Chapter 1
EgoClustering: Overlapping Community Detection via Merged Friendship-Groups

Bradley S. Rees and Keith B. Gallagher

Abstract There has been considerable interest in identifying communities within large collections of social networking data. Existing algorithms will classify an actor (node) into a single group, ignoring the fact that in real-world situations people tend to belong concurrently to multiple (overlapping) groups. Our work focuses on the ability to find overlapping communities. We use egonets to form friendship-groups. A friendship-group is a localized community as seen from an individual's perspective that allows an actor to belong to multiple communities. Our algorithm finds overlapping communities and identifies key members that bind communities together. Additionally, we will highlight the parallel feature of the algorithm as a means of improving runtime performance, and the ability of the algorithm to run within a database and not be constrained by system memory.

1.1 Introduction

An escalation in the number of Community Detection algorithms [2,9,11–14,22,24, 26, 34–36, 38, 40, 45, 46] has occurred in recent years. The focus of the algorithms shifted away from the classical clustering principles of grouping nodes based upon some type of shared attribute [20,36], to one where the relationships and interactions between individuals are emphasized. The shift has caused algorithms to view the data as a graph and focus on exploiting (detecting) the "small-world effect" [44] found in social networks – the phenomena that a small path length separates any two randomly selected nodes – and on detecting the clustering property of social networks in which the density of the edges is higher within the group than between the groups [2, 13, 14, 22, 24, 26, 34–36, 38, 40, 45].

B.S. Rees (✉) · K.B. Gallagher
Department of Computer Science, Florida Institute of Technology, Melbourne, FL, USA
e-mail: brees2011@my.fit.edu; kgallagher@fit.edu

Moody and White [33] reasoned that communities are held together by the presence of multiple independent paths between members. Extrapolating from the goal of discovering clusters, where internal edge density is maximized, it follows that the identification of cliques [15, 26, 38] {k-cliques, k-clans, or k-cores, where k is the number of nodes comprising the group} would be a viable approach; the density is maximal within those structures. However, given that a five-clique, for example, contains a number of overlapping four-cliques, each of which is a community in its own right [15], presents the question of whether the algorithm is really revealing communities or just doing pattern matching.

Other approaches have focused on centrality [17] to identify key nodes or edges, and follow a hierarchical clustering approach to recursively extract clusters [13, 22, 26]. While centrality is a powerful and useful idea for identifying key (central) actors in a network, many of the centrality approaches require that the centrality measurement be recalculated after each graph edit, causing the algorithms to be highly inefficient [13, 35, 36].

In this paper, which is an expanded version of the one we presented at ASONAM 2010 [41], we present a radically different approach to group detection that finds communities based on the collective viewpoint of individuals. The notion postulated is that each node in the network knows, by way of its egonet [16, 18], who is in its *Friendship-Groups*. We use the term friendship-group to represent the small clusters, extracted from egonets, containing the central node and communal neighbors. Therefore, by calculating the aggregation of each individual's friendship-groups, we find overlapping communities, in a process we term *EgoClustering*. Additionally, the algorithm is designed to be highly parallelizable as a means of improving runtime, and able to operate within a database and therefore not constrained by system memory.

The contributions of this paper are:

1. A precise mathematical formulation of a Friendship-Group
2. A full fledged implementation of the EgoClustering algorithm
3. An algorithm producing communities with maximal size by allowing for overlap
4. A more intuitive approach to community detection
5. An algorithm that can be run on disk-based data
6. An Algorithm that can be easily parallelized.

1.1.1 Terminology

Social Network Analysis derives from the social sciences with its own taxonomy and argot, while graph theory derives from mathematics with a different taxonomy. In graph theory [6], the terms vertex and edge are used to describe a graph, while social networking [10, 43] uses node or actor and edge, link, or arc to describe a graph. For the purpose of this paper, the terms are used interchangeably, with a slight preference toward nodes and edges.

A graph is defined as $G = \{V, E\}$ where V is a set of vertices (nodes) and E is a set of edges, represented by unordered pairs of vertices, called the *start node* and *end node*. The edge set defines connections between pairs of vertices. An optional weighting can be assigned to the pair. If the pairs are ordered, the graph is directed. A *path* is an ordered sequence of edges in the graph where the end node of an edge is the start node of the next in the sequence. Any two nodes on a path are *connected*. The shortest path between to nodes is one with the least number of edges. If there is no path between two nodes, they are *disconnected*.

The neighbors of a vertex, v, is defined as the set of vertexes connected by way of an edge to vertex v, or $\mathrm{N}(v) = \{U\}$ where $v \in V$ and $\forall \mathrm{u} \in \mathrm{U}\ \exists\ \mathrm{edge(v,u)} \in \mathrm{E}$. The degree of a vertex, $\delta(v)$, is the number of edges incident to that vertex. In the case where the graph contains no loops (edges that have the same starting and ending vertex) the degree of a vertex is also equal to the number of neighbors, $\delta(v) = |\mathrm{N}(v)|$.

The density of a graph, or subgraph, is the measure of the number of edges in the graph, over the maximum number of possible edges. A value of 1 indicates that all possible edges are present, while a value of 0 indicates the absence of any edges. The most edges a node can have is $(n-1)$; the maximum number of edges possible in an undirected graph is $\frac{n(n-1)}{2}$. Density can then be defined as: $d(n) = \frac{2m}{n(n-1)}$, where n is the number of nodes and m is the number of edges. A sparse graph is one where the number of edges is close to the number of nodes, and a dense graph is one where the density measurement approaches, or is equal to, 1. There is no agreed upon threshold between a sparse graph and a dense graph.

Centrality [17] is a measure of how important, or central, a node is in relation to the whole graph. The *betweenness centrality* of a node, n, is number of paths that contain n in the *all-pairs-shortest-path* set of the graph G. Betweenness centrality can also be obtained for edges [36].

The term egonet [10, 16, 18] derives from egocentric network. An egonet is an induced subgraph consisting of a central node, (the *ego-node*), its neighbors, and all edges among the neighbors. The individual's viewpoint reduces the network under consideration to just those vertices adjacent to the central "ego" node and any edges between those nodes.

Given a graph G, the egonet on a node, n, is:
ego(n) = the subgraph H of G where

$$\begin{aligned}
&V(H) = \{v, N(v)\} \\
&E(H) = \\
&\quad \forall (n_1, n_2) \in V(H) \text{ if} \\
&\quad \exists\ e(n_1,\ n_2) \in E(G) \text{ then} \\
&\quad \exists\ e(n_1, n_2) \in E(H)
\end{aligned}$$

A dyad is two nodes joined by an edge. A triad is three nodes connected by a minimum of two edges and a maximum of three edges.

All graphs in this work are considered to be "sparse", unweighted, undirected, and containing no loops. For this work, we define sparse as being graphs with density less than 0.4.

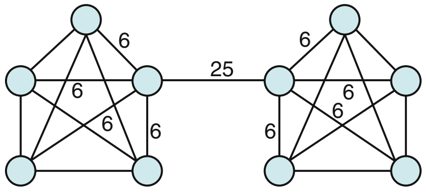

Fig. 1.1 Edge betweenness centrality scores

1.2 Related Work

One of the more prevalent algorithms comes from work by Girvin and Newman [22,36] (GN). The GN algorithm follows a divisive hierarchical method, which iteratively removes edges with the highest edge-betweenness centrality score. This is based on the principle that *between community* edges have higher centrality than *within community* edges, as shown in Fig. 1.1.

The GN algorithm recognized that the centrality score must be recalculated after each edge removal. However, the recalculating of centrality causes the algorithm to have high computational demands, running in $O(n3)$ to $O(n4)$ time on sparse graphs. Newman addressed the performance factor in a subsequent paper [35] by developing an agglomerative method that reduced runtime to $O(n2)$.

Hierarchical clustering approaches, divisive or agglomerative, present some problems. As Newman points out [35] "…the GN community structure algorithm always produces some division of vertices into communities, regardless of whether the network has any natural such divisions." Moreover, the "fast-Newman" [35] algorithm suffers from an NP-complete subproblem [46].

The notion of using some form of centrality as the means for determining edge removal was extended by Hwang et al. [34], by the concept of Bridging Centrality. A bridge, in graph theory terms, is an edge whose removal will break the graph into two disconnected subgraphs. Hwang et al. defined Bridging Centrality as the ranked product of betweenness centrality and a bridging coefficient. Informally, the bridging coefficient is the probability of having common neighbors.

Agglomerative methods start with one node per cluster and iteratively joins clusters; divisive methods start with one cluster and iteratively divides. The iterations of both processes can be represented as a *dendrogram*. Selecting different stopping points in those processes will produce different numbers of communities [34, 36]. The challenge is that the decision of where to stop should to be done a priori. The following illustration, Fig. 1.2, shows a dendrogram with three possible cut points (A, B, and C), producing two, four, or six possible clusters, each of which does not necessarily equate to a community [40]. *Modularity* (a probabilistic method) and density have both been used as means of determining the stopping point [26, 36].

Modularity was first introduced by Newman and Girvan [36] as a means of determining when to stop processing within their divisive algorithm. Since then, modularity has become a widely studied community quality measure [7, 8, 37, 42] (non-exhaustive list). More recently, Brandes et. al. [5] published a critique of

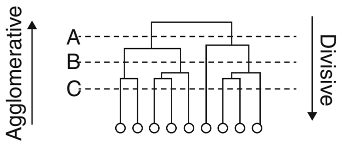

Fig. 1.2 Dendrogram with three possible cuts

modularity and illustrated how finding the optimal modularity value is an NP-complete problem. Modularity can be described as the notion that communities do not occur by random change. The Modularity, denoted Q, is the measure of a cluster against the same cluster in a null (or random) graph. A greater than random probability indicates a good cluster.

These approaches suffer the additional problem that nodes are forced to exist only in a single community. Real-world networks are not so nicely constrained, and contain realistic amounts of overlap between communities [9, 33, 38]. Each person (node) could have a community for family, friends, work, and interest, for example, and community detection algorithms must allow for, and detect, overlapping groups. Forcing a node into a single community and not allowing for overlap could prevent the detection of the true underlying community structures [9, 30, 38].

A number of solutions for finding overlapping communities have been developed [2, 9, 13, 14, 24, 38]. Gregory [24], for example, modified the GN algorithm to highlight overlapping communities by splitting nodes, thus permitting a node to be represented in the graph multiple times, and allowing each instance of the node to clustered into a different community. While the modification does find overlapping communities, it also degrades the algorithm's performance.

Local clustering has been explored in a number of algorithms [1, 8, 30]. This technique, which builds communities independently, does not remove nodes from the graph for subsequent iterations. Overlapping communities can be found using local clustering. Baumes et al. [2, 3] present a unique two-step approach to finding overlapping communities. The first part of the algorithm is called Rank Removal, or *RaRe*, which iteratively removes high ranked nodes, thus breaking the network into disconnected clusters. Baumes et al. discuss the use of PageRank and high degree nodes (degree-centrality) as a means of finding important nodes, however it would seem logical to expand that process to leverage any of the previously discussed community detection approaches. The second step is the truly unique portion of their algorithm, and involves adding nodes that were not part of the cluster and evaluating whether the clusters density increased. This step considers all neighboring nodes, rather than all nodes, as a means of improving performance. Additionally, it is this step that permits the assumption that nodes belong to multiple communities and therefore overlap.

The notion of local-based community construction was also used by Lancichinetti et al. [30] in what they termed as finding the "natural community" of a node. Lancichinetti's algorithm works by randomly selecting a node and iteratively adding neighboring nodes, checking for an increase in "fitness." Fitness is roughly similar to modularity [35] or Radicchi's definition of community [40], and is defined as the measure of edges within a community over the sum of edges within and leaving the community: $f_G = \frac{k_{in}^G}{(k_{in}^G + k_{out}^G)\alpha}$.

The factor, α, is used to control, or limit, community size. However, as Lancichinetti points out, the best results are obtained where $\alpha = 1$. The values of *kin* and *kout* are the degree of edges within the community and leaving the community respectively. Since each community is built independently, and based on the full graph, overlap between the communities can occur.

The notion of a clique (a subgraph with maximal density) being synonymous with a community is not new, and approaches for finding cliques originated as early as the late 1940s [15]. Palla et al. [38] extended the theory of cliques as communities by introducing the definition that a community, specifically a *k-clique-community*, is a union of all k-cliques that can be reached via adjacent k-cliques. The process works by rolling, or *percolating*, a k-clique over the network to find other k-cliques that share $k - 1$ nodes. The percolating [11] is performed by moving the selection of one node within the k-clique to an unselected neighbor node that also form a k-clique. Since only one node is selected each time, the subsequent k-clique must share exactly $k - 1$ nodes.

1.3 Our Approach

1.3.1 Defining Community

There is no formal, or conventional, definition of social community [12] beyond "a collection of individuals linked by a common interest" [32]. Rather than trying to define, or redefining community, we turn instead to work by Moody and White [33], who focused on defining four characteristics that bind a community together, referred to as "structural cohesion." One definition of interest from Moody and White is that community cohesion is tied to the number of independent paths between members. That definition is supported by the qualitative observations [40] that communities have greater internal edge density than external, inter-community, density. Consider the graph in Fig. 1.3a; it contains two obvious communities with a single edge between them. As the number of links between communities increases, the ability of clustering algorithms to find distinct communities degrades [22]. Increasing the number of edges between the two communities, Fig. 1.3a, b poses the question: Are there still two communities, have the two merged into one, or are there now three communities?

A second definition from Moody and White is that the removal of one member (node) should not cause the community to collapse. Therefore, for this version of the algorithm, a dyad is not a community; likewise a node of degree 1 cannot be part of a community. However, nodes of degree 1 could be easily subsumed into its neighbor – future version of the algorithm.

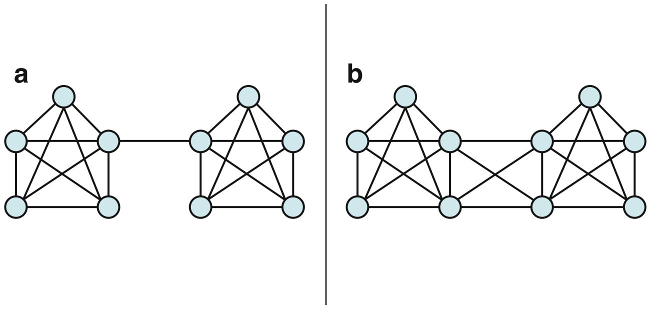

Fig. 1.3 Example of increased edges between communities

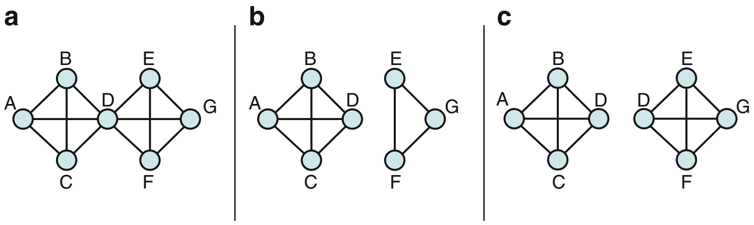

Fig. 1.4 Need to allow overlap

1.3.2 The Need to Allow Overlapping

A key feature [11] of most real-world communities from social networks is that they overlap [9, 13, 24, 30], allowing a single node to belong to multiple communities. The notion should be intuitive, and empirically evident [9, 19] that individuals can belong to multiple simultaneous groups, for example families, social circles, and work communities. Moreover, in hierarchical clustering, as several have pointed out [9, 30, 38, 39], the assignment of a node to a single community can cause the remaining communities to fall apart, thus preventing the detection, or discovery, of the true social structures. A simple proof to this statement can be seen in the following example.

We ran two popular community detection algorithms on the simple and very small graph – for illustration purposes – shown in Fig. 1.4a. In this case, the *FastModularity* algorithm of Clauset and Newman [7], and the modified GN [22] algorithm, called "A Fast Algorithm", from Radicchi et al. [40]. Each of the algorithms detected the same two communities shown in Fig. 1.4b.

If we examine the smaller community, {E, G, F}, Fig. 1.4b, independent of the other communities and under the premise that all nodes and edges not within that community are available for consideration in the community, we can then evaluate the effect of adding each neighbor node into the community. In this case the inclusion of node D within the smaller community increases the modularity score, and therefore uncovers the true community. Both local clustering and our

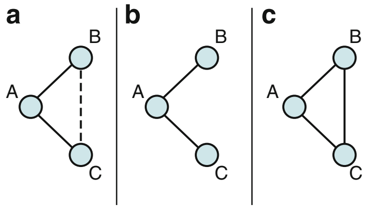

Fig. 1.5 Triangles and the Forbidden triad

EgoClustering algorithms produce the result of Fig. 1.4c, which we believe are the valid communities of the graph.

For the purpose of this study, we are interested in finding all communities within a social network, and not simply on partitioning nodes into clusters. Therefore we make the statement that detected communities can only be guaranteed to be maximal if overlap is allowed, and by not allowing overlap, erroneous results can be obtained; moreover all overlapping nodes must be found.

1.3.3 Triangles

When examining undirected, unweighted, and unlabeled graphs, a few assumptions need to be made: (1) That there is some form of *homophily* (common interest) that binds communities together; (2) that each edge represents the same level of relationship strength; and, (3) that there is an equal amount of reciprocity in each edge. With those assumptions in mind, we can look at triads and their relationship to communities.

Consider a triad comprised of the three nodes {A, B, C}, Fig. 1.5a. If there is a tie between A and B, and A and C, the probability that B and C are linked is so much greater than random that Granovetter [23, 27] deemed the absence of such a link as the "Forbidden" triad. The presence of a triad indicates that there is a strong tie [23] between the nodes and therefore some type of shared interest, which could be called a community.

For the purpose of this work, we are considering the absence of a link between node B and C, Fig. 1.5b, to be an indication that B and C are not similar and therefore, initially, not within the same community. Conversely, the presence of a tie between B and C, Fig. 1.5c, is an indication of a community.

1.3.4 Friendship-Groups

With the rudimentary definition of community, the need to allow overlap, and the value of triad defined, we can now define the basic building block of our algorithm, the *Friendship-Group*. Consider the graph shown in Fig. 1.6a, an egonet build around node A.

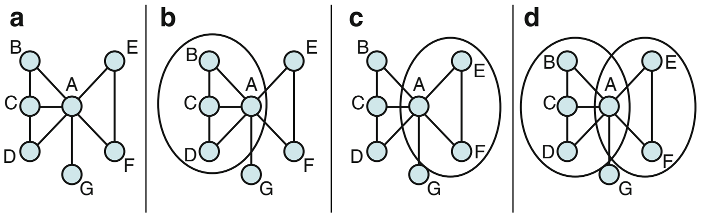

Fig. 1.6 Friendship-Groups

Node A has a strong connection to nodes B and C, since nodes B and C are connected. Additionally, node A has a strong connection to nodes C and D, which are also connected. Without additional information we can infer that node A, B, C, and D form a community, Fig. 1.6b. At the same time, node A has a strong connection to nodes E and F, due to the connection between E and F. Since nodes are allowed to belong to multiple communities, we conclude that nodes A, E, and F form a community as shown in Fig. 1.6c. The connection between node A and G fits the definition of a dyad, which we have previously defined as not constituting a community.

We define a Friendship-Group to be the local view of communities within an egonet from the perspective of the ego node. Or, an induced subgraph extracted from an egonet, adhering to the same constraints mentioned above for a community; multiple paths and no dyads or single nodes. We make the distinction between communities and friendship-group since the friendship-group is myopic view of the egonet, and one or more friendships-groups can be combined to form a community. The egonet in Fig. 1.6 contains two friendship-groups as shown in Fig. 1.6d.

1.3.5 Algorithm

The algorithm executes in two phases; the first phase is the detection of friendship-groups, the second phase comprises the aggregation of friendship-groups into communities.

In phase 1, the algorithm iterates through every vertex in the graph and derives the egonet for that vertex. From that derived egonet, friendship-groups are extracted. The process for finding friendship-groups from the egonet is performed by first removing the central, or ego, node, since it is known to exist in multiple friendship-groups. By removing the ego vertex, the graph breaks into multiple connected components, each of which can be easily found. The egocentric vertex is then added back to each found component to form the friendship-groups.

For example, given the following simple network, Fig. 1.7a, the egonet for vertex D would be just those vertices connected to D, or B, C, E, and F, as shown in Fig. 1.7b.

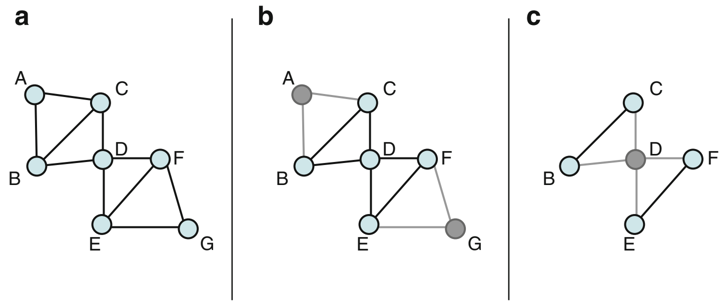

Fig. 1.7 Detecting Friendship-Groups

```
Pseudo code:

1. For each node $\forall n \in \{V\}$
   • Get egonet of n: H = ego(n)
   • Find Friendship-Groups:
      – Remove n from the egonet
      – Find the connected components of the remaining subgraph
      – Add n to each component
2. Merge and Reduce Sets
   • Remove proper subsets
   • Merge "close" matches
   • Repeat until no more merges can be performed
```

From the point-of-view of vertex D, nodes B and C are friends and E and F are friends. The removal of D, grayed out in Fig. 1.7c, creates two distinct components. That yields two friendship-groups, with the ego vertex added back in, of {B, C, D} and {D, E, F}. That process is repeated for every vertex in the network. The result of that first phase is a collection of friendship-groups, from an egocentric point-of-view.

The next step, phase 2, is to merge all the friendship-groups into communities. That process is done by first merging all exact matches, groups that are either complete or proper subsets of other groups. The final step is to merge groups that are "relatively close"; in this case, groups that match all but one item from the smaller group. Given two sets, Sl and Ss, where Sl is larger than, or equal to, Ss, then the sets are merged (union) if the size of the intersection is equal to one less than the size of the smaller set: $S_l \bigcap S_s| = |S_s| - 1$; i.e., the size of the set difference is 1. This step compensates for egonets not having a complete picture of the community, and allows communities of different sizes to be compared. Continuing the example from above, Fig. 1.7a, group {A, B, C}, obtained from egonet centered on node A, would merge with group {B, C, D}, from egonet centered on B and/or C, to form {A, B, C, D}. Notice that even though A and D are not directly connected, they are in the same community.

1.3.6 Performance

The runtime performance of the algorithm is greatly influenced by the density of the graph being analyzed. Consequently, we will compute performance for the boundary conditions, density $= 0$ and density $= 1$, and for the anticipated runtime when applied to sparse graphs, typical of social networks. For performance definition, we use n to represent the number of nodes, m to represent the number of edges, δ to represent the average degree of a node, and s to represent the number of friendship-groups sets identified. We will delay reducing any equation until after the base equation has been defined. Lastly, as with any algorithm, the method of implementation can affect performance. Here we assume that the graph is stored either as an adjacency matrix, or spared edge list.

The first phase of the algorithm comprises the identification of friendship-groups within derived egonets. The process of identifying the egonet can be done in constant time, since the base graph does not have to be modified. The process only needs to identify the neighbors of the selected node. If the data is stored in an edge matrix, then the neighbors are specified in the row corresponding to the ego-node. The complexity of iterating over each node is captured in the following description.

The process of finding the egonet friendship-groups, or disjoint connected components, can be done using the classic union-find algorithm, in $O(log(n))$ time. The process of finding the friendship groups requires that the approximately δ incident nodes of the egonode be compared against the δ incident nodes of each neighbor of the egonode, gives $O(\delta^2)$. Since the process of finding friendship-groups is done for each node in the network, the runtime for the first phase is $O(n\delta^2)$.

The second step is filtering and merging, which can be accomplished with a modified merge-sort algorithm. A traditional merge-sort runs in $O(slog(s))$, however the merging process in this case produces a new set (partial community) that needs to be reexamined and compared to the remaining set. That modification increases runtime to $O(s^2log(s))$.

Lower Boundary: When density equals 0 (i.e. there are no edges), all nodes are disconnected. Therefore, the average degree of a node is 0 and $\delta = 0$. That reduces the first phase to $O(n)$. As detected friendship-groups consist of only the ego-node, the number of sets is equal to the number of nodes, $s = n$. Additionally, since we know that each set is unique, no merges will occur and the algorithm will not need to reexamine any merged sets. This brings the runtime of the second phase down to $O(nlog(n))$. The total runtime is then $O(n^2log(n))$. Since we know that single node sets cannot merge during the second phase, we could programmatically have removed those sets and not done the all-pair comparison, further reducing runtime to: $O(n)$.

Upper Boundary: When density equals 1 (i.e. every possible edge exists), then the graph is one large clique. The average degree of every node is $\delta = (n - 1)$, which we reduce to just $\delta = n$. This causes the first phase to have a runtime of $O(n^3)$.

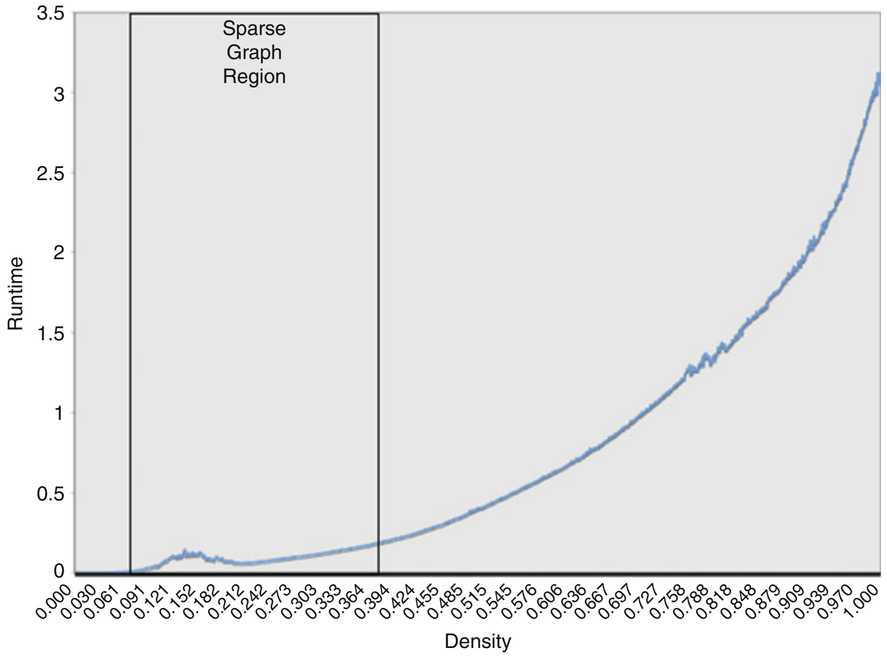

Fig. 1.8 Runtime

For the second phase, each node will have detected only a single friendship-group, $s = n$. However, all friendship-groups will be the same, hence the first pass will merge all sets down to a single set. This reduces the runtime of the second phase to $O(n)$. The total runtime is thus: $O(n + n^3)$.

Anticipated Runtime: For sparse graphs where the number of edges scales linearly with the number of nodes, Hwang et al. [26] points out that the average degree is approximately $logn$, which we will use for the anticipated engonet size of a sparse graph, $\delta = log(n)$. The first phase becomes: $O(nlog^2(n))$. For the second phase, we assume that the maximum number of sets per friendship-groups is the same as the average degree, or $s = log(n)$. Runtime for phase 2 then becomes: $O(n^2 log(n))$, and the total runtime is: $O(n(log^2(n)) + n^2 log(n))$.

The runtime performance of the algorithm can now be expressed as:

$$O(n) < O(n(log^2(n)) + n^2 log(n)) < O(n^3)$$

Figure 1.8 depicts the performance of running the algorithm over a graph with 100 nodes and increasing the density from 0 to 1. The inserted box represents the targets sparse area.

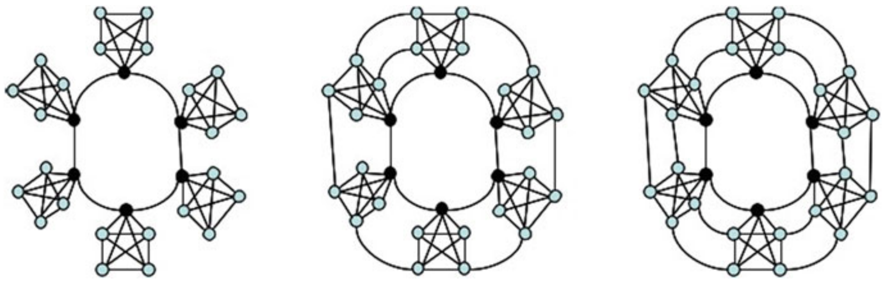

Fig. 1.9 Caveman graph

1.3.7 Improving Performance

The runtime performance shown above is not an improvement, and in some cases inferior, to existing algorithms. Conversely, our algorithm is designed to operate in a parallel fashion as a means of improving performance and scalability.

The initial phase of the algorithm is the identification of friendship groups by iterating over all nodes in the graph. Friendship groups are found for each node, independent of the other nodes, and therefore can be performed in a parallel fashion. The second phase is an all-pair comparison, where a selected set (friendship-group or community) is compared with all others to determine if the set warrants merging, deletion, or retention. As each comparison is acted independently from the previous examination, these processes can also be performed in parallel.

Disk-Resided Processing

One advantage of the algorithm is that it does not need to operate on the graph as a whole; this is true for sparse graphs that are the focus of this work. The algorithm can extract egonets from database resident adjacency matrixes and save detected friendship-groups as sets within a caches database table for the merge and reduce phase. This allows the algorithm to operate against very large graphs that would be too large to fit within available memory.

1.4 Application

1.4.1 Caveman

The algorithm was first applied against a Caveman graph, Fig. 1.9, a term coined by Watts and Strogatz [44] for a network containing a number of fully-connected clusters (cliques) or “caves.” The number of connections between the caves is increased to determine at which point the algorithm stops identifying the core cave groups.

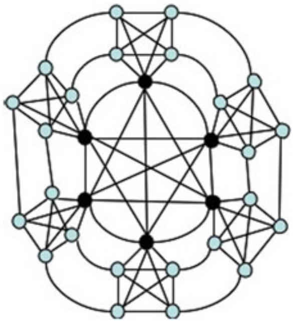
Fig. 1.10 Fully connected Caveman graph

In the case of the three examples shown in Fig. 1.9, the algorithm found the groups with no errors. Although this was a simple case and the links were not really added at random – additional links did not form any new triads and thus no additional groups were detected. When additional links were added linking all the center nodes, Fig. 1.10, the algorithm then detected the six original groups plus a new overlapping community formed by the center nodes. The introduction of a new community is an indication that linkages between communities cannot be added without regard for the implication of the newly formed relationships.

1.4.2 Zachary

The Zachary [47] Karate Club dataset is well studied, and widely utilized as a test bed for many community detection algorithms [13, 22, 24, 34–36, 45]. Zachary observed the social interactions of members of his karate club over a period of 2 years. By chance a dispute broke out between two members that caused the club to split into two smaller groups.

When our algorithm was applied to the Zachary dataset, four communities were found. The following graph, Fig. 1.11, illustrates the discovered networks as well as highlighting the two clubs formed after the split, group 1 is shown by the circles hexagons and group 2 shown by squares and triangles.

Cluster A: [1, 17, 7, 11, 6, 5]
Cluster B: [13, 33, 1, 4, 14, 3, 22, 20, 2, 9, 18, 8]
Cluster C: [25, 32, 26]
Cluster D: [29, 33, 1, 21, 3, 31, 9, 15, 34, 28, 24, 30, 16, 27, 32, 19, 23]
Not a member of a community: 10, 12

At first glance, it might appear that our algorithm was in error when it detected four communities in contrast with what the Zachary states as the final outcome. However, the focus of the Zachary paper was on group fission and not on communities, or overlapping communities, within the group. Additionally, the Zachary paper presented a method for creating edge weights based on an aggregation of the number

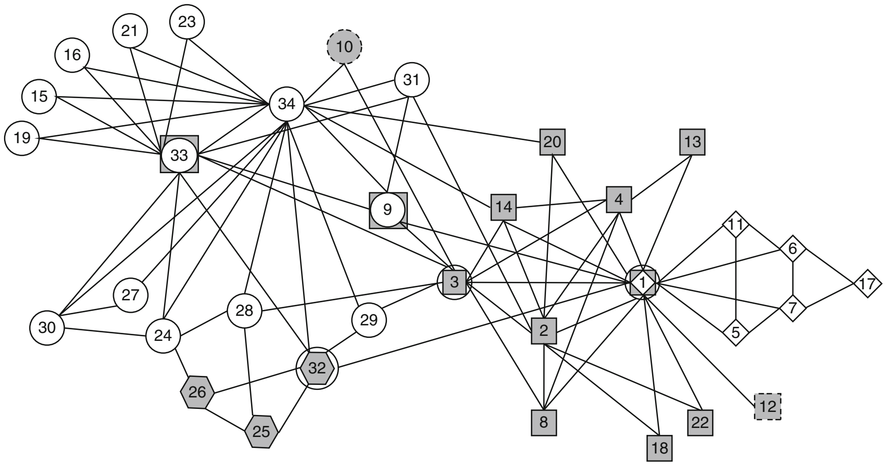

Fig. 1.11 The Zachary Karate club dataset

of different social interaction domains at individuals attended together. Each of these domains has the possibility of defining a community.

Hierarchical clustering allows for the algorithm to be stopped at various points, producing from 1 to n clusters. Since the anticipated results were two clusters, that is the stopping point of most benchmarks against the Zachary dataset. The GN [22] algorithm, for example, identifies the two communities within the dataset, when programmed to extract only two communities.

The FastModularity algorithm of Clauset and Newman [7], selects a stopping point by optimizing modularity. Their algorithm finds three communities, denoted as circles, squares, and triangles as shown in Fig. 1.12.

As we mentioned in Sect. 1.3.2, a community can only be guaranteed to be maximal – inclusion or removal of one additional node decreases quality of the community – if overlap is allowed. Since the FastModularity algorithm does not allow for overlap, it appears as if one community, denoted as squares, is a collection of left over nodes. The inclusion of node “1” within the square community would increase modularity and density.

As an additional comparison, Donetti and Muñoz [12] presented an overlapping algorithm based on modularity that stops processing when modularity is maximized. Their algorithm finds four clusters and one single node.

If the goal was to simply produce two clusters, then a few additional communities merging would have to occur. Looking at Community A, this community is virtually independent from the rest of the communities, with the overlap occurring solely due to node “1”. When the split in the karate group happened, this group would follow node “1” and community A would merge in with community B. Looking at the dendrogram from the GN [22] and the Donetti [12] papers, the node comprising cluster A and B are merged in the final step. Community C is less independent than A, but only has an overlap with community D at node “32” and would merge in

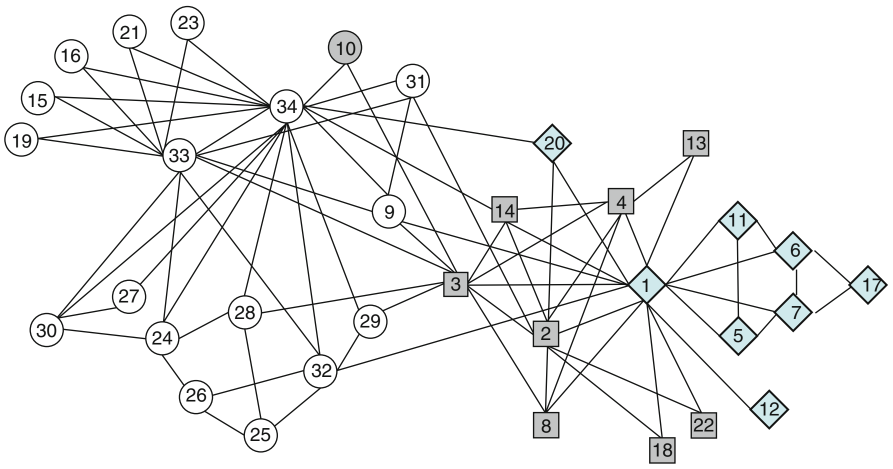

Fig. 1.12 Results using FastModularity

with that cluster. A merger of cluster A with cluster B and cluster C with cluster D would produce two communities, with the only error being the overlapping nodes that appear within both communities.

Yet the purpose of our algorithm was to find overlapping communities and not graph partitioning. In detecting communities, the algorithm also identifies those nodes that form the overlap and act as brokers, or social bridges, between communities. Of particular interest from the karate club are nodes 1, 3, 9, and 33. Those four nodes appear to be the glue that held the groups together. For example, breaking the edge between nodes 3 and 33 and nodes 9 and 1 causes our algorithm to remove the overlap between the two groups. From that we can deduce that any strife within the group affected those four nodes, has the potential to impact the entire karate club.

1.4.3 Other Datasets and Follow-on Work

A number of other datasets were processed by our algorithm and are shown in Table 1.1. However, as the sizes of the graphs being examined grew, so did the complexity of displaying and analyzing the results. The table shows some basic metrics – number of nodes (order), the number of edges (size), the average degree, and the density – on each dataset along with the number of detected communities and the number of nodes not assigned to any community, show in parentheses. Additionally, the number of communities detected from running the FastModularity from Clauset and Newman [7] and the CFinder algorithm of Palla et al. [38] are shown for comparison. For CFinder, the results for $k = 3$ were used. (Each author on his or her respected web sites generously provided source code for each algorithm).

Table 1.1 Additional datasets and number of communities detected

Dataset	Nodes	Edges	Avg. degree	Density	Communities detected # (single)	Runtime (s)	Fast modul-arity	CFinder
Dolphins[a]	62	159	5.13	0.084	6 (16)	0.117	4	4
Zachary[b]	34	78	4.6	0.14	4 (2)	0.45	3	3
Football[c]	115	613	10.66	0.0935	23 (0)	0.277	7	4
Jazz[d]	198	2,742	27.69	0.14	64 (6)	0.86	4	2
Email[e]	1,133	5,452	9.6	0.008	390 (293)	24.38	12	41
PGP[f]	10,680	24,316	4.55	0.26	793 (7,181)	654.74	698	734

Datasets from http://deim.urv.cat/~Eaarenas/data/welcome.htm
[a]See reference [31].
[b]See reference [47].
[c]See reference [22].
[d]See reference [21].
[e]See reference [25].
[f]See reference [4].

1.5 Follow-on Work

Since our algorithm presents a new definition of community, it was anticipated that there would be significant deviation in results between our algorithm and what others have achieved. Identifying the most efficient methods to measure and compare our results against other algorithms, beyond a simple Jaccard similarity score, is an area for future research. As our algorithm also identifies the nodes that form the overlap, an analysis of those nodes for their ability to act as brokers, and as structural holes, should be cultivated.

In processing the larger datasets, a growing number of nodes not belonging to any community were detected, raising two questions that we plan to further investigate: (1) Is our assumption that a single edge node does not belong to a community valid? (2) Can link weighting be used to further cast a node into one community or another?

Another technique for evaluating our algorithm is to compare it using the LFR [28, 29] benchmark from Lancichinetti, Fortunato, and Radicchi. The benchmark includes a data generator that produces a graph with a known number of communities, and a known internal structure of those communities. Furthermore, the generator does allow communities to overlap. The generator can produce a number of graphs with varying amount of interaction (i.e. links) between the communities. The benchmark measures an algorithm's ability to detect communities as the numbers of cross-community edges are increased. Excepting the algorithm to constantly detect the original communities as the number of cross-community edges increases we believe to be erroneous.

The main focus of this research has been on the community detection portion of the algorithm and not on the merging of the friendship groups. An investigation of that set merging is expected to aid in the reduction of overall runtime. Lastly, we would like to further study the parallel capabilities of the algorithm by exploring multi-threading options or rewriting the algorithm in OpenCL.

1.6 Conclusion

Detection of the underlying community structure is an important part of intuitive network analysis. Failure to consider and account for overlapping groups creates a situation where the true community structure can go undetected. In this paper, we have presented a new approach for detecting overlapping communities, based on the unique perspective of individual group members, which we called friendship-groups. This approach, we believe, defines a more insightful notion of community and creates a potential for future performance enhancements.

Acknowledgements The authors are grateful to Graham Cruickshank for his proofreading skill.

References

1. Bagrow, J.P.: Evaluating local community methods in networks. J. Stat. Mech. **2008**(05), P05001 (2008)
2. Baumes, J., Goldberg, M., Magdon-Ismail, M.: Efficient identification of overlapping communities. In: IEEE International Conference on Intelligence and Security Informatics (ISI), pp. 27–36. Springer, Berlin/Heidelberg (2005)
3. Baumes, J., Goldberg, M.K., Krishnamoorthy, M.S., Magdon-Ismail, M., Preston, N.: Finding communities by clustering a graph into overlapping subgraphs. In: Guimarães, N., Isaías, P. (eds), Proceedings of the IADIS International Conference on Applied Computing, Algarve, Portugal, 97–104. IADIS Press (2005)
4. Bogua, M., Pastor-Satorras, R., Diaz-Guilera, A., Arenas, A.: Models of social networks based on social distance attachment. Phys. Rev. E **70**, 056122 (2004)
5. Brandes, U., Delling, D., Gaertler, M., Gorke, R., Hoefer, M., Nikoloski, Z., Wagner, D.: On modularity clustering. IEEE Trans. Knowl. Data Eng. **20**(2), 172–188 (2008)
6. Chartrand, G.: Introductory Graph Theory. Dover, New York (1985) [1977]
7. Clauset, A., Newman, M.E.J., Moore, C.: Finding community structure in very large networks. Phys. Rev. E **70**, 066111+ (2004)
8. Clauset, A.: Finding local community structure in networks. Phys. Rev. E **72**, 026132 (2005)
9. Davis, G., Carley, K.: Clearing the fog: fuzzy, overlapping groups for social networks. Soc. Netw. **30**, 201–212 (2008)
10. de Nooy, W., Mrvar, A., Batagelj, V.: Exploratory Social Network Analysis with Pajek. Cambridge University Press, Cambridge (2005)
11. Der enyi, I., Palla, G., Vicsek, T.: Clique percolation in random networks. Phys. Rev. Lett. **94**, 160202+ (2005)
12. Donetti, L., Munoz, M.A.: Detecting network communities: a new systematic and efficient algorithm. J. Stat. Mech. **2004**(10), 10012 (2004)
13. Du, N., Wu, B., Pei, X., Wang, B., Xu, L.: Community detection in large-scale social networks. In: WebKDD/SNA-KDD '07: Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis, pp. 16–25. ACM, New York (2007)

14. Du, N., Wang, B., Wu, B.: Overlapping community structure detection in networks. In: CIKM '08: Proceeding of the 17th ACM Conference on Information and Knowledge Management, pp. 1371–1372. ACM, New York (2008)
15. Everett, M.G., Borgatti, S.P.: Analyzing clique overlap. Connections **21**(1), 49–61 (1998)
16. Everett, M., Borgatti, S.P.: Ego network betweenness. Soc. Netw. **27**, 31–38 (2005)
17. Freeman, L.C.: Centrality in social networks conceptual clarification. Soc. Netw. **1**(3), 215–239 (1979)
18. Freeman, L.C.: Centered graphs and the structure of ego networks. Math. Soc. Sci. **3**, 291–304 (1982)
19. Freeman, L.C.: The sociological concept of group; an empirical test of two models. Am. J. Sociol. **98**(1), 152–166 (1992)
20. Getoor, L., Diehl, C.P.: Link mining: a survey. SIGKDD Explor. Newsl. **7**(2), 3–12 (2005)
21. Gleiser, P.M., Danon, L.: Community structure in jazz. Adv. Complex Syst. (ACS) **6**(4), 565–573 (2003)
22. Girvan, M., Newman, M.E.: Community structure in social and biological networks. Proc. Nat. Acad. Sci. **99**, 7821–7826 (2002)
23. Granovetter, M.S.: The strength of weak ties. Am. J. Sociol. **78**(6), 1360–1380 (1973)
24. Gregory, S.: An algorithm to find overlapping community structure in networks. In: PKDD 2007: Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases, pp. 91–102, Springer, Berlin/Heidelberg (2007)
25. Guimera, R., Danon, L., Diaz-Guilera, A., Giralt, F., Arenas, A.: Self-similar community structure in a network of human interactions. Phys. Rev. E **68**, 065103(R) (2003)
26. Hwang, W., Kim, T., Ramanathan, M., Zhang, A.: Bridging centrality: graph mining from element level to group level. In: KDD '08: Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 336–344. ACM, New York (2008)
27. Kossinets, G., Watts, D.J.: Empirical analysis of an evolving social network. Science **311**, 88–90 (2006)
28. Lancichinetti, A., Fortunato, S.: Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. Phys. Rev. E **80**, 016118 (2009)
29. Lancichinetti, A., Fortunato, S., Radicchi, F.: Benchmark graphs for testing community detection algorithms. Phys. Rev. E **78**, 046110 (2008)
30. Lancichinetti, A., Fortunato, S., Kertesz, J.: Detecting the overlapping and hierarchical community structure of complex networks. New J. Phys. **11**, 033015 (2009).
31. Lusseau, D., Schneider, K., Boisseau, O.J., Haase, P., Slooten, E., Dawson, S.M.: The bottlenose dolphin community of Doubtful Sound features a large proportion of long-lasting associations. Behav. Ecol. Sociobiol. **54**, 396–405 (2003)
32. Marriam-Webster online dictionary. http://www.merriam-webster.com
33. Moody, J., White, D.R.: Structural cohesion and embeddedness: a hierarchical concept of social groups. Am. Sociol. Rev. **68**(1), 103–127 (2003)
34. Newman, M.E.J.: Detecting community structure in networks. Eur. Phys. J. B **38**, 321–330 (2004)
35. Newman, M.E.J.: Fast algorithm for detecting community structure in networks. Phys. Rev. E **69**(6), 066133+ (2004)
36. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Phys Rev. E, **69**(12), 026113 (2003)
37. Noack, A., Rotta, R.: Multi-level algorithms for modularity clustering. In: SEA '09: Proceedings of the 8th International Symposium on Experimental Algorithms, pp. 257–268. Springer, Berlin/Heidelberg (2009)
38. Palla, G., Derenyi, I., Farkas, I., Vicsek, T.: Uncovering the overlapping community structure of complex networks in nature and society. Nature **435**, 814 (2005)
39. Pinney, J.W., Westhead, D.R.: Betweenness-based decomposition methods for social and biological networks. In: Barber, S., Baxter, P.D., Mardia, K.V., Walls, R.E. (eds.) Interdisciplinary Statistics and Bioinformatics, Leeds, England, 87–90. Leeds University Press (2006)

40. Radicchi, F., Castellano, C., Cecconi, F., Loreto, V., Parisi, D.: Defining and identifying communities in networks. Proc. Nat. Acad. Sci. U.S.A. **101**, 2658–2663 (2004)
41. Rees, B.S., Gallagher, K.B.: Overlapping community detection by collective friendship group inference. Int. Conf. Adv. Soc. Netw. Anal. Min. **0**, 375–379 (2010)
42. Wakita, K., Tsurumi, T.: Finding community structure in mega-scale social networks. In: WWW '07: Proceedings of the 16th International Conference on World Wide Web, pp. 1275–1276. ACM, New York (2007)
43. Wasserman, S., Faust, K.: Social Network Analysis: Methods and Applications. Cambridge University Press, Cambridge/New York (1994)
44. Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world' networks. Nature **393**, 440–442 (1998)
45. Wu, F., Huberman, B.A.: Finding communities in linear time: a physics approach. Eur. Phys. J. B **38**, 331–338 (2004)
46. Xu, X., Yuruk, N., Feng, Z., Schweiger, T.A.J.: Scan: a structural clustering algorithm for networks. In: KDD '07: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 824–833. ACM, New York (2007)
47. Zachary, W.: An information flow model for conflict and fission in small groups. J. Anthropol. Res. **33**, 452–473 (1977)