Computing Communities in Large Networks Using Random Walks
L42: Assessment 1

Jan Ondras (jo356), Trinity College
Word count: 2,418 words*

November 29, 2017

Abstract

Nowadays, when we deal with large complex networks in various domains such as biology, sociology or computer science, the detection of communities (dense subgraphs of sparse graphs) in such networks is of vital importance. However, computing these communities is usually expensive and not straightforward. This report provides a summary of the paper [1] that proposes a random-walk based algorithm Walktrap to efficiently find clusters in graphs. Also, extensive comparison tests with other algorithms and results from experiments on randomly-generated as well as real-world networks are presented. One additional community detection algorithm and two real-world networks beyond the scope of the original paper are considered.

Contents

*this word count was computed by `texcount -inc report.tex`

Overview

The first part of this report describes the paper's main contributions, key mathematical concepts, as well as its limitations. The second part shows my implementation of the Walktrap algorithm, results from its evaluation and various experiments, and conclusions drawn from these outcomes.

The following notation and terminology will be used throughout this report.

- Undirected graph $G = (V, E)$ where V and E are sets of vertices and edges respectively. ($n = |V|$ and $m = |E|$ denote total numbers of vertices and edges in G respectively.)
- Community C is a highly interconnected subgraph with few links to external vertices.
- Partition $P = \{C_1, ..., C_i\}$ describes a particular community structure of the graph.

1 Main contributions

The work *Computing Communities in Large Networks Using Random Walks* [1] by Pons and Latapy has the following contributions:

- New random-walk based algorithm called Walktrap that computes community structure of a given graph. Its worst-case complexity is $O(mn^2)$ time and $O(n^2)$ space, whereas in most real-world cases $O(n^2 \log n)$ time and $O(n^2)$ space
- New measure (r) of similarity between vertices (and between communities) based on random walks. In contrast to previous approaches [2], this distance metric is very efficient to compute and it can well capture information on the community structure.
- New metric (η) to assess the quality of partition of a graph into communities. This metric is able to identify relevant partitions at various scales.
- Extensive comparison tests showing that Walktrap surpasses previous approaches in terms of the quality of the resulting community structures and is among the best ones in terms of running time.
- The paper also provides substantial proofs and relations with spectral approaches.

2 Key concepts

The **key idea** of the proposed approach is that random walks on a graph tend to get stuck in densely connected subgraphs, i.e. communities. This suggests that random walks can be used to determine the distance metric between vertices and between communities. Consequently, the obtained distances can be used by agglomerative clustering algorithm to compute hierarchical community structure, graphically represented by a tree called *dendogram*.

This approach has a clear **advantage** over spectral methods [3] in terms of efficiency. It avoids expensive explicit computation of eigenvectors which becomes untractable for larger graphs and requires $O(n^3)$ time even for sparse matrices.

2.1 Random walks

Random walk on graph G is driven by its transition matrix P that can be obtained from the graph adjacency matrix A. The random walk of length t is then represented by matrix P^t where $P^t_{ij} = (P^t)_{ij}$ is the probability of going from vertex i to j in t steps. The parameter t should be set so that the random walk is long enough to gather enough information about the topology of the graph, but not too long to avoid reaching the stationary distribution.

2.2 Distance metric r

The probabilities P_{ij}^t then determine the vertex similarity measure between vertex i and j as:

$$r_{ij} = \left\| D^{-\frac{1}{2}} P_{i\bullet}^t - D^{-\frac{1}{2}} P_{j\bullet}^t \right\| \tag{1}$$

where $||.||$ is the Euclidean norm of $\Re^n$, $P_{i\bullet}^t$ is the i^{th} row of P^t, and D is the diagonal matrix of nodes' degrees. The distance r_{ij} is large if the two vertices belong to different communities and small otherwise. Similarly, we can define the distance $r_{C_1C_2}$ between communities C_1 and C_2:

$$r_{C_1C_2} = \left\| D^{-\frac{1}{2}} P_{C_1\bullet}^t - D^{-\frac{1}{2}} P_{C_2\bullet}^t \right\| \tag{2}$$

where $P_{C\bullet}^t$ is the probability vector defined by entries

$$P_{Cj}^t = \frac{1}{|C|} \sum_{i \in C} P_{ij}^t$$

that define the probability of going from community C to vertex j in t steps.

2.3 Hierarchical clustering

The next step is to use these distances to perform hierarchical clustering. At each step we merge two adjacent communities C_1 and C_2 that minimize[1] the variation $\Delta\sigma(C_1, C_2)$ among all pairs of communities in the current partition.

$$\Delta\sigma(C_1, C_2) = \frac{1}{n} \frac{|C_1||C_2|}{|C_1| + |C_2|} r_{C_1C_2}^2 \tag{3}$$

This quantity determines the increase in the mean of the squared distances between each vertex and its community if we merged C_1 and C_2. It is important to notice that the fact that r represents an Euclidian distance makes the computation of $\Delta\sigma$ very efficient.

After merging C_1 and C_2 we need to update the values $\Delta\sigma(C_3, C)$ between C_3 and all affected communities C, that is, communities previously adjacent to C_1 and/or C_2. It can be shown that if C_1 and C_2 were both adjacent to C, then the $\Delta\sigma(C_3, C)$ can be computed efficiently using

$$\Delta\sigma(C_3, C) = \frac{(|C_1| + |C|)\Delta\sigma(C_1, C) + (|C_2| + |C|)\Delta\sigma(C_2, C) - |C|\Delta\sigma(C_1, C_2)}{|C_3| + |C|} \tag{4}$$

otherwise, Equation (3) must be used.

2.4 Quality of a partition

At each merging step k the algorithm generates a partition P_k. In order to choose the best partition P_* we need to define a measure of quality of a partition. The most commonly used such measure is modularity Q, defined as:

$$Q(P_k) = \sum_{C \in P_k} \frac{I_C}{G_w} - \left(\frac{T_C}{G_w}\right)^2 \tag{5}$$

where I_C and T_C are internal and total[2] weights of community C respectively, and G_w is the total weight of graph edges. Self-edges do not count here. The best partition P_* is then chosen as $P_* = \arg\max_{P_k} Q(P_k)$.

- **Advantage:** suitable for finding an unique partition.

[1] for efficiency the values of $\Delta\sigma$ are stored in min-heap datastructure

[2] weights of edges between communities contribute half to each community

- **Disadvantage:** not well suited to find communities of different scales.

Therefore, to address this limitation of modularity, the paper proposes a new quality criterion η defined as:

$$\eta(P_k) = \frac{\Delta\sigma_k}{\Delta\sigma_{k-1}} \tag{6}$$

where $\Delta\sigma_k$ is the minimal $\Delta\sigma$ that determined which communities to merge at iteration k. The best partition P_* is then chosen according to $P_* = \arg\max_{P_{k-1}} \eta(P_k)$. The intuition here is that if we merge two very different communities (in terms of distance r), then the value $\Delta\sigma_k$ at this step is large which suggests that the community structure at step $k-1$ was appropriate.

- **Advantage:** can reveal relevant community structures at different scales.
- **Disadvantage:** may require another criterion (e.g. community size) to choose among the best partitions identified.

2.5 Comparing partitions

If we want to evaluate how well the obtained partition P_* matches the ground truth partition P_G, a comparison of their modularities is not a reliable method (as we will see later). Instead, a comparison metric such as corrected Rand-index R' is used:

$$R'(P_*, P_G) = \frac{n^2 \sum_{i,j} |C_i^* \cap C_j^G|^2 - \sum_i |C_i^*|^2 \sum_j |C_j^G|^2}{\frac{1}{2}n^2\big(\sum_i |C_i^*|^2 + \sum_j |C_j^G|^2\big) - \sum_i |C_i^*|^2 \sum_j |C_j^G|^2} \tag{7}$$

where $(C_i^x)_{1\leq i\leq k_x}$ are communities of the partition P_x. Compared to the commonly used "ratio of vertices correctly identified", the quantity R' is more sensitive because it can capture similarities between partitions even if they differ in the number of communities.

3 Limitations

In this section, I briefly summarized consequences of assumptions made and other limitations of the proposed method.

- Inability to handle directed graphs since the provided proofs for random walks are not valid in such cases.
- Cannot deal with extremely large graphs (millions of vertices) due to its high memory requirements. For such graphs other approaches like Fast Modularity [4] might be preferred.
- The algorithm generates only non-overlapping communities, because of the nature of the agglomerative clustering.

4 Implementation

4.1 Algorithm

Following the concepts presented in Section 2 I used Python to implement the Walktrap algorithm (Algo. 1) and Networkx library[5] for graph representation and visualization. I also implemented the calculation of modularity Q, quality of partition η, and corrected Rand-index R' (for evaluation).

Algorithm 1 Walktrap pseudo-code

```
1:  procedure WALKTRAP(Graph G, t)
2:      P^t ← take t steps of random walk on G
3:      partitions[0] ← initialize first partition
4:      initialize Δσ-min-heap
5:      for k in 1:(n-1) do
6:          Δσ(C_1, C_2), C_1, C_2 ← Δσ-min-heap.extract_min()
7:          C_3 ← merge(C_1, C_2)
8:          partitions[k] ← create new partition from partitions[k − 1] adding C_3 and removing C_1, C_2
9:          for C in C_3.neighbours do
10:             if C ∈ C_1.neighbours && C ∈ C_2.neighbours then
11:                 Δσ(C, C_3) ← apply Equation (4)
12:             else
13:                 Δσ(C, C_3) ← apply Equation (3)
14:             Δσ-min-heap.push(Δσ(C, C_3))
15:     return partitions
```

4.2 Complexity

4.2.1 Time

The initialization steps (lines 2–4) take at most $O(tn^2)$ time (or $O(tnm)$ assuming P is sparse). Then, we iterate $O(n)$ times: choosing two communities to merge, merging them, creating a new partition, and updating $\Delta\sigma$-min-heap. The complexity of the outer loop is dominated by the inner loop that is executed at most $O(m)$ times thanks to the heuristic that we merge only adjacent communities. Each iteration of the inner loop involves calculation of $\Delta\sigma$ that can be performed in $O(1)$ time if Equation (4) can be applied. Otherwise Equation (3) must be used taking $O(n)$ time. Therefore, the overall time complexity is $O(mn^2)$.

4.2.2 Space

The storage of probability vectors and $\Delta\sigma$-min-heap requires $O(n^2)$ and $O(m)$ space respectively. Thus, the overall space complexity is $O(n^2)$.

5 Evaluation

To test my implementation and perform experiments I used both random and real-world graphs. I also compared my Walktrap implementation with the following methods:

- More recent algorithm for community detection by Louvain [6] which is based on direct modularity optimization. For this I used a freely available Python package *python-louvain*[3].
- Markov Clustering algorithm [7], available from Python module *markov_clustering*[4]. Expansion and inflation coefficients were set to 2.

5.1 Sample graph

First, I checked correctness of my implementation on the 16-node graph (Fig. 1) shown in the paper and obtained the same results in terms of partition, modularities, and η-s.

[3] https://github.com/taynaud/python-louvain/
[4] https://github.com/GuyAllard/markov_clustering

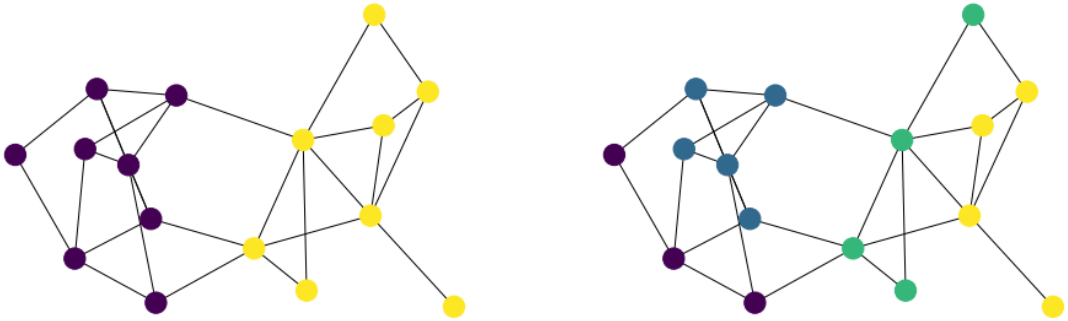

Figure 1: Sample 16-node graph (from the paper) with 2 communities. Left: partition obtained by Walktrap with $t = 2, 5, 8$. Right: partition obtained by Louvain's algorithm.

Subsequently, I examined the effect of parameter t on partitioning (Fig. 2). As we can see in this case the parameter t does not affect the best partition chosen neither by modularity nor by η. However, we can notice that the graph of η indicates (by a small peak at $k = 12$) that a partition at another scale might be of some relevance as well, namely, $K = 4$, which was actually chosen by Louvain's algorithm as the best partition in this case.

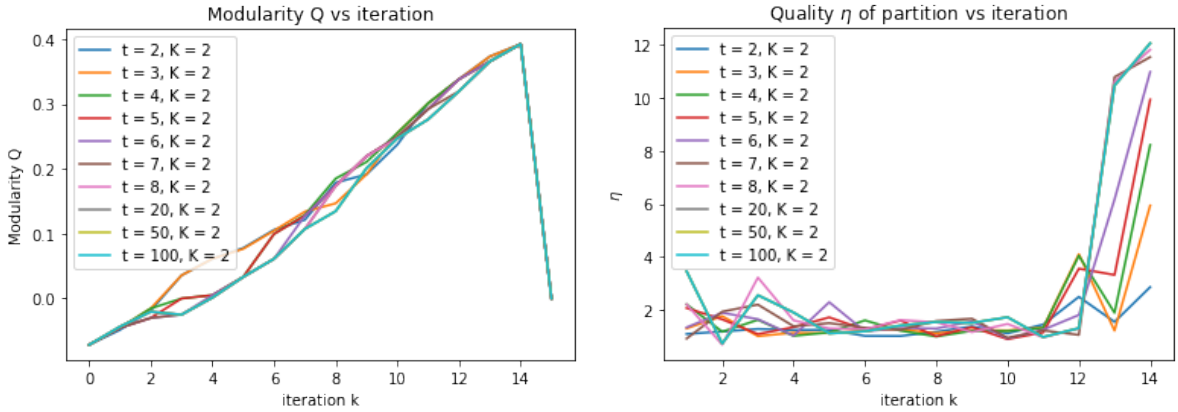

Figure 2: Effect of varying t on partitioning for both measures of partition's quality: modularity Q (left) and η (right). K is the number of communities when Q (or η) is maximized.

5.2 Real-world graphs

Next, I examined the following real-world graphs[5] where the ground truths[6] (actual number K' of communities) were available.

- **Zachary's karate club network [9]:** $K' = 2$, $n = 33$
- **American college football network [10]:** $K' = 12$, $n = 115$
- **Les Miserables coappearance network [11]:** $K' = 6$, $n = 77$
- **Dolphin social network [12]:** $K' = 2$, $n = 62$

[5] downloaded from `http://www-personal.umich.edu/~mejn/netdata/`
[6] provided in [8]

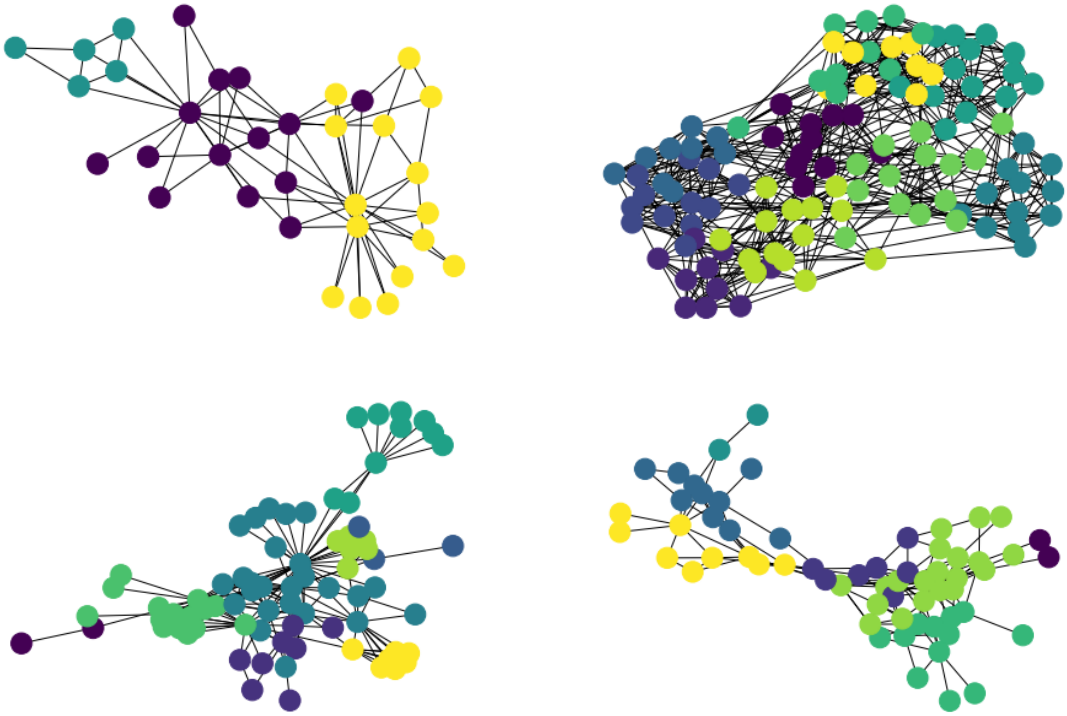

Figure 3: Communities identified by Walktrap algorithm with $t = 5$. Zachary's karate club network (top left), American college football network (top right), Les Miserables coappearance network (bottom left), Dolphin social network (bottom right).

For each of these networks I performed the following experiments:

1.) Investigation of the influence of Walktrap's parameter t on Q and η values.

2.) Comparison of the three algorithms Walktrap (WT), Louvain's (LO), and Markov Clustering (MC), in terms of identified number K of communities, modularity Q, and running time t_r.

5.2.1 Experiment 1: influence of parameter t

In the first experiment, I looked at the plots of $Q(k)$ and $\eta(k)$ while varying $t \in \{2, 3, 4, 5, 6, 7, 8, 20, 50, 100\}$. All the graphs are shown in Fig. 4. From the plots of modularity we can see that higher t values achieved lower maximum in modularity and that in general the values of K do not match K' very well. When looking at plots of $\eta(k)$ the value $t = 2$ produced very good results for Karate and Football networks, but surprisingly high values were suitable for Dolphin network. This confirms that η is capable of extracting relevant partitions at different scales, in this case it was able to detect structures that modularity metric did not discover.

The value of t that produced K closest to the truth K' is not consistent among the networks for neither Q nor η. It seems that there is no general rule how to set the t value, it might be a graph-dependent parameter.

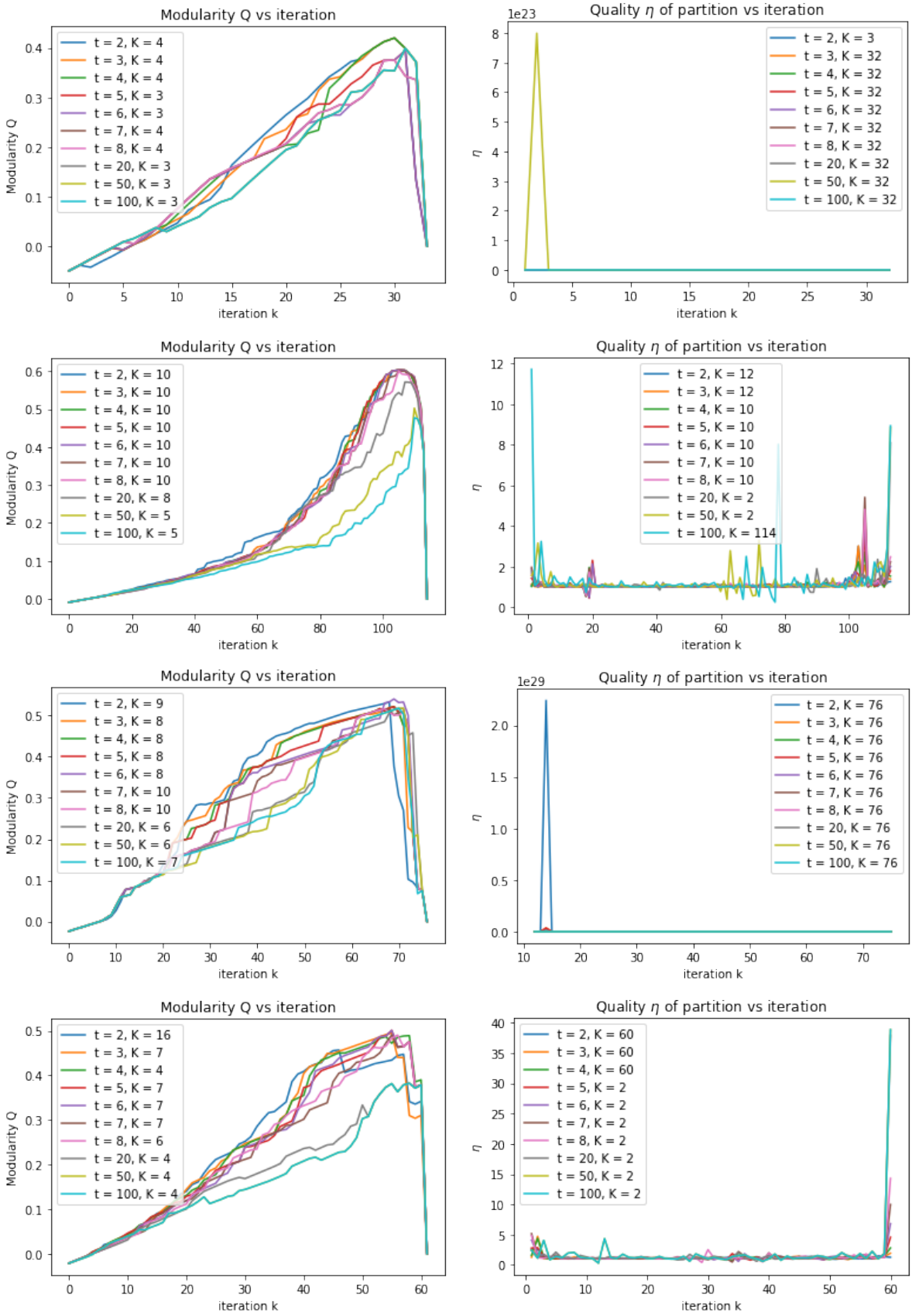

Figure 4: Influence of Walktrap's parameter t on Q and η values in four real-world networks (from top): Zachary's karate club network, American college football network, Les Miserables coappearance network, and Dolphin social network. Ground truth number of communities (from top): 2, 12, 6, 2.

5.2.2 Experiment 2: comparison of algorithms

The results of the second experiment comparing various algorithms are summarized in Fig. 5. We can clearly notice that the fastest algorithm was LO.

Zachary's karate club network			
Algorithm	K	Q	t_r[ms]
WT 2	4	0.420	14.0
WT 5	3	0.394	11.9
WT 8	4	0.375	11.0
LO	4	0.511	3.2
MC	2	0.360	20.3
Ground truth	2	–	–

American college football network			
Algorithm	K	Q	t_r[ms]
WT 2	10	0.603	183.5
WT 5	10	0.603	201.1
WT 8	10	0.601	193.0
LO	11	0.651	35.2
MC	12	0.601	37.5
Ground truth	12	–	–

Les Miserables coappearance network			
Algorithm	K	Q	t_r[ms]
WT 2	9	0.533	104.6
WT 5	8	0.521	101.7
WT 8	10	0.516	95.1
LO	9	0.624	14.9
MC	5	0.415	22.9
Ground truth	6	–	–

Dolphin social network			
Algorithm	K	Q	t_r[ms]
WT 2	16	0.457	38.1
WT 5	7	0.501	33.7
WT 8	6	0.490	31.5
LO	7	0.602	6.3
MC	12	0.455	26.7
Ground truth	2	–	–

Figure 5: Comparison of algorithms WT 2, WT 5, WT 8, LO, and MC on four real-world networks in terms of the number K of detected communities, modularity Q, and running time t_r.

Since these networks do not significantly differ in size, I summarized the average relative errors (defined as $\frac{1}{4}\sum_{i\in networks}\frac{|K'_i-K_i|}{K'_i}$) of these algorithms over all 4 networks into Tab. 1. We can conclude that Walktrap with $t = 3$ was most successful in terms of similarity between K and K' on these networks. However, it is difficult to objectively judge the quality of the partitions obtained since the ground truths provide only the number K' of communities and not the partition itself. When comparing two partitions it might be the case that similar values of Q or same values of K represent completely different partitions. It is thus challenging to perform evaluation on real-world networks where the true partition P_G is not specified.

Algorithm	WT 2	WT 3	WT 5	LO	MC
average relative error	2.166	0.875	0.9583	1.02	1.29

Table 1: Relative errors of algorithms averaged over Karate, Football, Les Miserables, and Dolphin networks.

5.3 Random graphs

In order to compare algorithms more objectively I performed experiments on random graphs where the ground truth partition P_G is known. Namely, I generated[7] *Random partition graphs* by specifying the following parameters:

- the list $l = [s_1, \ldots, s_{K'}]$ of sizes of (non-overlapping) communities,
- the probability p_{in} that two nodes within a community are connected,
- and the probability p_{out} that two nodes between communities are connected.

[7] using the NetworkX library [5]

By specifying the sizes s_i of communities I was able to generate both *homogeneous* and *heterogeneous* graphs. During the experiments I also ensured that I deal only with connected graphs.

5.3.1 Homogeneous graphs

I compared the performance of the five above-mentioned algorithms (WT 2, WT 5, WT 8, LO, and MC) on graphs of various sizes $n \in \{30, 100, 200, 300, 500, 700, 1000\}$ using the corrected Rand-index R', since the true partition P_G was known. To get more reliable estimates and lessen the effect of a particular setting of the graph-generator parameters, for each n I averaged the obtained values of R' over graphs corresponding to all combinations of the following parameters:

- ground truth number of communities $K' = n^\gamma$ with $\gamma \in \{0.3, 0.4, 0.5\}$ such that all communities have equal size: $\forall i, j \in \{1, \ldots, K'\}.s_i = s_j$
- $p_{in} \in \{0.4, 0.5\}$
- $p_{out} \in \{0.1, 0.2\}$

In the paper they only considered graphs where external degrees of communities were proportional to their internal degrees. However, while I was experimenting with the graph-generator parameters I noticed high variability of the results, especially when the above constraint was not met. Therefore, I decided not to make this constraint in a hope to get more generalizable results.

Fig. 6 shows that Louvain's algorithm performs slightly better than Walktrap and Markov Clustering on smaller graphs, whereas Walktrap might be preferred on larger graphs. Poor performance of MC algorithm is probably caused by the fact that it relies on a granularity parameter that is usually set manually for each graph. Furthermore, the results suggest that on smaller graphs smaller values of t (such as $t = 2$) may result in better performance. However, more experiments with wider ranges of network parameters would be required to draw some general conclusions if any.

The histogram on the right side of the Fig. 6 confirms that comparisons based on $K \stackrel{?}{=} K'$ can be misleading (e.g. for $n = 30$ when comparing relative performance of the algorithms using R' (left) versus using accuracies (right)).

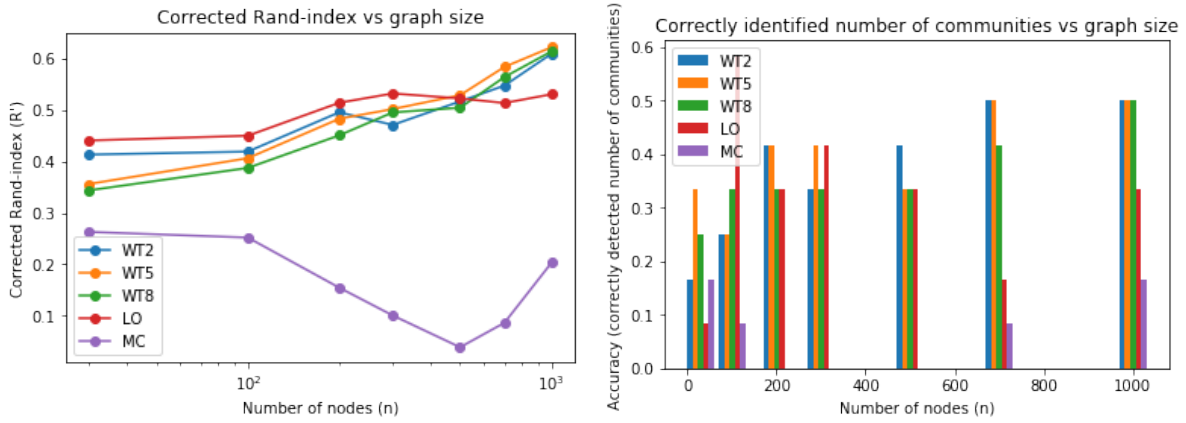

Figure 6: Comparison of algorithms WT 2, WT 5, WT 8, LO, and MC on random *homogeneous* graphs in terms of the corrected Rand-index R' (left) and success rate given by $K \stackrel{?}{=} K'$ (right).

5.3.2 Heterogeneous graphs

I followed the same procedure as for homogeneous graphs with the only difference that the sizes s_i of communities were chosen randomly obeying the constraint $\sum_{i=1}^{K'} s_i = n$. Comparing the Figures 7 and 6 we can see that graph heterogeneity did not significantly affect the results, except for a small indication

that smaller values of t might be preferred even for larger networks in the presence of heterogeneity, but again, it is challenging to generalize this.

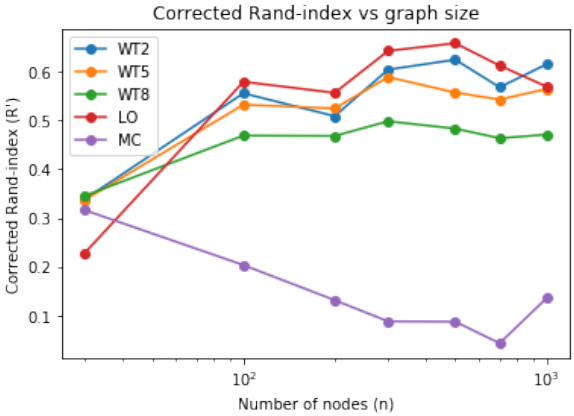

Figure 7: Comparison of algorithms WT 2, WT 5, WT 8, LO, and MC on random *heterogeneous* graphs in terms of the corrected Rand-index R'.

6 Conclusions

6.1 Summary

In this report I tried to reproduce the work by Pons [1] on community detection using random walks on graphs in terms of both theory and experiments, adding some extensions.

My findings from the experiments can be summarized as follows:

- I verified the paper's findings that
  - the partition quality metric η can reveal relevant community structures at various scales (even if the use of modularity fails to identify correct partition, as was the case of real-world networks studied),
  - partitions with similar values of modularity can represent considerably different community structures,
  - the evaluation of partition's quality based on $K \stackrel{?}{=} K'$ can be misleading,
  - the heterogeneity of graphs does not significantly influence the Walktrap's performance,
  - and there seems to be no general rule on how to set Walktrap parameter t.
- In addition to the original paper, I further
  - compared Walktrap with a more recent algorithm by Louvain,
  - performed comparison tests on another two real-world graphs (Les Miserables and Dolphin networks),
  - and investigated the effect of Walktrap's parameter t more closely, namely, considering more values of t I found that:
    - in the case of real-world networks examined, higher t results in lower maximum in modularity,
    - and in the case of heterogeneous random networks smaller t such as $t = 2$ might produce better results for networks of various sizes.

6.2 More recent work

Currently, with increasing success of machine learning methods, the community detection task can be approached in a new way, for instance, using graph neural networks [13].

Also, addressing one of the limitations of Walktrap, the detection of overlapping community structures seems to be another popular research area these days, both using random-walk based approach [14], [15] as well as other methods [16], [17].

References

[1] Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks. *J. Graph Algorithms Appl.*, 10(2):191–218, 2006.

[2] Stanley Wasserman and Katherine Faust. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

[3] Luca Donetti and Miguel A Munoz. Detecting network communities: a new systematic and efficient algorithm. *Journal of Statistical Mechanics: Theory and Experiment*, 2004(10):P10012, 2004.

[4] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.

[5] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008.

[6] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[7] S VAN Dongen. *Graph clustering by flow simulation.* PhD thesis, Standardization and Knowledge Transfer, 2000.

[8] Mark EJ Newman and Gesine Reinert. Estimating the number of communities in a network. *Physical review letters*, 117(7):078301, 2016.

[9] Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977.

[10] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[11] Donald Ervin Knuth. *The Stanford GraphBase: a platform for combinatorial computing*, volume 37. Addison-Wesley Reading, 1993.

[12] David Lusseau, Karsten Schneider, Oliver J Boisseau, Patti Haase, Elisabeth Slooten, and Steve M Dawson. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology*, 54(4):396–405, 2003.

[13] Joan Bruna and Xiang Li. Community detection with graph neural networks. *arXiv preprint arXiv:1705.08415*, 2017.

[14] Zhiyong Yu, Jijie Chen, Kun Quo, Yuzhong Chen, and Qian Xu. Overlapping community detection based on random walk and seeds extension. In *Proceedings of the 12th Chinese Conference on Computer Supported Cooperative Work and Social Computing*, pages 18–24. ACM, 2017.

[15] TianRen Ma, Zhengyou Xia, and Fan Yang. An ant colony random walk algorithm for overlapping community detection. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 20–26. Springer, 2017.

[16] Xuyun Wen, Wei-Neng Chen, Ying Lin, Tianlong Gu, Huaxiang Zhang, Yun Li, Yilong Yin, and Jun Zhang. A maximal clique based multiobjective evolutionary algorithm for overlapping community detection. *IEEE Transactions on Evolutionary Computation*, 21(3):363–377, 2017.

[17] Lei Zhang, Hebin Pan, Yansen Su, Xingyi Zhang, and Yunyun Niu. A mixed representation-based multiobjective evolutionary algorithm for overlapping community detection. *IEEE Transactions on Cybernetics*, 2017.