On community detection in very large networks

Alexandre P. Francisco and Arlindo L. Oliveira

INESC-ID / CSE Dept, IST, Tech Univ of Lisbon
Rua Alves Redol 9, 1000-029 Lisboa, PT
{aplf,aml}@inesc-id.pt

Abstract. Community detection or graph clustering is an important problem in the analysis of computer networks, social networks, biological networks and many other natural and artificial networks. These networks are in general very large and, thus, finding hidden structures and functional modules is a very hard task. In this paper we propose new data structures and a new implementation of a well known agglomerative greedy algorithm to find community structure in large networks, the CNM algorithm. The experimental results show that the improved data structures speedup the method by a large factor, for large networks, making it competitive with other state of the art algorithms.

1 Introduction

The problem of graph clustering or community finding has been extensively studied and, for the majority of the interesting formulations, this problem is NP-hard. Thus, in the study of large networks, fast approximation algorithms are required even though we may obtain suboptimal solutions. For a deep review on this topic, we refer the reader to a recent survey on community finding by Fortunato [1]. Here we revisit the modularity maximization problem, which is NP-hard [2], and a well known greedy approach proposed by Newman [3]. The simplest algorithm based on his approach runs in $O(n(n+m))$ time, or $O(n^2)$ for sparse graphs, where n is the number of vertices and m is the number of edges. More recently, Clauset *et al.* [4] exploited some properties of the optimization problem and, by using more sophisticated data structures, they proposed the *CNM algorithm* which runs in $O(md \log n)$ time in the worst case, where d is the depth of the dendrogram that describes the community structure.

In this paper we propose a new implementation of the CNM algorithm, using improved data structures. Although the asymptotic time bound is the same of the CNM algorithm, experimental results show a speed up of at least a factor of two. Moreover, we introduced randomization within our implementation which is useful to evaluate stability as different runs can provide different clusterings. The experimental evaluation includes several public available datasets and benchmarks. We also evaluate the performance of our implementation on large graphs generated with the partial duplication model [5]. The maximum modularity values obtained for these graphs are rather large, which is interesting given that these are random graphs. Finally, we briefly discuss the application and integration of this method with other measures and schemata.

2 Algorithm and data structures

The proposed algorithm starts with each vertex being the sole member of its community and then, iteratively, it merges pairs of communities that maximize the modularity score Q. Given a graph and a specific division of it into communities, modularity evaluates the difference between the fraction of edges that fall within communities and the expected fraction of edges within communities, if the edges were randomly distributed while respecting vertices degrees [6]. Let $G = (V, E)$ be an undirected graph and A its *adjacency matrix*, *i.e.*, $A_{uv} = 1$ if $(u, v) \in E$, and $A_{uv} = 0$ otherwise. Let n be the number of vertices and m be the number of edges of G. The *degree* d_u of a vertex $u \in V$ is given by $\sum_{v \in V} A_{uv}$. A *clustering* or *partition* $\mathcal{P}$ of G is a collection of sets $\{V_1, \ldots, V_k\}$, with $k \in \mathbb{N}$, such that $V_i \neq \emptyset$, for $1 \leq i \leq k$, $V_i \cap V_j = \emptyset$, for $1 \leq i < j \leq k$, and $\bigcup_{1 \leq i \leq k} V_i = V$. Given a partition $\mathcal{P}$ for G, we compute its *modularity* as

$$Q(\mathcal{P}) = \frac{1}{2m} \sum_{u,v \in V} \left[A_{uv} - \frac{d_u d_v}{2m} \right] \delta_{\mathcal{P}}(u, v), \tag{1}$$

where $m = |E|$ and the $\delta_{\mathcal{P}}$-function is such that $\delta_{\mathcal{P}}(u, v) = 1$ if both $u, v \in C$ for some $C \in \mathcal{P}$, $\delta_{\mathcal{P}}(u, v) = 0$ otherwise. The modularity Q_G of a graph G is defined as the maximum modularity over all possible graph partitions. Although Eq. (1) can take negative values, Q_G takes values between 0 and 1. Values near 1 indicate strong community structure and 0 is obtained for the trivial partition where all nodes belong to the same community. Typically, values for graphs with known community structure are in the range from 0.3 to 0.7 [6, 7].

Let $C_i, C_j \in \mathcal{P}_t$ be two communities, where $0 \leq i, j < |\mathcal{P}_t|$ and $\mathcal{P}_t$ is the partition achieved after $t \geq 0$ iterations. The change ΔQ_{ij} in Q after merging C_i and C_j to form a new partition $\mathcal{P}_{t+1}$ is given by manipulation of Eq. (1),

$$\Delta Q_{ij} = Q(\mathcal{P}_{t+1}) - Q(\mathcal{P}_t) = \frac{1}{2m} \, 2 \sum_{u \in C_i} \sum_{v \in C_j} \left[A_{uv} - \frac{d_u d_v}{2m} \right]. \tag{2}$$

Since calculating the ΔQ_{ij} for each pair C_i, C_j and for each iteration t becomes time-consuming, as in the original CNM algorithm, we store these values for each pair and only update them when needed. Given $C_i \in \mathcal{P}_t$, let $\bar{d}_i = \sum_{u \in C_i} d_u/(2m)$ and assume that $C_i, C_j \in \mathcal{P}_t$ are merged into $C_k \in \mathcal{P}_{t+1}$ at iteration $t+1$. Then, for $\mathcal{P}_{t+1}$, $\bar{d}_k = \bar{d}_i + \bar{d}_j$ and, for each C_ℓ adjacent to C_k,

$$\Delta Q_{k\ell} = \begin{cases} \Delta Q_{i\ell} + \Delta Q_{j\ell} & \text{if } C_\ell \text{ is connected to } C_i \text{ and } C_j, \\ \Delta Q_{i\ell} - 2\bar{d}_j \bar{d}_\ell & \text{if } C_\ell \text{ is connected to } C_i \text{ but not to } C_j, \\ \Delta Q_{j\ell} - 2\bar{d}_i \bar{d}_\ell & \text{if } C_\ell \text{ is connected to } C_j \text{ but not to } C_i. \end{cases} \tag{3}$$

These equations follow easily from Eq. (2). Communities are adjacent or connected if there is at least one edge between them. Note that merging two communities for which there is no connecting edge does not increase Q (when C_i is disconnected from C_l, the first term of Eq. (2) is zero and only the second one remains). Therefore, we will not store the value ΔQ for such pairs.

```
struct adj_node {
    int id;
    int u;
    int v;
    struct adj_node *u_nxt;
    struct adj_node *u_prv;
    struct adj_node *v_prv;
    struct adj_node *v_nxt;
};
```

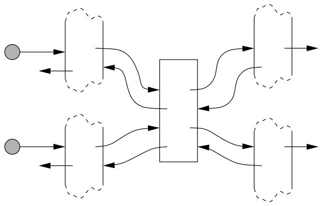

Fig. 1. Cross-linked adjacency list data structure. List nodes are defined by the `C` structure on the left and are linked as depicted on the right.

As described above, the algorithm starts with each vertex $u \in V$ being the sole member of a cluster. Let $C_u \in \mathcal{P}_0$ be such that $C_u = \{u\}$, for each $u \in V$. Then, for each $u \in V$ and $(u, v) \in E$, we initially set $\bar{d}_u = d_u/2m$ and $\Delta Q_{uv} = 1/m - 2\bar{d}_u\bar{d}_v$. Accordingly to Eq. (1), the initial value of Q is set to $Q = -\sum_{u \in V} \bar{d}_u\bar{d}_u$. The algorithm proceeds iteratively as follows:

1. select the pair (i, j) with maximum ΔQ_{ij};
2. merge C_i and C_j into C_k (assuming that we are in iteration $t + 1$, $\mathcal{P}_{t+1}$ is obtained from $\mathcal{P}_t$ replacing C_i and C_j by C_k);
3. update $\bar{d}_k$ and $\Delta Q_{k\ell}$ for each C_ℓ adjacent to C_k accordingly to Eq. (3);
4. update modularity Q by adding ΔQ_{ij};
5. repeat from step 1 until one community remains.

Here we are assuming that the graph is connected and that we are storing each partition $\mathcal{P}_t$ obtained at iteration t. If the graph is not connected, the algorithm stops when a pair (i, j) does not exist in step 1. Note also that we are usually interested in the partition that maximizes the modularity score. Thus, we can stop when a pair (i, j), selected in step 1, is such that $\Delta Q_{ij} < 0$. By Eq. (3), we know that ΔQ values can only decrease after a such pair be selected and, thus, the modularity value will not increase more since all ΔQ values are negative.

The main point is that we must find the maximum values and extract elements from the adjacency lists as fast as possible. Here we use a single heap data structure to store needed ΔQ values (at most m) and cross-linked adjacency lists to store community adjacencies. Since we have to both decrease and increase values in the heap (recall Eq. (3)), we use a binary heap data structure, for which the get maximum operation takes constant time and the insert, delete and update operations take $O(\log m)$ time, in the worst case. For community adjacencies, we use doubly-linked lists with cross references (see Fig. 1) and, thus, we can solve side effects in constant time when merging two adjacency lists.

Let c_u, c_r, c_t, c_ℓ be real constants. Updating a value in the heap takes $c_u \log m$ time and extracting a value takes $c_r \log m$. Thus, the extraction in step 1 takes $c_r \log m$ time at most and, since there are m elements in the heap, this step is repeated m times at most. Because we get a direct reference in step 1 and we have double-linked lists, removing the edge (i, j) from the community adjacency data structure in step 2 takes constant time. Step 2 requires also $3c_\ell n$ time to merge the adjacencies. Note that there are at most n adjacent communities to C_i and to C_j and that we can solve side effects in constant. Moreover, if a community C_k appears twice in the result, we only keep it once. Thus, to achieve linear time with unsorted lists, without loss of generality, we must process the adjacency of C_i, building a bit array of size n at most, and then process the adjacency of C_j, checking whenever a community C_k occurs in both adjacencies and updating the bit array. Finally we reprocess the adjacency of C_i in order to find the communities C_k which were not in the adjacency of C_j. Step 3 is done along with step 2 and each update takes $c_u \log m$ time at most. Therefore, step 3 takes less than $c_u n \log m$ time. Step 4 takes constant time. Although there exist m elements in the heap, steps 2-4 are executed at most $n-1$ times and, thus, the running time of the algorithm is at most $c_r m \log m + 3c_\ell n^2 + c_u n^2 \log m$, i.e., $O(n^2 \log n)$ time in the worst case assuming as usual that $m = O(n^2)$.

The differences between our algorithm and the CNM algorithm reside on how we store ΔQ values and how we manage the adjacency of the communities. The CNM algorithm stores ΔQ values in a sparse matrix with each row being stored both as a balanced binary tree and as a binary heap. It maintains also a binary heap containing the largest element of each row. Considering the same max-heap implementation and an efficient implementation of binary trees, updating an element takes $c_u \log n$ and extracting an element takes $c_r \log n$, where n is the maximum size of the heaps in this case. Thus, step 1 takes $c_r \log n$ time. Removing the selected pair from the community adjacency data structure in step 2 takes $2c_t \log n$ to update binary trees plus $2c_u \log n$ to update the heaps. Steps 2 and 3 require also $2n(c_t + 2c_u) \log n + c_u n$ time, since we must update the trees, the k-th heap and the main heap for each C_k in adjacency lists being merged. The heap associated with the resulting adjacency list can be updated in $c_u n$ time. Step 4 takes constant time. Since steps 2–3 are executed at most $n-1$ times, the running time of the CNM algorithm is at most $(c_r + 2c_t) n \log n + 2(c_t + 2c_u) n^2 \log n + c_u n^2$, *i.e.*, $O(n^2 \log n)$. Note that, although the asymptotic runtime bounds are the same, we get an improvement of at least a factor of two.

For sparse and hierarchical graphs we can provide a better upper bound. A graph $G = (V, E)$ is sparse if $m = O(n)$ and G is hierarchical if the resulting dendrogram for the community merging is balanced. In this case, the sum of the communities degrees at a given depth d is at most $2m$. Therefore the running time is at most $O(md \log n)$, where d is the depth of the dendrogram. Then for sparse and hierarchical graphs, since $m = O(n)$ and $d = O(\log n)$, the algorithm running time becomes $O(n \log^2 n)$. The space requirement of the algorithm is $O(n + m)$ as we store the connections for each community, a total of at most n communities and m connections, and m elements in the heap.

We also included in our implementation a randomized edge comparison function. As noted by Brandes *et al.* [2], the algorithm may perform badly if pairs with equal ΔQ are chosen in some crafted order. Although we cannot avoid undesired behavior by ordering these pairs randomly, we expect that it will not happen frequently. With respect to such fluctuations of modularity, we must mention that even small fluctuations may correspond to very different node clusterings [8]. Thus, several runs may be desirable to evaluate the stability of a given clustering, *i.e.*, how stable is vertex assignment along different runs.

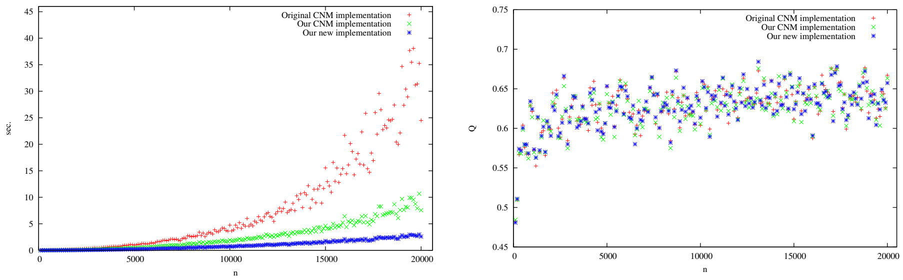

Fig. 2. Average running time and average maximum modularity Q for duplication model graphs obtained with $p = 0.5$. For each n were generated 10 random graphs. The number of edges for those graphs is about 10 times the number of vertices.

3 Experimental evaluation

In this section we consider 3 implementations in `C`, the original implementation of the CNM algorithm as provided by the authors, our implementation using optimized data structures to ensure fairness in the comparison and our new implementation. The running times below include the tracking of community membership. For that we use a disjoint sets data structure and, therefore, the running time cost is negligible. All implementations were compiled with the GNU `C` compiler with flag `-O3`. The experiments were conduced in a 2.33 GHz quad core processor with 16 GB of memory, running a GNU/Linux distribution.

In order to evaluate the performance on large networks, we generated artificial networks from the partial duplication model [5]. Although the abstraction of real networks captured by the partial duplication model, and other generalizations, is rather simple and no community structure is ensured, the global statistical properties of, for instance, biological networks and their topologies can be well represented by this kind of model [9]. For each number of vertices, we generated 10 random graphs with selection probability $p = 0.5$, which is within the range of interesting selection probabilities [5]. The number of edges for those graphs is approximately 10 times the number of vertices. Fig. 2 provides the running time of our implementation versus the running time of the CNM algorithm, where we observe an improvement of at least a factor of two. We ran also some tests with very large networks and, for a network with 1 million vertices and more than 13 millions edges, our new implementation takes about 9 hours and requires 744 MB of memory, while our implementation of the CNM algorithm takes 40 hours and requires 1,796 MB. In Section 4 we discuss how prioritizers can further reduce the running time. Although this model does not ensure any community structure, note that the values of modularity are usually higher than 0.5 (see Fig. 2). This is an interesting fact that deserves a better understanding.

Given that in our algorithm we pick randomly a pair whenever two pairs have the same ΔQ value, we evaluated our implementation on several public

Table 1. Maximum and minimum modularity for 4 real networks after 1,000 runs. $|V|$ is the number of vertices and $|E|$ is the number of edges for each network. $\max Q$ is the maximum modularity, $\min Q$ is the minimum modularity and $\#\mathcal{P}$ is number of different partitions obtained for 1,000 runs.

Network	$\|V\|$	$\|E\|$	$\min Q$	$\max Q$	$\#\mathcal{P}$
Zachary's karate club [10]	34	78	0.381	0.381	1
Bottlenose dolphins'network [11]	62	159	0.492	0.495	2
C. elegans metabolic network [12]	454	2,025	0.385	0.413	253
Protein interaction network [13]	2,215	2,203	0.842	0.846	770

datasets and benchmarks, focusing on the stability of the obtained clusterings. Table 1 provides details for four real networks. Unsurprisingly, Q values are identical to those reported by the CNM algorithm. But the partitions found for the last two networks are rather unstable, namely for the protein interaction network where in 1,000 runs 770 different partitions were found. Although we did not analyse further these networks, our results raise an important question concerning partition stability. This is an important issue in the study of networks and, until now, most of the analyses in the literature just consider one partition.

4 Discussion

There are alternative approaches for the greedy optimization of modularity. Schuetz and Caflisch [14] proposed an approach where they merge at once ℓ disjoint pairs of communities instead of just one pair and which can benefit from improvements proposed here. More recently, Blondel *et al.* [15] proposed an alternative greedy approach. The algorithm proceeds by alternating two main steps. In the first step, it iteratively considers a vertex, removes it from the current cluster computing the change in modularity, and then selects the cluster that provides the better improvement by moving the vertex to that cluster. This is repeated until no change occurs. The second step consists of building a coarsened graph where each cluster becomes a vertex. Then, we iterate these two steps while there are edges in the coarsened graph. Although the authors do not provide a theoretical bound, the running time seems to be almost linear from the experiments, making it one of the fastest algorithms to date. The improvements proposed in this paper make the CNM algorithm competitive with that algorithm, if not faster. With respect to clustering quality, Noack and Rotta [16] stated that the most effective method consists of a multilevel schema, where the greedy approach studied here is used for the coarsening phase, and the first step of the method proposed by Blondel *et al.* for the refinement phase. They used also prioritizers to improve both the running time and the clustering quality.

One of the first prioritizers was proposed by Wakita and Tsurumi [17], favoring the merge of equal size communities, enforcing the running time bound of $O(n \log^2 n)$ for sparse graphs. Since Newman and Girvan [6] proposed the modularity score to account for the intra-cluster density versus the inter-cluster sparsity, given two clusters or communities C_i and C_j, a natural prioritizer is $\varDelta Q_{ij}/(d(C_i)d(C_j))$, that favors the merge of clusters with lower weight density, conducting to more dense clusters on average. Although previous studies pointed in this direction [18], only recently Noack and Rotta [16] explicitly used this prioritizer and the variant $\varDelta Q_{ij}/\sqrt{d(C_i)d(C_j)}$. Although both prioritizers are related, the second one is closely tied to the null model underlying the modularity measure. By Eq. (2), $2m\varDelta Q_{ij}$ is the difference between the observed and the expected number of edges between C_i and C_j. On the other hand, since the null model assumes a binomial distribution, we know that for large graphs the variance of the number of edges between C_i and C_j is approximately $d(C_i)d(C_j)/(2m)$. Thus, the second prioritizer accounts for the number of standard deviations between the observed and the expected number of edges between C_i and C_j. We ran our algorithm with this prioritizer for the network with 13 million edges and we obtained an outstanding speedup, it takes now 130 seconds instead of 9 hours. Also, the values of the modularity did not decrease, as expected, given the modularity definition and its close relation with this prioritizer. As observed before [19, 17], the reason for the speedup is that, without any prioritizer, the greedy approach merges the cluster with the largest contribution to modularity with its best neighbor. The strength of the cluster increases and the process continues until all good neighbors are merged. But, since this cluster has great influence, several bad neighbors may also be merged before any other merge pairs be considered. This produces very unbalanced partitions taking the running time up to the upper bound $O(n^2 \log n)$. Note also that the modularity value may be lower since some bad neighbors were merged. In fact, after considering the prioritizer, we obtained higher modularity values. For instance, for the Zachary's karate social network we achieved a value of 0.419. The most important conclusion from our work is that, with the new implementation and considering good prioritizers, we are able to effectively process real large scale-free networks and evaluate its community structure stability.

Several measures have been proposed to evaluate clusterings quality, in particular because modularity suffers some resolution problems [20, 21]. Thus, it is important to note that the optimization approach discussed in this paper can be easily adapted for other measures and, in general, it is sufficient to rewrite Eq. (3). This is straightforward for measures based on modularity, such as the modularity for weighted graphs or the similarity-based modularity [22]. For other measures, the updates after each merging step may require some more careful analysis. Nevertheless, we can employ this greedy method either alone or combined with other approaches. For instance, Rosvall and Bergstrom [23, 24] used recently this approach in their study of mutual information and of maps of random walks to uncover community structure. In their work they improve the final results by using a simulated annealing based approach [24].

References

1. Fortunato, S.: Community detection in graphs. Physics Reports **486** (2010) 75–174

2. Brandes, U., *et al.*: On finding graph clusterings with maximum modularity. In: Graph-Theoretic Concepts in Computer Science. Volume 4769 of LNCS., Springer (2007) 121–132
3. Newman, M.E.J.: Fast algorithm for detecting community structure in networks. Physical Review E **69** (2004) 066133
4. Clauset, A., Newman, M.E.J., Moore, C.: Finding community structure in very large networks. Physical Review E **70** (2004) 066111
5. Chung, F., Lu, L., Dewey, T.G., Galas, D.J.: Duplication models for biological networks. Journal of Computational Biology **10**(5) (2003) 677–687
6. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Physical Review E **69** (2004) 026113
7. Guimerà, R., Sales-Pardo, M., Amaral, L.A.N.: Modularity from fluctuations in random graphs and complex networks. Physical Review E **70**(2) (2004) 025101
8. Agarwal, G., Kempe, D.: Modularity-maximizing communities via mathematical programming. The European Physical Journal B **66**(3) (2008) 409–418
9. Bhan, A., Galas, D.J., Dewey, T.G.: A duplication growth model of gene expression networks. Bioinformatics **18**(11) (2002) 1486–1493
10. Zachary, W.W.: An information flow model for conflict and fission in small groups. Journal of Anthropological Research **33** (1977) 452–473
11. Lusseau, D., *et al.*: The bottlenose dolphin community of Doubtful Sound features a large proportion of long-lasting associations. Behavioral Ecology and Sociobiology **54**(4) (2003) 396–405
12. Duch, J., Arenas, A.: Community identification using extremal optimization. Physical Review E **72** (2005) 027104
13. Jeong, H., Mason, S., Barabási, A.L., Oltvai, Z.N.: Centrality and lethality of protein networks. Nature **411**(6833) (2001) 41–42
14. Schuetz, P., Caflisch, A.: Efficient modularity optimization by multistep greedy algorithm and vertex mover refinement. Physical Review E **77**(4) (2008) 46112
15. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics **P10008** (2008)
16. Noack, A., Rotta, R.: Multi-level algorithms for modularity clustering. In: Experimental Algorithms. Volume 5526 of LNCS., Springer (2009) 257–268
17. Wakita, K., Tsurumi, T.: Finding community structure in mega-scale social networks. In: International World Wide Web Conference, ACM (2007) 1275–1276
18. Reichardt, J., Bornholdt, S.: Statistical mechanics of community detection. Physical Review E **74** (2006) 016110
19. Danon, L., Duch, J., Diaz-Guilera, A., Arenas, A.: The effect of size heterogeneity on community identification in complex networks. Journal of Statistical Mechanics **P11010** (2006)
20. Fortunato, S., Barthélemy, M.: Resolution limit in community detection. PNAS **104**(1) (2007) 36–41
21. Kumpula, J.M., Saramäki, J., Kaski, K., Kertész, J.: Limited resolution in complex network community detection with Potts model approach. The European Physical Journal B - Condensed Matter and Complex Systems **56**(1) (2007) 41–45
22. Feng, Z., Xu, X., Yuruk, N., Schweiger, T.A.J.: A novel similarity-based modularity function for graph partitioning. In: Data Warehousing and Knowledge Discovery. Volume 4654 of LNCS., Springer (2007) 385–396
23. Rosvall, M., Bergstrom, C.T.: An information-theoretic framework for resolving community structure in complex networks. PNAS **104**(18) (2007) 7327
24. Rosvall, M., Bergstrom, C.T.: Maps of random walks on complex networks reveal community structure. PNAS **105**(4) (2008) 111–118