
3 for $k \leftarrow 2$ to $r$ do
4     $\mathbb{C}(t_k) \leftarrow \mathbb{C}(t_{k-1})$ ;
5     $\mathbb{C}(t_k) \leftarrow$ REMOVE ELEMENTS $(\mathbb{C}(t_k), G(t_k))$ ;
6     $\mathbb{C}(t_k) \leftarrow$ ADD ELEMENTS $(\mathbb{C}(t_k), G(t_k))$ ;
7     $\mathbb{C}(t_k) \leftarrow$ SPLIT CLUSTERS $(\mathbb{C}(t_k), L)$ ;
8     $\mathbb{C}(t_k) \leftarrow$ RECLUSTER $(\mathbb{C}(t_k), H)$ ;
9     $\mathbb{C}(t_k) \leftarrow$ DISBAND CLUSTERS $(\mathbb{C}(t_k), M)$ ;
```

on empirical analysis. We now give a brief description of each of the main operations in Algorithm 1.

Add/Remove Elements Operation - Since nodes and edges do not necessarily appear at every time point, this operation adds or removes them from our clusters at each time point. When removing a node no longer associated with the current time point, the cluster after this operation contains the following nodes: $C_i(t_k) = C_i(t_{k-1}) \setminus \{v : v \in C_i(t_{k-1}) \land v \notin G(t_k)\}$. When adding a vertex that was not present in the previous time slice to $C_i(t_k)$, the cluster after this operation contains the following nodes: $C_i(t_k) = C_i(t_k) \cup \{v\}$ s. t. $v \in (G(t_k) \setminus G(t_k - 1))$ $\forall C_j(t_k) \in \mathbb{C}(t_k), C_p(C_i(t_k), \{v\}) \geq C_p(C_j(t_k), \{v\})$.

Cluster Splitting Operation - For this operation we want to use the percentage decrease in cluster density to determine whether or not we should attempt to split a given cluster into multiple clusters. The change in density of a cluster $C_i(t_{k-1})$ after it has changed to $C_i(t_k)$ is defined as $\Delta D(C_i(t_k), C_i(t_{k-1})) = (D(C_i(t_{k-1})) - D(C_i(t_k)))/D(C_i(t_k))$. The density of a cluster is $D(C_i) = |E(C_i)|/(0.5 \times |V(C_i)| \times (|V(C_i)| - 1))$. When the Density Percent Decrease of C_i from t_{k-1} to t_k exceeds the *Splitting Threshold* L, $\Delta D(C_i(t_k), C_i(t_{k-1})) > L$ we split $C_i(t_k)$ by running a pass of our static algorithm. The full cluster splitting operation considers every cluster in $\mathbb{C}(t_{k-1})$ for splitting, and returns a modified set of clusters. Figure 3 shows a cluster where $\Delta D(C_{t_2}, C_{t_1}) = 0.28$. If L is less than 0.28, then nodes in this cluster will be reclustered.

Cluster Group Reclustering Operation - This operation works by using a pairwise connectivity measure to find groups of clusters that could benefit from having their component nodes and edges reclustered using a pass of the static algorithm. In order to create groups of clusters needing to be reclustered, we will iterate over our clusters using a modified

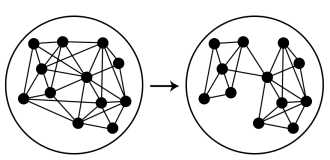


Fig. 3: An example of a cluster C_i at time t_1 (left) and C_i at time t_2 (right) that loses edges from time t_1 to t_2. The cluster density decrease will cause the cluster to be split into two clusters at t_2.

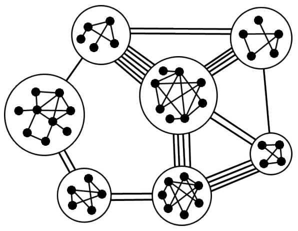


Fig. 4: A set of clusters to be operated on by the Cluster Group Reclustering Operation. Lines between clusters represent *cross-cluster edges*.

Breadth-First traversal. We define a cluster C_i to be a neighbor of another cluster C_j if $E(C_i, C_j) \neq \emptyset$. Thus, we can traverse over incident edge sets $E(C_i, C_j)$ as if they were edges, and clusters as if they were nodes, using the Breadth-First traversal. In order to reduce the number of edges that need to be traversed, we will only traverse over an edge set when the Pairwise Connectivity $R_p(C_i, C_j)$ exceeds the *Reclustering Threshold* H. Here, $R_p(C_i, C_j) = |E(C_i, C_j)|/|E(C_j)|$. Thus, pair connectivity describes how strong the edge connectivity between any two clusters is compared to the internal edge connectivity of one of the clusters.

A traversal containing more than one cluster is considered a group, $\mathbb{Q}$, where $\mathbb{Q} = \mathbb{Q} \cup \{C_j\} : \exists C_i \in \mathbb{Q}, R_p(C_i, C_j) > H$ $\vee R_p(C_j, C_i) > H$ Then we run a pass of the static algorithm to recluster all the nodes and edges in this cluster group. Figure 4 shows an example with seven clusters. The two clusters on the left side of the figure will not need to be reclustered. However, the three on the top right and the two on the bottom right will need to be reclustered as separate subgroups based on this operation. This is the most powerful operation in our dynamic clustering algorithm because its reclustering is very versatile, allowing for merging of clusters, redistributing of nodes within multiple clusters, and splitting of larger clusters into smaller ones.

Cluster Disbanding Operation - Our final operation examines a single cluster, C_i, and its One-to-Many Connectivity, $R_m(C_i, \mathbb{C})$. $R_m(C_i, \mathbb{C}) = (\sum_{j=1, j\neq i}^{n} E(C_i, C_j))/E(C_i)$. One-to-Many connectivity represents the number of outgoing edges from C_i to other clusters versus *in-cluster edges* of C_i. It describes how strongly C_i is connected to other clusters in the graph compared to its own internal connectedness. If a cluster, C_i's one-to-many connectivity is greater than *Disbanding Threshold* M, $C_m(C_i, \mathbb{C}) > M$, then we make the judgment that C_i is too weakly connected in itself compared to its connectivity to other clusters, so we simply "disband" it and redistribute its component nodes and edges among the other clusters to which it is strongly connected. This operation is very useful as a cleanup operation to eliminate weak clusters that may have arisen during the reclustering operation.

B. Multi-resolution Dynamic Alliance Detection Algorithm

Objective: Given a dynamic graph $G(t)$, find groups/alliances that occur frequently during different resolutions of time.

Our dynamic clustering algorithm outputs a set of cluster sets, $\{\mathbb{C}(t_1), \ldots, \mathbb{C}(t_r)\}$, at a single detailed resolution, a cluster per time point. Based on these clusters, our *multi-resolution dynamic alliance detection algorithm* identifies common groups and individuals persistent across different time resolutions in the generalized time hierarchy $\mathbb{T}$. Algorithm 2

Algorithm 2: ALLIANCE DETECTION

```
Input:
  Dynamic Clustering Results: $\{\mathbb{C}(t_1), \ldots, \mathbb{C}(t_r)\}$
  Frequent Itemset Support Threshold: $S$

Output:
  Hierarchy $\mathbb{T}$;

1 Create a set of frequent item sets $\{I_{i,j}\}$ ;
2 for $j \leftarrow 1$ to $r$ do
3     $I_{1,j} \leftarrow \mathbb{C}(t_j)$;
4 for $i \leftarrow 2$ to $r$ do
5     for $j \leftarrow 1$ to $r - i$ do
6         $I_{i,j} \leftarrow$ FrequentItemsetMining$(I_{i-1,j}, I_{i-1,j+1}, S)$ ;
7 Construct a hierachy $(\mathbb{T}, \{I_{i,j}\})$ ;
```

gives us a high level view of our multi-resolution dynamic alliance detection algorithm. Each cluster set, $\mathbb{C}(t_k)$, maps to a leaf node $\mathbb{T}$ (Line 3). In order to identify the persistent alliances and social butterflies for the other nodes in the tree, we apply frequent itemset mining [25] to this problem. In this formulation, each graph vertex is considered an item and each cluster is considered an itemset. The hierarchy is built bottom-up by recursively applying frequent itemset mining to the itemsets of two adjacent nodes and populating their common parent node with the result. In the algorithm, $I_{i,j}$ is the j-th node at the i-th level of $\mathbb{T}$. The process continues until the root node is populated (Line 4 - 6). Thus, each node in $\mathbb{T}$ contains frequent itemsets (groups/alliances or individuals/social butterflys) in its corresponding time period, and frequent itemsets at different level nodes represent groups persistent through different time resolutions.

In our hierarchy, the time window associated with a node increases by one when traversing up the tree to produce a generalized binary hierarchy. In principle, there is no reason that this framework could not be generalized to an m-ary hierarchy, where the time window associated with a node increases by m when traversing up each level of the tree.

IV. Dynamic Framework Evaluation

We now evaluate the different components of our framework. After describing our data sets and evaluation metrics, we begin by evaluating the complete dynamic clustering framework and each of the three operations' contribution to the whole framework. We then explore the parameter setting for our dynamic framework, followed by an efficiency evaluation of our dynamic clustering against running static clustering algorithm for each subgraph repeatedly. Then we evaluate our alliance detection methodology and conduct a case study to show the benefit of our approach for finding both expected and unexpected group patterns.

A. Data Set Explanation

Because clusters are not always known apriori, using synthetic data sets to evaluate a clustering algorithm is meaningful. We have five synthetic data sets (*1-2 Split*, *Merge*, *2-3 Split*, *Disband*, *Multi-purpose*). Each of the first four data sets are designed to highlight one or two of the three cluster updating operations. The *1-2 Split* data set begins with two clusters, and in each successive time step one of the clusters splits into

TABLE I: Our data sets' statistics

	#Node	#Edge	#Time slice
1-2 Split	16	128	3
Merge	16	129	3
2-3 Split	16	149	3
Disband	20	90	3
Multi-purpose	25	268	10
Dolphin-months	303	3,873	6
Dolphin-years	712	117,769	24

two clusters, highlighting the splitting operation. The *Merge* data set is essentially the opposite of the *1-2 Split*, highlighting both reclustering and disbanding operations. The *2-3 Split* data set highlights the reclustering operation. It begins with two clusters in the first time step, and in each successive time step two clusters are reorganized into three clusters. For the *Disband* data set, disbanding operation is the key to the correct clustering. Our last synthetic data set *Multi-purpose* needs all the operations to accurately cluster the graph.

While the synthetic data sets help us determine if our algorithm has the expected behavior, we use real world data sets to understand how well the dynamic clustering framework captures persistent and changing group structures. Specifically, we use a data set about a bottlenose dolphin society in Shark Bay, Australia. Researchers have been monitoring these dolphins for over 25 years [26]. A social network graph generated using the observation data of the dolphin society is structured such that each node represents a dolphin, and each edge, associated with the observation date, represents two dolphins observed together. A data set (*Dolphin-months*) focusing on short term dynamics from June 1995 to November 1995 is used to assess the base resolution dynamic clustering network. We use another data set (*Dolphin-years*) spanning from 1988 to 2011 to evaluate our multi-resolution algorithm. Table I shows all the data sets' statistics.

B. Evaluation Mertics

To validate the accuracy of our results, we compute R-E ratio, precision, recall, and F1.

Result-Expect Cluster Ratio (R-E ratio) - This ratio $|\mathbb{C}(t_k)|/|\mathbb{S}(t_k)|$ compares the count of the a cluster set $\mathbb{C}(t_k)$ to the count of the expected cluster set $\mathbb{S}(t_k)$.

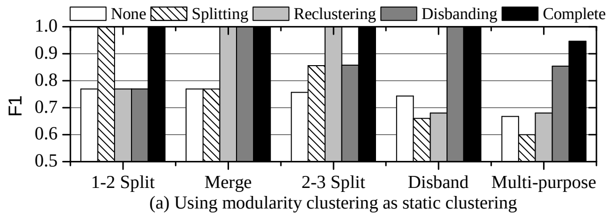
Precision - The precision of a cluster C_i is $P(C_i) = (|V(C_i) \cap V(S_j)|)/|V(C_i)|$, where S_j is the expected cluster which shares the most nodes with C_i. Then the precision of a cluster set $\mathbb{C}$ is $\sum P(C_i)/|\mathbb{C}|$. Further, we average the cluster set precision to get the overall precision of the set of cluster sets generated by our dynamic clustering framework.

Recall - The recall of a cluster C_i is $R(C_i) = (|V(C_i) \cap V(S_j)|)/|V(S_j)|$. We also calculate the cluster set recall $\sum R(C_i)/|\mathbb{C}|$ and the overall recall of the set of cluster sets generated by our dynamic clustering framework.

F1 Score - $F1(C_i) = 2 \times (P(C_i) \times R(C_i))/(P(C_i) + R(C_i))$ integrates precision and recall into a single metric.

C. Dynamic Clustering Accuracy

We test our dynamic algorithm integrating three popular static algorithms - modularity clustering [2] and edge betweenness clustering [1]. We use the following dynamic framework configuration: *Splitting Threshold* $L = 0.15$, *Reclustering Threshold* $H = 0.6$, and *Disbanding Threshold* $M = 1.2$. Figure 5 shows the average F1 score produced by the framework with none of the three dynamic operations, with the splitting operation, with the reclustering operation, with the disbanding operation, and using the complete dynamic framework, respectively. We do not consider the Add/Remove elements operation as a distinct factor like the other operations, since the entire framework depends on it for renovating the subgraph $G(t_{k-1})$ to $G(t_k)$.



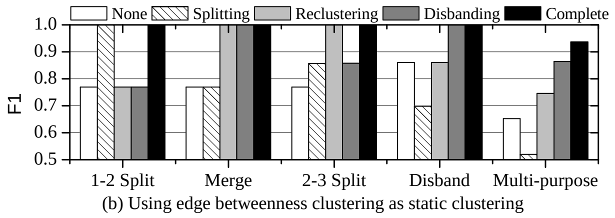


Fig. 5: Clustering performance of the framework with no dynamic operations, a single dynamic operation, or all the dynamic operations on five synthetic data sets.

As expected, the framework with one operation always outperforms the framework with no operation. The splitting operation works much better than the other two operations for the *1-2 Split* data set. Similarly, the reclustering operation performs best for the *2-3 Split* data set, and the disbanding operation is best for the *Disband* data set. Because both reclustering and disbanding operations have the ability to merge clusters, both achieve a 100% F1 score on the *Merge* data set. We observe that for these five data sets using the frameworks with one or all the operations generally outperform the empty framework. The exception is when the splitting operation performs worse than the empty framework for the last two data sets. This results because our framework with only splitting operation tends to produce very small clusters. Such a clustering has a high precision but a comparably low recall, which incurs a low F1. Finally, we observe that the trend of accuracy results is similar for both modularity and betweenness with modularity generally producing slightly better accuracy results.

We also compare the performance of our dynamic framework against applying a static algorithm to all time slices of the dynamic graph repeatedly on the *Multi-purpose* data set. The comparison is shown in Table II. The parameter setting is same as in the above experiments. We observe that the precision results are very comparable, as are the recall results using modularity. For edge-betweenness the recall of the static clustering is much lower. For the other four synthetic data sets, both of the static and dynamic algorithms produce the correct clusterings. While the advantage of our approach is less visible for these synthetic data sets, we will show that for a real world data set that contains less 'clear' clusters,

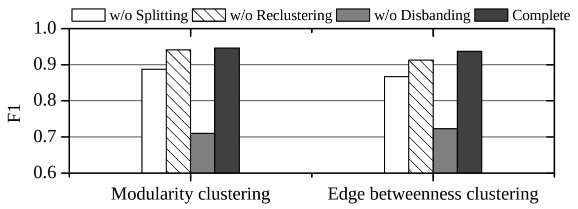


Fig. 6: Clustering performance of the complete framework and the framework with a single operation removed ($L = 0.15$ $M = 1.2$ $H = 0.6$).

our dynamic framework has a higher accuracy and is more efficient.

For the *dolphin-months* data set, we apply our dynamic clustering on a monthly time scale since all dolphins are not observed every day. To evaluate our results, we had a domain expert who has been studying the Shark Bay dolphin population for over 20 years look at the clusters produced by four algorithms, static clustering at each time point using edge betweenness (SE), static clustering at each time point using modularity (SM), dynamic clustering using edge betweenness (DE) and dynamic clustering using modularity (DM). She rank ordered them as follows: DE, DM, SM, SE. She indicated that DE and DM were definitely better. SM was reasonable, but SE was not a good clustering.

D. Lesion Study

Figure 6 illustrates each operation's contribution to the overall framework clustering accuracy on the *Multi-purpose* data set. The first three bars represent the average F1 score produced by the framework with each of the three operations removed, respectively. The fourth bar represents the average F1 score of the complete framework (It is also shown in Figure 5. It is included here for comparison purpose). The results show that each operation makes its own contribution to the framework: the F1 score of the framework with one operation removed is always less than that of the complete framework.

E. Parameter Setting

Our dynamic framework involves three parameters: *Splitting Threshold* L for splitting operations, *Reclustering Threshold* H for reclustering operations, and *Disbanding Threshold* M for disbanding operations. To better understand the effect of the parameter setting's on the dynamic framework clustering results, we run the dynamic framework on the *Multi-purpose* data set with different combinations of parameter setting. The default setting is $L = 0.15,\ H = 0.6,\ M = 1.2$. Figure 7 shows the F1 score comparison when one parameter is varied and the others are stable. In general, we find that modularity is a little less sensitive to parameter selection than edge betweenness.

F. Efficiency Evaluation

Since our dynamic clustering algorithm does not re-cluster all the nodes at each time step, while the static clustering algorithm does, we also present the execution time and the number of nodes that are re-clustered (participate in at least one of the three dynamic operations) at each time step using our dynamic framework and compare them to the time needed to run the static algorithm. We report results for *Dolphin-months* data set in Table III. As our dynamic clustering framework actually runs static clustering for the first time slice, we only show the results starting from the second time slice. When edge betweenness clustering serves as the static clustering algorithm, the run time for the dynamic clustering algorithm is approximately one tenth that of the static algorithm. The large difference in run time can be attributed to the cost of calculating the shortest paths for the betweenness measure [1]. When modularity clustering serves as the static clustering algorithm, the run time for the dynamic clustering algorithm is approximately one third that of the static algorithm. The difference can be attributed to the cost of eigenvector calculation [2]. Given that our dynamic clustering framework produces no worse (or even better) clusterings, as shown in Table II, the dynamic framework clearly outperforms running the static algorithm repeatedly.

TABLE II: Performance comparison between dynamic framework and running a static algorithm at each time slice

		Modularity		Edge Betweenness	
		Dynamic	Static	Dynamic	Static
Multi-purpose	Precision	94.4%	91.2%	92.8%	97.2%
	Recall	93.8%	94.6%	94.7%	66.1%

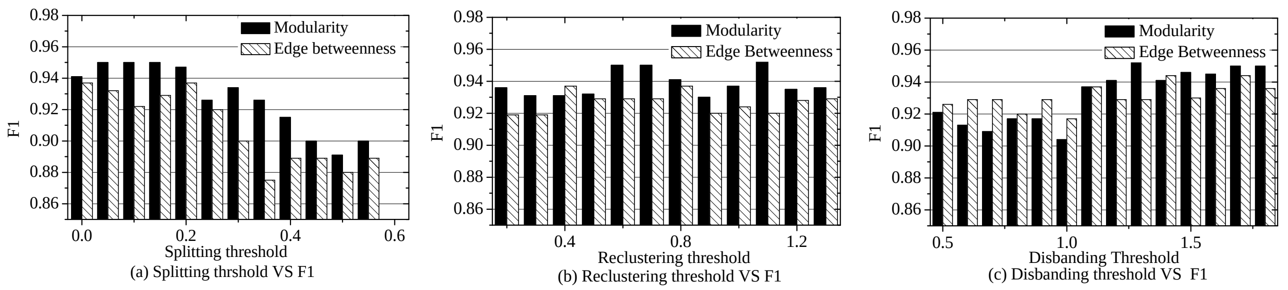
TABLE III: Efficiency evaluation between dynamic and static clustering on the *Dolphin-months* data set ($L = 0.15$ $M = 1.2$ $H = 0.6$). Total node = the total number of nodes in a time slice, Dynamic node = the number of nodes re-clustered using dynamic clustering, Static node = the number of nodes re-clustered using static clustering, Dynamic time = dynamic clustering run time(ms), Static time = static clustering run time (ms).

		Jul	Aug	Sept	Oct	Nov
Total node		139	58	108	123	141
Modularity	Static node	139	58	108	123	141
	Dynamic node	64	34	51	92	103
	Static time	6,966	905	3,436	6,168	6,969
	Dynamic time	956	456	186	2716	2,290
Edge betweenness	Static node	139	58	108	123	141
	Dynamic node	96	34	52	104	103
	Static time	28,949	156	2,150	33,448	24,462
	Dynamic time	2,017	132	309	2,188	1,739

G. Alliance Detection Evaluation

We evaluate alliance detection results using three alliance trees $\mathbb{T}$ for *Dolphin-years* data set, where a different alliance tree is created for each season of data (Biologists studying dolphins divide the calendar into three seasons). We accomplish this by showing the resulting multi-resolution clusters to our domain expert and asking her to identify the clusters that had at least one incorrect member based on her understanding of the dolphin community structure. The results of our analysis are shown in Table IV. For this analysis, we chose modularity as our static algorithm. Our domain expert indicated that the multi-resolution clustering was much more informative than the single resolution. This is because dolphins change groups frequently and the detailed temporal changes are not indicative of group alliance structures. Over the course of a dolphin's life, groups change. Without multi-resolution analysis, capturing these changing group structures that span different time resolutions is a more difficult task for biologists.

Even though the focus of this paper is not the visual analytic component of the framework, we pause to mention that when our domain expert used our visualization tool to explore the dolphin data set, she found supporting evidence



(a) Splitting thrshold VS F1 (b) Reclustering threshold VS F1 (c) Disbanding threshold VS F1

Fig. 7: Dynamic clustering accuracy on the *Multi-purpose* data set. (a) *Splitting Threshold* vs. F1 ($H = 0.6$ $M = 1.2$). (b) *Reclustering Threshold* vs. F1 ($L = 0.15$ $M = 1.2$). (c) *Disbanding Threshold* vs. F1 ($L = 0.15$ $H = 0.6$).

for a behavior members of her lab were aware of, but had not sufficiently explored - the changing social groups of juvenile males. While biologists realize that juvenile males switch between groups when they are trying to establish an alliance, the hierarchical multi-resolution visualization gave insight into how many groups the juvenile males joined and how often they switched groups. Using this visual analytic tool provided our domain expert with potential directions for new research.

V. Conclusions and Future Directions

This paper presents a holistic approach for identifying evolving groups and multi-resolution alliances. Providing scientists with a platform for comparing clustering results, computing multi-resolution groups and social butterflies, and visualizing the varying structures is important for a better understanding of complex time varying networks. Promising future directions include: investigating overlapping communities with this framework, considering semantic based dynamic clustering algorithms and improving the clustering performance.

Acknowledgements

This work was supported in part by the National Science Foundation Grant Nbrs. 0941487 and 0937070, and the Office of Naval Research Grant Nbr. 10230702.

References

[1] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[2] M. E. J. Newman, "Modularity and community structure in networks," *Proceedings of the National Academy of Sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[3] A. Banerjee, S. Basu, and S. Merugu, "Multi-way clustering on relation graphs," in *SDM*, 2006, pp. 145–156.

[4] B. Long, X. Wu, Z. M. Zhang, and P. S. Yu, "Unsupervised learning on k-partite graphs," in *KDD*, 2006, pp. 317–326.

[5] G. Palla, I. Derenyi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, vol. 435, p. 814, 2005.

TABLE IV: Alliance experiment results for the *dolphin-large* data set.

	Season1	Season2	Season3
# of Group Alliance	29	137	60
# of Single Alliance	94	349	114
# of Tree Levels	4	6	4
# of Tree Nodes	10	21	10
Group Accuracy	100%	96%	100%

[6] H. Shen, X. Cheng, K. Cai, and M.-B. Hu, "Detect overlapping and hierarchical community structure in networks," *Physica A: Statistical Mechanics and its Applications*, vol. 388, no. 8, pp. 1706–1712, 2009.

[7] Y. Zhou, H. Cheng, and J. X. Yu, "Graph clustering based on structural/attribute similarities," in *VLDB*, 2009, pp. 718–729.

[8] C. Tantipathananandh, T. Berger-Wolf, and D. Kempe, "A framework for community identification in dynamic social networks," in *KDD*, 2007, pp. 717–726.

[9] G. Palla, A. Barabasi, and T. Vicsek, "Quantifying social group evolution," *Nature*, vol. 446, no. 7136, pp. 664–667, 2007.

[10] C. Aggarwal and P. Yu, "Online analysis of community evolution in data streams," in *SDM*, 2005, pp. 56–67.

[11] R. Görke, T. Hartmann, and D. Wagner, "Dynamic graph clustering using minimum-cut trees," *Algorithms and Data Structures*, pp. 339–350, 2009.

[12] J. Hopcroft, O. Khan, B. Kulis, and B. Selman, "Tracking evolving communities in large linked networks," *Proceedings of the National Academy of Sciences*, vol. 101, no. Supplemental 1, pp. 5249–5253, 2004.

[13] J. Sun, C. Faloutsos, S. Papadimitriou, and P. Yu, "Graphscope: parameter-free mining of large time-evolving graphs," in *KDD*, 2007, pp. 687–696.

[14] T. Warren Liao, "Clustering of time series data - a survey," *Pattern Recognition*, vol. 38, no. 11, pp. 1857–1874, 2005.

[15] C. Antunes and A. Oliveira, "Temporal data mining: An overview," in *KDD Workshop on Temporal Data Mining*, 2001, pp. 1–13.

[16] S. Das, A. Abraham, and A. Konar, "Automatic clustering using an improved differential evolution algorithm," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 38, no. 1, pp. 218–237, 2008.

[17] D. Chakrabarti, R. Kumar, and A. Tomkins, "Evolutionary clustering," in *KDD*, 2006, pp. 554–560.

[18] J. Leskovec, D. Chakrabarti, J. Kleinberg, and C. Faloutsos, "Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication," in *PKDD*, 2005, pp. 133–145.

[19] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan, "Group formation in large social networks: membership, growth, and evolution," in *KDD*, 2006, pp. 44–54.

[20] P. Bródka, S. Saganowski, and P. Kazienko, "Ged: the method for group evolution discovery in social networks," *Social Network Analysis and Mining*, pp. 1–14, 2012.

[21] S. Asur, S. Parthasarathy, and D. Ucar, "An event-based framework for characterizing the evolutionary behavior of interaction graphs," in *KDD*, 2007, pp. 913–921.

[22] L. Friedland and D. Jensen, "Finding tribes: identifying close-knit individuals from employment patterns," in *KDD*, 2007, pp. 290–299.

[23] H. Habiba, Y. Yu, T. Berger-Wolf, and J. Saia, "Finding spread blockers in dynamic networks," in *SNAKDD*, pp. 55–76.

[24] H. Sharara, L. Singh, L. Getoor, and J. Mann, "Finding prominent actors in dynamic affiliation networks," *Human Journal*, vol. 1, no. 1, pp. 1–14, 2012.

[25] J. Han and M. Kamber, *Data mining: concepts and techniques*, 2006.

[26] "Shark bay dolphin project," http://www.monkeymiadolphins.org.