Clique Guided Community Detection

Diana Palsetia*, Md. Mostofa Ali Patwary†, William Hendrix*, Ankit Agrawal*, Alok Choudhary*

*Northwestern University, Evanston, IL
{drp925,whendrix,ankitag,choudhar}@eecs.northwestern.edu

†Parallel Computing Lab, Intel, Santa Clara, CA
mostofa.ali.patwary@intel.com

***Abstract*—Discovering communities to understand and model network structures has been a fundamental problem in several fields including social networks, physics, and biology. Many algorithms have been developed for finding the communities. Modularity based technique is fairly new relative to clustering, though it is very popular currently. Although some fast modularity based algorithms exist for detecting communities, the quality of these solutions is limited. At the other extreme, a clique embodies a basic community as it has the greatest possible edge density. However, the requirement that each pair of vertices be connected is too strict. Therefore, techniques to merge partitioned cliques using a hill-climbing greedy algorithm have been studied to form communities. However, the task of finding cliques is computationally expensive.**

In this paper, we present a new approach for fast and efficient community detection. We propose a clique guided community detection framework that consists of two phases. In the first phase, the framework finds disjoint cliques. In the second phase, the cliques from the first phase are used to guide the merging of individual vertices until a good quality solution is obtained. For the first phase, we develop an algorithm named MACH (Maximum Clique Heuristic), which is a new approach to compute disjoint cliques using a heuristic-based branch-and-bound technique. We provide experimental results to demonstrate the efficiency of the new algorithm and compare our approach with other previously proposed algorithms.

***Keywords*-Link and Graph Mining; Graph Clustering; Community Detection**

I. INTRODUCTION

A fundamental problem in the study of networks is community detection [1]. Analyzing networks is useful as they model complex systems [2]. For example, Social networking Sites (SNS), which allow millions of users to interact and allow connections represented by frendship relations or content sharing [3]. The identification of community structures in large-scale SNSs allows to group users with similar profiles and interests and can be employed to identify targets for marketing or improving the quality of recommender systems. In recent years, there has been a surge of interest to develop computational techniques that help understand the properties of large network datasets.

Fortunato [4] provides a survey of community detection algorithms on different types of networks. The communities can be disjoint or overlapping. In this paper we restrict our work to networks that are unweighted, and undirected, which form disjoint communities. Recent work in clustering has focused on the idea of maximizing modularity, a metric to measure the quality of the solution. Although some fast algorithms exist for detecting communities [5], [6], the quality of these solutions for large-scale networks is affected by the resolution limit i.e., the tendency of producing larger communities that may contain small communities with high density but lower connectivity between them [7], [8].

In social sciences, cliques are among the most visible and interesting types of groups in society. In network theory, a clique embodies a basic community as it has the greatest possible edge density. However, the requirement that each pair of vertices be connected is too strict. Recently techniques have been developed where cliques are considered as the starting communities to be merged based on modularity or other metrics [9], [10]. However note that the task of finding cliques itself is computationally expensive [11]. To tackle this problem, heuristics have been developed to obtain sub-optimal solution in a reasonable time.

In this paper, we present a new approach for fast and efficient community detection. We propose a clique guided community detection framework that consists of two phases. In the first phase, the framework finds disjoint cliques. For the first phase, we develop an algorithm named MACH (Maximum Clique Heuristic), which is a new approach to compute disjoint cliques using heuristic-based branch-and-bound technique. The idea is that MACH first computes the maximum clique size for each vertex and maintains the vertex order based on decreasing clique size. The heuristic bounds the graph traversal from a vertex to strictly fewer neighbors (whose degree is greater than or equal to the size of the maximum clique already computed). Pre-computing the graph traversal order is another strategy that reduces the search space for our algorithm as it avoids repeated exploration of the same vertices. It then explores the graph based on the vertex order to find all disjoint cliques (i.e. initial communities). In the second phase, the cliques from the first phase are used to guide the merging of individual vertices until a good quality solution is obtained.

978-1-4799-5666-1/14/$31.00 ©2014 IEEE

Our approach, though similar to approach in [12], differs in both phases. In the first phase, Yan and Gregory [12] use approximate BronKerbosch, which is an existing maximal clique algorithm. In the second phase, the start up cost that is required to setup the initial communities for our approach does not require additional time and space. We also compare our MACH algorithm to a modified version of Tomita et el [13] called TOMD (Tomita Disjoint) to find all maximal disjoint cliques. From our experiments, we find that TOMD provides significant improvement in runtime over technique propsed in [12] for finding all disjoint cliques. However, our MACH approach achieves significantly faster execution time over TOMD. Our results on both random and real-world datasets show that MACH with guided merging technique has a higher qualitiy solution and is better or equivalent in runtime when compared to the previous approaches in [6], [7], [12], [14], [15].

II. Related Work

In recent years, many new algorithms for detecting communities have been proposed, most of which belong to one of the two broad categories, *divisive* and *agglomerative*. Divisive algorithms initially assume that all vertices belong to one community and then the algorithms keep partitioning the vertices until they obtain the desired communities. For example in the divisive approaches proposed in [16], [17], [18], the vertices or edges with the largest *betweenness* (the number of shortest paths passing through the edge or vertex) are removed one by one to split the graph into communities hierarchically. Agglomerative algorithms [5], [19], [20] on the other hand starts assuming each vertex as a singleton communtiy and then the algorithms iteratively merge the communities until they obtain meaningful communities.

Many of these community detection algorithms use an well-known metric called *modularity* during the division or the merging. Modularity is a quantitative measure of the quality of a partition of a graph. More formally, it is the fraction of edges that fall within the given communities minus the expected fraction of edges if the edges were distributed at random [5]. The formulation of modularity reflects that each community should have a high number of intra-community edges and few inter-community edges. It can be used to compare the quality of different partitioning algorithms of a graph. The formula for modularity is provided in Equation 1. The term e_{ii} denotes the fraction of edges that fall within community i. Then $\sum_i e_{ii}$ is the total fraction of edges that fall within all communities and the term $a_i = \sum_j e_{ij}$ is the proportion of links belonging to community i.

$$Q(C) = \sum_i (e_{ii} - a_i^2) \tag{1}$$

The original algorithm proposed by Newman [5] uses a greedy approach in which the algorithm starts with n communities corresponding to the vertices of G i.e., each community starts with a singleton member. The algorithm repeatedly chooses the community pair i and j with the largest contribution to Q (maximum ΔQ) and merges them into a new community. Since the number of community pairs decreases monotonically, the algorithm eventually stops when maximum *modularity* is reached. It has a runtime complexity of $O((m+n)n)$ or $O(n^2)$ for sparse networks.

In [6] Clauset, Newman, and Moore (CNM) further improved the complexity to $O(md \log n)$ where d is depth of the dendrogram by demonstrating that the contribution, ΔQ can be easily computed using the previous iteration, thereby avoiding recalculations. However, CNM computation is significantly affected when merging communities of unbalanced sizes, which yields very unbalanced dendrograms. In such cases, the relation $d \sim \log n$ does not hold anymore and hence the runtime complexity deteriorates from $O(n \log^2 n)$ to $O(n^2 \log n)$ for sparse graphs [15].

Dannon, Diaz and Arenaz (DDA) [14] proposed a normalization to ΔQ so as to treat communities of different sizes as equal by diving ΔQ by the number of links (k_i). DDA improves CNM by improving the modularity while retaining its speed. Note that DDA applies the normalization to all community pairs (i,j) and the CNM step is modified to choose a pair(i,j) with with $max_{ij}(\Delta Q^*_{ij})$ where $\Delta Q^*_{ij} = \frac{\Delta Q}{k_i}$. Note that the real value of Q is calculated at each step using the original ΔQ. Wakita and Tsurumi (WT) [15] proposed some modifications to accelerate CNM with different data structures and consolidation-ratio heuristic to avoid unbalanced community merges, but encountered a decrease in modularity in their fastest algorithm (called HE). In our analysis we compare our approach to HE' implementation as it provides the best modularity but has higher cost compared to HE.

Yuta and Leon [7] further proposed an improvement to CNM model. They introduced their own data structures and acceleration techniques namely LY, DDA-M1 and DDA-M2 to further improve quality and speed of CNM. LY applies factor that penalizes communities that have large number of interconnected communities (denoted as *nic*) and is applied to the to community pair(i,j) with $max_{ij} \Delta Q_{ij}$. DDA-M1 and DDA-M2 are extensions to DDA but unlike DDA, the factors keep the symmetry by taking $min(k_i, k_j)$.

Based on the experiments and discussion in [7], LY and DDA-M1 behave differently based on the level of randomness of a network. DDA-M1 is slower for larger level of randomness as it prioritizes the pair of communities that has lower degree regardless of the degree of the pair, which may be larger in randomized networks and therefore impacting the speed. Based on the experiments, LY on average produces lower modularity and DDA-M2 is found to be in between LY and DDA-M1.

However, having maximum modularity does not necessarily reflect that a network has a community structure [4], [8]. In particular, it remains true even if the communities

are cliques [8]. These hill climbing algorithms could give poor results in some cases, where some communities tend to become excessively large [12]. One way to overcome this problem is by detecting communities at a much finer scale by recursively applying the algorithm on sub-networks that are eligible for further partitioning have been proposed in [21], [22]. Another approach is by forming initial communities [23], [24], [25]. An alternate approach is considering the cliques of a graph as the starting communities and then merge them based on modularity or other metrics. Several such algorithms exists such as CFinder [9], [26] and Greedy Clique Expansion (GCE) [27], but these techniques are only applicable to identify overlapping communities.

Yan and Gregory [12] presents a general clique based community detection algorithm called CliqueMod (CM). It consists of two phases: (i) divide the graph into a set of disjoint cliques (intial communities) and (ii) repeatedly merge the communities until maximum modularity is reached. The second phase is similar to the merging used in the CNM algorithm. In the first phase, [12] used two variations of clique algorithms: Konc and Janezic (KJ) algorithm [28] and Bron and Kerbosch (BK) algorithm [29]. These two variations of the CliqueMod (CM) algorithms are denoted by CM-KJ and CM-BK. KJ finds a maximum clique using graph-coloring techniques. CM-KJ thus finds all maximum cliques, by repeatedly calling KJ with the optimization that the size of the last clique found is used as and upper bound. CM-BK finds all approximately maximal disjoint cliques. Like BK, it repeatedly expands candidate cliques until they are maximal. But once a maximal clique is found, it outputs it immediately and deletes vertices that belong to current maximal clique instead of exploring alternatives. Both algorithms have been compared with the existing well known algorithms including WT [15], and CNM [6] algorithms. [12] established that CM-KJ and CM-BK algorithms outperform all these algorithms both in speed and quality (modularity). Additionally, CM-BK is faster than CM-KJ as the time complexity of CM-KJ is exponential. CM-BK in the worst case has $O(n^2)$ complexity. WT is found to be competitive with the CM algorithms in terms of speed, but consistently gives lower modularity. Similar to CM algorithm, our proposed framework uses a 2-phase approach, but differs in both phases. In the first phase, we develop a new technique to find all disjoint cliques called MACH. In the second phase, the key difference is that we still start with singleton communities and then use the cliques to guide the initial merging process. This is advantageous because we do not incur the overhead of formulating initial communities with cliques.

For comparison purposes, we have also implemented a modified version of the BK algorithm developed by Tomita et al [13]. It uses a specific pivoting policy to cut computational branches. Instead of iterating at each expansion on the set P (set containing possible candidates), it choses a pivot. The results will have to contain either the pivot or one of its non-neighbors, since if none of the non-neighbors of the pivot is included, then we can add the pivot itself to the result. Hence we can avoid iterating on the neighbors of the pivot at this step. Like BK in [12], for disjoint cliques, once a maximal clique is found, it gets outputted immediately and vertices that belong to current maximal clique are deleted instead of exploring alternatives. Throughout the paper, we use BKD and TOMD to denote Bron and Kerbosch and Tomita algorithms, which extract approximately maximal disjoint cliques.

III. ALGORITHM

In the following subsections, we present our framework for community detection using clique guided merging. Our framework consists of two phases. In the first phase all disjoint cliques are extracted from the network. In the second phase we start to merge the individual vertices using the cliques to guide the merging process.

Let n be the number of vertices of the input graph $G = (V, E)$ where $V = \{v_1, v_2, \ldots, v_n\}$ and E denote the set of vertices and edges, respectively. The set of vertices adjacent to a vertex v_i, the set of its neighbors, is denoted by $N(v_i)$ and the cardinality of $N(v_i)$, its degree, is denoted by $d(v_i)$. The community detection problem is typically formulated as finding a partition $C = \{c_1, \ldots, c_k\}$ of G, where $\forall_i, c_i \subseteq V$ and $\forall_{i,j}, c_i \cap c_j = \emptyset$, which gives non-overlapping communities. C is also known as a *clustering* of G. We use k to denote the number of resulting communities, that is, $|C| = k$.

A. Maximum Clique Heuristic (MACH) Algorithm

To discover all cliques, we use a heuristic based branch-and-bound technique to determine a maximum clique. MACH first computes the maximum clique size for each vertex containing itself and stores them in a priority-queue, and then iterates over the priority-queue to find all disjoint cliques. The cliques are used to guide the merging process in next phase.

A clique in an undirected graph is a subset of vertices in which every pair of vertices are adjacent to each other. The *maximum* clique problem is to find the clique of largest possible size in a given graph. The maximum clique problem is NP-Hard [11]. Therefore, most practical applications employ some form of branch-and-bound approach [30]. While *branching* systematically searches for all candidate solutions, *bounding* (also known as *pruning*) discards fruitless candidates based on a previously computed bound. An example of a simple and effective branch-and-bound algorithm for the maximum clique problem is presented in [30].

In [31], we proposed an effective branch-and-bound algorithm based on degree of a vertex. To obtain the maximum clique in a graph, our approach computes the largest clique containing each vertex and the largest among these is picked. The main ingredient of our algorithm is that during the search for the largest clique containing a given vertex, vertices that cannot form cliques larger than the current maximum clique are *pruned*, in a hierarchical fashion. This reduces the search space and thus improves the performance. We also introduced a heuristic approach, which speeds up this process by examining only a subset of the relevant cliques making the algorithm orders of magnitude faster than the exact algorithm, while providing optimal or near-optimal solutions.

A naïve way to extract all disjoint cliques would be by extracting the largest clique at every iteration, however, it can repeat traversal of vertices that have not been eliminated from the graph and that are not part larger cliques. This approach would be very expensive for large graphs. Therefore MACH's main contribution is that it determines the largest clique size for each vertex in graph G and stores this information in a priority-queue Q where the priority is the vertex with the largest clique. To get all disjoint cliques, MACH iterates through the running graph G' in order of the vertices in the priority-queue Q. This exploration order significantly prunes the search space and hence provides a significant time reduction in finding all disjoint cliques.

Algorithm 1 Heuristic for finding the maximum clique in a graph. *Input*: Graph $G = (V, E)$. *Output*: Set of Vertices that form maximum clique [31].

```
1: procedure MaxCliqueHeu(G = (V, E))
2:     for i : 1 to n do
3:         if d(v_i) >= max then                          ▷ Pruning 1
4:             U ← ∅
5:             for each v_j ∈ N(v_i) do
6:                 if d(v_j) >= max then                  ▷ Pruning 2
7:                     U ← U ∪ {v_j}
8:             CliqueHeu(G, U, 1)
```

– *Subroutine*

```
1: procedure CliqueHeu(G = (V, E), U, size)
2:     if U = ∅ then
3:         if size > max then
4:             max ← size
5:         return
6:     Select a vertex u ∈ U of maximum degree in G
7:     U ← U \ {u}
8:     N'(u) := {w|w ∈ N(u) ∧ d(w) > max}              ▷ Pruning 3
9:     CliqueHeu(G, U ∩ N'(u), size + 1)
```

The branch-and-bound heuristic that assists MACH is presented in Algorithm 1. The main routine MaxCliqueHue considers only the neighbors with *maximum degree* at each step instead of recursively considering all neighbors. The routine thus generates for each vertex $v_i \in V$ a set $U \subseteq N(v_i)$ (neighbors of v_i that survive pruning) and calls the subroutine CliqueHeu on U. Since we are looking for the largest clique containing each vertex, the maximum degree vertex is more likely to be a member of the largest clique compared to the other vertices. The effect of choosing the maximum degree vertex as opposed to any random vertex is discussed in detail in [31]. Throughout the algorithm, the variable max stores the size of the maximum clique found so far. Variable $size$ indicates the size of the clique found at any point through the recursion. Since we start with a clique of just one vertex, the value of $size$ is set to be 1 initially when the subroutine CliqueHeu is called (Line 8 of Algorithm 1), from MaxCliqueHue. Our algorithm consists of several pruning steps to filter our vertices. The most significant of these pruning steps is on Line 8 of subroutine CliqueHeu, which further chooses only those neighbors of the max degree vertex whose degree is greater than or equal to the current max clique.

Algorithm 2 MACH(G): Disjoint Clique Extraction Algorithms *Input*: Graph $G = (V, E)$. *Output*: Cliques $Clq\_list = \{c_1, c_2, \ldots, c_k\}$ and $|Clq\_list|$ denotes the number of cliques.

```
1: procedure MACH(G = (V, E))
2:     /*Phase I - Find all cliques*/
3:     Declare Clq_Size[n] as Integer
4:     Initialize Clq_Size[i] ← 0 for all i
5:     for each v_i in G do
6:         for each v_j ∈ N(v_i) do
7:             if d(v_j) >= Clq_Size[v_i] then
8:                 U ← U ∪ {v_j}
9:         clq ← CliqueHeu(G, U, 1)
10:        for each v_i in clq do
11:            if Clq_Size[v_i] < |clq| + 1 then
12:                Clq_Size[v_i] ← |clq| + 1
13:    Q ← ∅
14:    for c_i in Clq_Size do
15:        Q.add(c_i, |c_i|)
16:    G' ← G
17:    while (Q ≠ ∅) do
18:        {v, cs} ← Q.pop()
19:        if (v ∉ G') then
20:            continue
21:        clq ← MaxCliqueHeu2(G', v, cs, Clq_Size)
22:        clq ← clq ∪ {v}
23:        Mark all vertices c_i in clq as processed
24:        Clq_list ← Clq_list ∪ clq
25:        G' ← UpdateGraph(G', clq)
26:    return Clq_list
```

Algorithm 2 provides the psuedocode of MACH. The algorithm starts by initializing an array Clq_Size of size n. To determine the clique size per vertex (Line 5-12), the algorithm employs similar approach as discussed in Algorithm 1, except that it does not iterate over the entire graph but just for the vertex of interest. For each vertex, a set of neighbors U is chosen. The criteria for a neighbor, v_j, of vertex v_i to be part of U is based on whether its degree is

greater than equal to v_i's current clique size (Line 7). With the initial set U, the subroutine CLIQUEHEU is called and hence for each vertex v_i, a set vertices that form a clique (clq) with v_i is established. For each vertex $v_i \in clq$, their clique size($Clq\_Size[v_i]$) is updated if the vertex's current clique size is lower than the current clique size ($|clq| + 1$). This update allows each vertex to reflect the largest clique it can be part.

Next, the vertices are added to the priority-queue Q and ordered by their max clique size (Line 13-15). Next, we explore the graph to extract cliques (Line 17-25) based on the order of vertices in the priority-queue. Note this is a significant pruning strategy as we now have a guide in traversing the graph, and we will also not explore the same vertices repeatedly. Because of this pruning step, we achieve significantly faster execution time. To find all disjoint cliques we still use our heuristic described in Algorithm 1 but with few modifications. Algorithm 3 describes the modified heuristic algorithm. MAXCLIQUEHEU2 explores the vertex of interest (instead of going through all vertices). In the pruning step, we only allow neighbors that have degree greater than equal to $cs - 1$ where cs is clique size of the interested vertex, and whose clique size is greater than or equal to cs (Line 6 in Algorithm 3). The latter is similar bounding principal [31] but instead of the current maximum clique size, we use the size of the largest clique the interested vertex can be part of. Similarly, in subroutine CLIQUEHEU2, the pruning step uses cs (Line 9) instead of using max (where max is the size of the current clique) to prune the neighbors. Once the clique is determined, the graph is updated (Line 25). Additionally we make sure that all the vertices that form a clique are marked as processed so that we can skip these vertices when they are popped from the queue (Line 23).

After all cliques are found, Phase II of our approach is performed. The singleton communities are merged based on the cliques. This allows the greedy merge to start at a good starting point. Once the initial clique communities are established, the rest of the communities are established based on greedy merge.

B. CLIQUE GUIDED COMMUNITY DETECTION

The greedy hill climbing algorithm proposed in [5] starts with singleton communities. The algorithm then computes change in modularity, denoted by ΔQ, for each pair of communities, say c_i and c_j when they are merged. Then the algorithm repeatedly chooses a pair of communities with maximum ΔQ and joins them into a new community. Note that due to this merge, ΔQ needs to be updated for those pair communities connected to this pair. Also, this merging reduces the number of communities monotonically and therefore, algorithm continues as long as there two communities to be merged or when $\Delta Q < 0$.

Algorithm 3 Heuristic for finding the maximum clique in a graph. *Input*: Graph $G = (V, E)$. *Output*: Set of Vertices that form maximum clique [31].

```
1: procedure MAXCLIQUEHEU2(G = (V,E), v_i, cs, Clq_Size)
2:     max ← 0
3:     U ← ∅
4:     clq ← ∅
5:     for each v_j ∈ N(v_i) do
6:         if d(v_j) >= cs − 1 ∧ Clq_Size[v_j] >= cs then
7:             U ← U ∪ {v_j}
8:     clq ← CLIQUEHEU2(G, U, max, clq, 1, Clq_Size[v_i])
9:     return clq
```

– *Subroutine*

```
1: procedure CLIQUEHEU2(G,U,max,clq, size, cs, Clq_Size)
2:     maxPrev ← 0
3:     if U = ∅ then
4:         if size > max then
5:             max ← size
6:         return
7:     Select a vertex u ∈ U of maximum degree in G
8:     U ← U \ {u}
9:     N'(u) := {w|w ∈ N(u) ∧ Clq_Size[w] >= cs}
10:    maxPrev = max
11:    CLIQUEHEU2(G, U ∩ N'(u), size + 1, cs, Clq_Size[u])
12:    if max > maxPrev then
13:        clq ← clq ∪ u
```

Algorithm 4 CGCD (G, Clq_list) : Clique Guided Community Detection Algorithm. *Input*: Graph $G = (V, E)$, Clq_list denotes set of cliques from Phase I. *Output*: Communities $C = \{c_1, c_2, \ldots, c_k\}$, where $|C| = k$, $\forall_i, c_i$ is a resulting community.

```
1: procedure CGCD(G = (V,E), Clq_list)
2:     procedure UPDATEΔQ()
3:         for each pair of communities c_i ∈ C, c_j ∈ C do
4:             Update change in modularity, ΔQ^C_{c_i,c_j}
5:     procedure JOIN((C, c_i, c_j))
6:         C ← (C \ {c_i, c_j}) ∪ {c_i ∪ c_j}
       return C
7:
8:     C ← v ∈ V|{v}                          ▷ Singleton Communities
9:     Clq_list ← DisjointCliques(G)          ▷ Phase I
10:    for each clq ∈ Clq_list do             ▷ Phase II : Guided merge
11:        for each (c_i, c_j) ∈ clq do
12:            JOIN(C, c_i, c_j)
13:            UPDATEΔQ( )
14:    while (true) do                        ▷ Phase II : Greedy merge
15:        Find a pair c_i ∈ C, c_j ∈ C with max ΔQ
16:        if max < 0 then
17:            break
18:        JOIN(C, c_i, c_j)
19:        UPDATEΔQ( )
```

Since we know that cliques are good starting point, we modify the above approach to merge singleton communities based on the Phase I findings (Clq_list). Note that Phase I can use any algorithm that computes all disjoint cliques, therefore, denoted by the function *DisjointCliques*. Lines

10 - 13 in Algorithm 4 basically allows our algorithm to get to a good starting point. After we find the C partitions that contain cliques, we use the standard greedy strategy. Additionally, we have also experimented with reduction strategies that improve upon CNM (discussed in section II) namely, WT(HE'), DDA, DDA-M1, DDA-M2 and LY. Therefore, line 15 in Algorithm 4 is modified according to the weighting specified by these algorithms.

C. Complexity Analysis

We now provide the time complexity for our framework. In Phase I, for each vertex, algorithm MACH (Algorithm 2) first determines the maximum clique size by calling the heuristic MAXCLIQUEHUE2 (Algorithm 3), possibly calling the subroutine CLIQUEHEU2, which effectively is a loop that runs until the set $U \subseteq N(v)$ is empty. Clearly, $|U|$ is bounded by the max degree Δ in the graph. For each (recursive) iteration of MAXCLIQUEHUE2, we need to find the maximum degree vertex in U, calculate $N(u)$, and then remove all of the vertices from U that aren't in N'. If we assume that checking adjacency is O(1), then the runtime can be bounded by $O(\Delta)$. At each iteration at least one vertex u is removed from U (though in the case of a complete graph, only u might be removed). Therefore, the time complexity of the heuristic is $O(n \cdot \Delta^2)$. To find all disjoint cliques, in the worst case, the algorithm iterates through all the vertices in the priority queue, thus complexity of finding all cliques is $O(n \cdot \log n \cdot \Delta^2)$. Thus our proposed MACH approach has polynomial complexity for extracting all disjoint cliques. From our experiments, we find that our pruning strategies make MACH significantly faster than TOMD and BKD. In addition to the time complexity, our algorithm MACH has space complexity, $O(n)$ for storing the clique sizes, and $O(n)$ for priority-queue (each item in the queue store the vertex id and its clique size). Thus, the overall space complexity is linear.

As we adopt the framework and data structures from [15], our community merge step will take $O(k)$ time, where k is the number of communities. If the merging is very unbalanced in this phase, we could perform $O(n)$ merges, taking up to $O(n^2)$ time for this phase. While this time is potentially longer than CNM's complexity of $O(md \log n)$, this worst case analysis doesn't represent expected runtime well, as can be seen in our empirical results.

IV. EXPERIMENTS AND RESULTS

In this section we evaluate our proposed approach. In the first phase of the algorithm we deploy the BKD, TOMD and MACH algorithms. In the second phase we use the results from first phase to guide the initial merges and then use the greedy hill-climbing approach from CNM. We also incorporate strategies that improve quality and speed of CNM, namely WT(HE'), DDA, DDA-M1, DDA-M2 and LY for Phase II of our approach. We denote algorithms with clique guided community detection framework as X+Y where X implies the disjoint clique algorithm and Y implies greedy hill climbing approach.

To perform our experiments, we adopt the C++ framework[1] from Wakita et al [15]. Besides the heuristics discussed in Section II , significant improvements were made to CNM community detection framework described in [6]. The framework in [15], replaces the balanced binary trees and max heap data structured with a doubly-linked list that is sorted in the order of community ID. This framework initially came with implementation of CNM and WT heuristics algorithm. We have modified this framework for our implementation and other modifications to incorporate DDA* and LY. Our experiments are performed on a Intel®Xeon®[2] E5-2407, which is 2 x quad-core CPU running at 2.20 GHz and memory size of 32 GB. The programs are compiled using gcc with -O3 optimization.

A. Synthesized Networks

To generate more realistic model of real world graphs, we use the LFR benchmark [32] for testing the community detection algorithms. Using several parameters, the random networks can be used to emulate real networks. The parameters are as follows 1) n : number of vertices 2) k : average degree 3) γ : exponent of the power-law distribution for degree 4) cs : minimum community size 5) β : exponent of the power-law distribution for community sizes 6) μ : mixing parameter, or the fraction of the edges a vertex shares with other vertices in other communities.

To evaluate the quality of the algorithms, we use Adjusted Rand Index [33], a metric to compare the community structure outputted by algorithm to that of the known community structure generated by the benchmark. The measure of this index is based on how the pairs of vertices have been clustered. The Adjusted Rand Index is the corrected-for-chance version of the Rand index. The score ranges from -1 to 1, where a value of 1 indicates that the two solutions perfectly match. More details can be found in [33]. We first provide how the quality of partitioning is affected by varying different network parameters. Note that all results are averaged over 10 random networks generated while varying one parameter.

Figure 1 shows the effect of varying the mixing parameter, μ with the quality of the solution, as measured by the Adjusted Rand Index (ARI). We compare our clique guided technique with MACH in Phase I to non-clique guided techniques such as CNM and its modifications (WT, LY, DDA, DDA-M1, DDA-M2). Applying our clique-guided approach

[1]http://smartnova.net/ wakita/software/katsura-clustering-110613.zip

[2]Intel, Xeon, and Intel Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries.

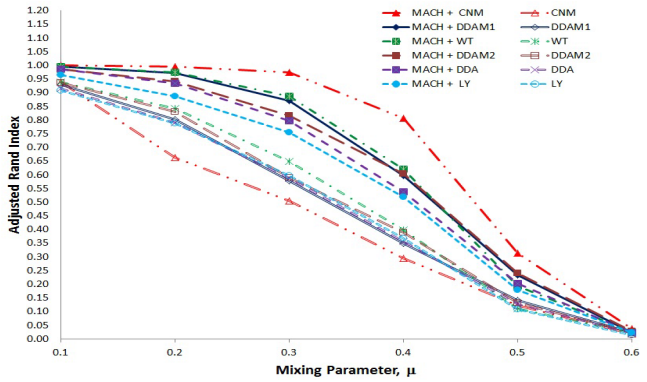


Figure 1. LFR Networks with n=1000, γ=2, β=1, k=10, cs=50, μ=0.1-0.6

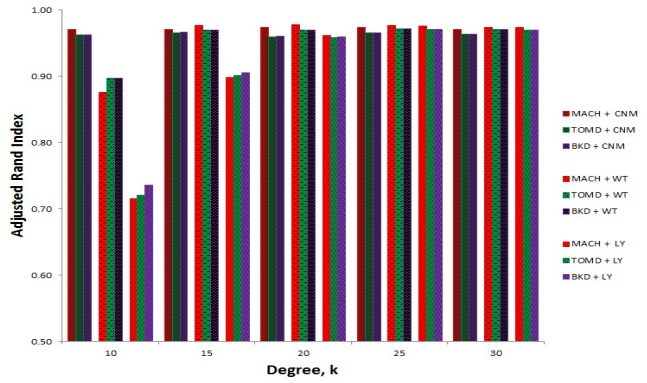


Figure 2. LFR Networks with n=1000, γ=2, β=1, μ=0.3, cs=50, k=10-30

improves overall quality compared to different baselines i.e. our MACH+X technique has higher quality than technique X in every case for $\mu < 0.6$. Our MACH+CNM (solid red marker) approach has the highest quality for this synthetic network configuration.

We chose μ=0.3 to further investigate change in other parameters as the network is random enough to determine how quality affected for our approach. In Figure 2 we vary the average degree parameter, k in the range 10-30. The maximum degree is set to 2.5*k. The quality for our approach improves with increase in degree for all approaches with MACH in Phase I except when CNM is used in Phase II. WT, and LY approaches in Phase II, also show significant improvement and have similar quality compared CNM in Phase II. Figure 2, shows the effect of quality with varying degree for BKD, MACH, and TOMD techniques in Phase I and CNM, WT and LY techniques for Phase II. Since our heuristic is based on maximum degree (discussed in Section III), all approaches that use MACH improve in quality and hence have similar or better quality compared to employing BKD and TOMD in Phase I. Thus, if a network has relatively high degree, our proposed disjoint clique algorithm is able to extract good quality cliques, which in turn produces a better quality partition. This is desirable feature for community detection for social networks as the average neighbor degree increases with the degree due to associativity, which is due to the preference for a network's nodes to attach to others that are similar in some way [34].

To vary the distribution parameters, we chose the range 2-3 for Degree Exponent parameter, γ and 1-2 for Community Size exponent, β as suggested in [32]. We find that quality for all approaches (greedy or clique guided) is unaffected. Similarly, to vary the community size parameter, we chose cs range 20-100. We find that all approaches except CNM have lower quality as the community size increases. Our approach MACH+CNM has lowest decrease in quality. The baseline CNM actually improves in quality with increase in community, which makes sense as to obtain a high modularity value, it tends to group more vertices into one community (to increase the first term in Equation 1). However, our approach still has higher quality compared to their respective baselines. For space, we do not include these figures.

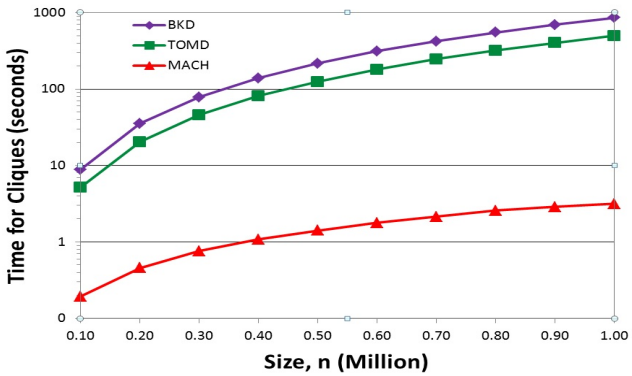
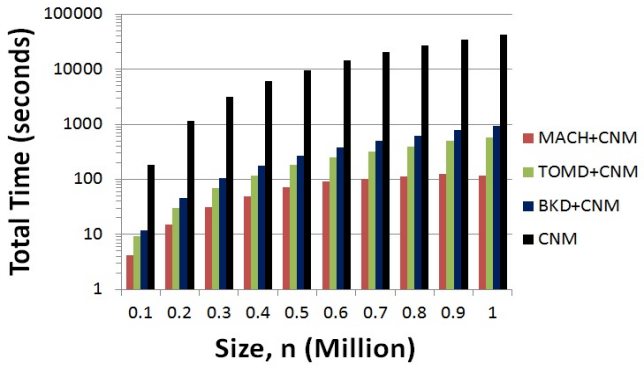
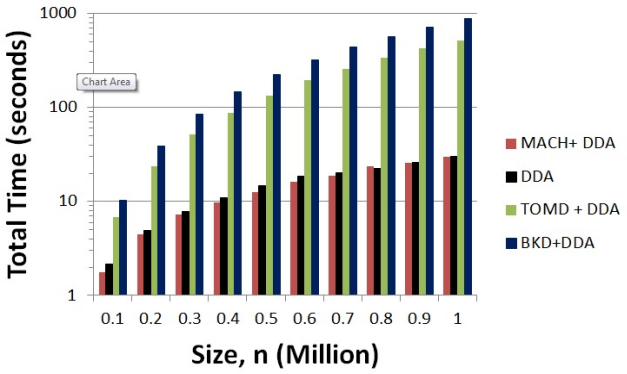


Figure 3. LFR Networks with k=10, cs = 50, γ=2, β=1, μ=0.3

Now we present the execution time taken by the algorithms. The network size, n (number of vertices in a graph) is varied from 100,000 to 1,000,000 vertices with 948,432 and 4,758,415 respectively. The other parameters are as follows: degree k=10, community size cs=50, γ=1, β=2, μ=0.4. Figure 3 provides the time to find all disjoint cliques. MACH clearly beats TOMD and BKD by two orders of magnitudes for this set of synthetic graphs because our algorithm has polynomial time complexity. TOMD is faster than BKD due its pivoting strategy, however the TOMD is only twice as fast compared to BKD. Figure 4 shows different clique algorithms in Phase 1 and total time, i.e. the time to find all disjoint cliques plus the time to merge to produce the final communities for CNM (Figure 4(a)) and DDA (Figure 4(b)) for Phase II. Our MACH with guided merging strategy is faster by up to an order of magnitude compared to the baseline CNM. Our approach is fast compared to TOMD and BKD, such that it does not add significant time to the overall algorithm. This is especially the case for baselines other than CNM. Thus, not only does MACH improves



(a) Total Time for CNM



(b) Total Time for DDA

Figure 4. Total Time for LFR Networks with k=10, cs = 50, γ=2, β=1, μ=0.3

Table I
MODULARITY FOR FACEBOOK COMMUNITY BASED ON RICE UNIVERSITY UNGDERGRAD POPULATION

	Ground Truth	CNM	MACH + CNM	DDA-M1	MACH + DDA-M1	DDA-M2	MACH + DDA-M2
Modularity	0.384	0.274	0.345	0.336	0.347	0.338	0.347
ARI	1.000	0.261	0.659	0.488	0.641	0.431	0.641

quality of the partition, but is competitive in terms of speed for baselines other than CNM (e.g. DDA in Figure 4(b)).

B. Real-World Networks

We experimented our new approach also on several real-world large-scale networks. The structural information of the networks can be found in Table II.

Table II
NETWORK INFORMATION OF REAL WORLD SOCIAL NETWORK GRAPHS

Network	Vertices	Edges	Ref
fb-rice-undergrad	1,220	43,208	[35]
ca-AstroPh	18,772	396,160	[36]
amazon0302	262,111	1,234,877	[37]
dblp	317,080	1,049,866	[37]
Delicious	103,144	1,419,519	[38]
Flixster	2,523,386	9,197,338	[38]
LiveJournal	2,238,731	14,608,137	[38]

In Table I we provide community detection results for the Facebook community extracted from Rice University undergraduate population [35]. The data has natural communities based on college id, year and major. We have extracted communities based college ids and use this partition to denote as ground truth. We show the Adjusted Rand Index (ARI) and modularity results for MACH+Y where Y = CNM, DDA, DDA-M1, DDA-M2. Note that modularity obtained for the partition based on college ids is higher than modularity obtained by all algorithms and hence ARI obtained will not be closer to 100%. Our technique is able to recover communities much more efficiently compared to their baselines. As interpreted from our synthetic networks, MACH+CNM is able to recover 65% of the communities, followed by MACH+DDA and MACH+DDA-M1/M2, which recovers 64.6% and 64.1% of the communities respectively. Using TOMD and BKD with any strategies in phase II, we achieve the same ARI and modularity values. For example, when we replace the MACH with TOMD or BKD and employing CNM for Phase II, 58.9% of partitions are recovered and modularity value is 0.322 for both.

In Table III, we provide the modularity (Q) the total execution time in seconds (T), speedup (S) over the respective baseline for real world networks (other than the fb-rice-undergrad dataset) with MACH as the Phase I algorithm and DDA* and CNM algorithms in Phase II. We also provide modularity and total execution time for baseline CNM as well. We observe speedup ranging from 1.15x - 11.49x (MACH+CNM technique has the highest speedup for the Amazon dataset). The only abnormal case is the Delicious dataset where baseline DDA-M1 does better compared to our approach (MACH+DDA-M1). The same is true for TOMD and BKD (result not shown). In terms of quality, any approach in Phase II with MACH in phase I provides higher compared to the baseline.

V. CONCLUSION

In this paper we have presented a fast and high quality community detection algorithm that uses the notion of clique guided merging. To discover all disjoint cliques, we develop a new algorithm called MACH (Maximum Clique

Table III
MODULARITY AND EXECUTION TIME FOR REAL WORLD GRAPHS FOR MACH WITH CNM AND DDA* , AND BASELINE CNM
Q : Modularity, T : Time (seconds), S: Speedup w/ resp. to Baseline

MACH	amazon			ca-AstroPh			dblp			Delicious			Flixster			LiveJournal		
with	Q	T(sec)	S	Q	T(sec)	S	Q	T(sec)	S	Q	T(sec)	S	Q	T(sec)	S	Q	T(sec)	S
DDA-M2	0.898	3.18	8.82	0.603	0.90	2.53	0.813	4.87	0.95	0.737	72.01	1.15	0.560	3642	1.59	0.618	7141	3.89
DDA-M1	0.900	5.85	4.89	0.608	1.08	1.83	0.812	26.98	1.26	0.726	1613.56	0.13	0.570	12371	1.61	0.600	31842.3	1.64
DDA	0.899	3.38	8.82	0.606	0.85	4.59	0.814	4.89	1.91	0.737	82.08	1.60	0.558	3568	1.78	0.620	7249	3.96
CNM	0.900	22.52	11.49	0.610	1.78	5.37	0.811	101.27	8.62	0.721	2258	1.66	0.572	93292	1.70	0.581	127884	2.15
Baseline CNM	0.813	258.72	1.0	0.511	9.53	1.0	0.733	873.43	1.0	0.673	3738	1.0	0.521	158477	1.0	0.532	275020	1.0

Heuristic). The main idea of our algorithm is that it first computes disjoint cliques using heuristic-based branch-and-bound technique and then merges singleton vertices initially using cliques from the previous phase followed by greedy merge to them to obtain the desired communities. Our proposed MACH approach has polynomial complexity, which is better than TOMD and BKD approaches. This is because our technique employs several pruning strategies to drastically reduce the search space. Based on the experiments, we conclude that MACH with clique guided merging approach is better in quality and time (up to an order of magnitude faster) compared to prior approaches such as [6], [7], [12], [14], [15].

In the future, we intend to optimize the merging phase of our algorithm using parallel techniques and advanced data structures. In many real-world networks, vertices may belong to more than one group, and such groups form overlapping communities. Finding such overlapping community is not supported by traditional community detection algorithms. However, clique based algorithms is one way in which a solution could possibly be obtained. Thus, we also would like to investigate the feasibility of our algorithm to compute overlapping communities.

VI. ACKNOWLEDGEMENT

This work is supported in part by the following grants: NSF awards CCF-1029166, ACI-1144061, IIS-1343639, and CCF-1409601; DOE awards DESC0007456.

REFERENCES

[1] M. E. J. Newman, "Modularity and community structure in networks," *PNAS*, vol. 103, no. 23, pp. 8577–8582, 2006.

[2] S. H. Strogatz, "Exploring complex networks," *Nature*, vol. 410, pp. 268–276, 2001.

[3] S. Wasserman and K. Faust, *Social Network Analysis*. Cambridge University Press, Cambridge, 1994.

[4] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3-5, pp. 75–174, 2010.

[5] M. E. J. Newman, "Fast algorithm for detecting community structure in networks," *Phys. Rev. E*, vol. 69, no. 6, p. 066133, 2004.

[6] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Phys. Rev. E*, vol. 70, no. 6, p. 066111, 2004.

[7] Y. I. Leon-Suematsu and K. Yuta, "A framework for fast community extraction of large-scale networks," in *Proceedings of the 17th International Conference on World Wide Web*, ser. WWW '08, 2008, pp. 1215–1216.

[8] S. Fortunato and M. Barthelemy, "Resolution limit in community detection," *Proceedings of the National Academy of Sciences*, vol. 104, no. 1, p. 36, 2007.

[9] B. Adamcsek, G. Palla, I. Farkas, I. Derenyi, and T. Vicsek, "Cfinder: locating cliques and overlapping modules in biological networks," *Bioinformatics*, vol. 22, no. 8, pp. 1021–1023, 2006.

[10] N. Du, B. Wu, X. Pei, B. Wang, and L. Xu, "Community detection in large-scale social networks," in *Proceedings of the 9th WebKDD and 1st SNA-KDD*. ACM, 2007, pp. 16–25.

[11] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[12] B. Yan and S. Gregory, "Detecting communities in networks by merging cliques," in *Intelligent Computing and Intelligent Systems, 2009. ICIS 2009. IEEE International Conference on*, vol. 1. IEEE, 2009, pp. 832–836.

[13] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theoretical Computer Science*, vol. 363, no. 1, pp. 28 – 42, 2006, computing and Combinatorics 10th Annual International Conference on Computing and Combinatorics (COCOON 2004). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304397506003586

[14] L. Danon, A. Díaz-Guilera, and A. Arenas, "The effect of size heterogeneity on community identification in complex networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2006, no. 11, p. P11010, 2006.

[15] K. Wakita and T. Tsurumi, "Finding community structure in mega-scale social networks:[extended abstract]," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 1275–1276.

[16] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *PNAS*, vol. 99, no. 12, pp. 7821–7826, 2002.

[17] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review E*, vol. 69, no. 2, p. 026113, 2004.

[18] J. Pinney and D. Westhead, "Betweenness-based decomposition methods for social and biological networks," *Interdisciplinary Statistics and Bioinformatics*, pp. 87–90, 2007.

[19] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi, "Defining and identifying communities in networks," *PNAS*, vol. 101, pp. 2568–2663, 2004.

[20] F. Wu and B. A. Huberman, "Finding communities in linear time: A physics approach," *The European Physical Journal B*, vol. 38, pp. 331–338, 2004.

[21] J. Ruan and W. Zhang, "Identifying network communities with a high resolution," *Physical Review E*, vol. 77, no. 1, p. 016104, 2008.

[22] D. Palsetia, M. M. A. Patwary, A. Agrawal, and A. N. Choudhary, "Excavating social circles via user interests," *Social Netw. Analys. Mining*, vol. 4, no. 1, 2014.

[23] J. M. Pujol, J. Béjar, and J. Delgado, "Clustering algorithm for determining community structure in large networks," *Physical Review E*, vol. 74, no. 1, p. 016107, 2006.

[24] H. Du, M. W. Feldman, S. Li, and X. Jin, "An algorithm for detecting community structure of social networks based on prior knowledge and modularity," *Complexity*, vol. 12, no. 3, pp. 53–60, 2007.

[25] H. Zardi and L. B. Romdhane, "An o(n2) algorithm for detecting communities of unbalanced sizes in large scale social networks," *Knowledge-Based Systems*, vol. 37, no. 0, pp. 19 – 36, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950705112001736

[26] G. Palla, I. Derenyi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, vol. 435, no. 7043, pp. 814–818, 2005.

[27] C. Lee, F. Reid, A. McDaid, and N. Hurley, "Detecting highly overlapping community structure by greedy clique expansion," *arXiv preprint arXiv:1002.1827*, 2010.

[28] J. Konc and D. Janezic, "An improved branch and bound algorithm for the maximum clique problem," *proteins*, vol. 4, p. 5, 2007.

[29] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.

[30] R. Carraghan and P. Pardalos, "An exact algorithm for the maximum clique problem," *Oper. Res. Lett.*, vol. 9, pp. 375–382, 1990.

[31] B. Pattabiraman, M. M. A. Patwary, A. H. Gebremedhin, W. keng Liao, and A. N. Choudhary, "Fast algorithms for the maximum clique problem on massive sparse graphs," *CoRR*, vol. abs/1209.5818, 2012.

[32] A. Lancichinetti, S. Fortunato, and F. Radicchi, "Benchmark graphs for testing community detection algorithms." *Phys Rev E Stat Nonlin Soft Matter Phys*, vol. 78, no. 4 Pt 2, p. 046110, 2008.

[33] L. Hubert and P. Arabie, "Comparing partitions," *Journal of Classification*, vol. 2, no. 1, pp. 193–218, 1985. [Online]. Available: http://dx.doi.org/10.1007/BF01908075

[34] M. E. J. Newman, "Mixing patterns in networks," *Phys. Rev. E*, vol. 67, p. 026126, Feb 2003. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.67.026126

[35] A. Mislove, B. Viswanath, K. P. Gummadi, and P. Druschel, "You are who you know: Inferring user profiles in Online Social Networks," in *Proceedings of the 3rd ACM International Conference of Web Search and Data Mining (WSDM'10)*, New York, NY, February 2010.

[36] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *ACM Trans. Knowl. Discov. Data*, vol. 1, no. 1, Mar. 2007. [Online]. Available: http://doi.acm.org/10.1145/1217299.1217301

[37] J. Leskovec, L. A. Adamic, and B. A. Huberman, "The dynamics of viral marketing," *ACM Trans. Web*, vol. 1, no. 1, May 2007. [Online]. Available: http://doi.acm.org/10.1145/1232722.1232727

[38] R. Zafarani and H. Liu, "Social computing data repository at arizona state university, school of computing, informatics and decision systems engineering," 2009. [Online]. Available: http://socialcomputing.asu.edu