A Fast Algorithm to Find Overlapping Communities in Networks

Steve Gregory

Department of Computer Science
University of Bristol, BS8 1UB, England
steve@cs.bris.ac.uk

Abstract. Many networks possess a community structure, such that vertices form densely connected groups which are more sparsely linked to other groups. In some cases these groups overlap, with some vertices shared between two or more communities. Discovering communities in networks is a computationally challenging task, especially if they overlap. In previous work we proposed an algorithm, CONGA, that could detect overlapping communities using the new concept of *split betweenness*. Here we present an improved algorithm based on a *local* form of betweenness, which yields good results but is much faster. It is especially effective in discovering small-diameter communities in large networks, and has a time complexity of only O(n log n) for sparse networks.

1 Introduction and Related Work

In recent years, networks (graphs) have increasingly been used to represent various kinds of complex system in the real world. Many networks exhibit *community structure*: the tendency of vertices to form *communities* (or *modules*) such that intracommunity edges are denser than the edges between communities. Communities often reflect important relationships between individuals (vertices), so the automatic discovery of communities has become one of the key tasks in network analysis.

Even if we restrict our attention to unipartite networks with undirected, unweighted edges, as we do in this paper, there is already a wide choice of community detection algorithms. Many of these are described in the survey papers of [6, 14], and there are also many recent algorithms, including [3, 17, 21, 24, 27].

Unfortunately, there is no standard definition of *community* and no consensus about how a network should be divided into communities. The vast majority of existing algorithms partition a network into a *flat* set of *disjoint* sets (clusters) of vertices, though it is often possible, or necessary, to choose the number of clusters. However, in some networks the community structure is not flat: for example, a collaboration network may contain a community for each research area, each comprising a number of subcommunities corresponding to research groups. A few algorithms [5, 11] can detect such a hierarchical community structure. Moreover, in many networks, communities are not disjoint: for example, some researchers work on more than one topic and therefore belong simultaneously to multiple research groups. Some algorithms [2, 9, 20, 28] are able to detect these overlapping communities.

In this paper we focus on the detection of overlapping communities. In previous work we designed an algorithm, CONGA (Cluster-Overlap Newman Girvan Algorithm)

W. Daelemans et al. (Eds.): ECML PKDD 2008, Part II, LNAI 5212, pp. 408–423, 2008.

[9], for this purpose. It extends Girvan and Newman's [8, 18] algorithm (the "GN algorithm") with the ability to split vertices between clusters, based on the new concept of *split betweenness*. CONGA yields good results but is extremely slow, with approximately cubic time complexity, so it can only cope with networks containing at most a few thousand vertices and edges. Many real-world networks are far larger than this, so CONGA cannot be used.

CONGA inherits its low speed from the GN algorithm. Both algorithms rely on *betweenness*, which is a *global* centrality measure: at each step, it counts the number of shortest paths between *all* pairs of vertices in the network. For a fast, scalable, algorithm we need a measure that can be computed *locally*. In this paper we show how CONGA can be made much faster using *local betweenness* [10, 23].

In the next section we outline CONGA, introduce the concept of local betweenness, and then describe our new algorithm: the CONGO (CONGA Optimized) algorithm. Section 3 presents the results of experiments with the new algorithm on both synthetic and real-world networks. We compare its performance and execution time with both CONGA and CFinder [20], another state-of-the-art algorithm for finding overlapping communities. Conclusions appear in Section 4.

2 The CONGO Algorithm

2.1 The CONGA Algorithm

The CONGA algorithm [9] comprises a sequence of steps, each of which removes an edge from the network *or* splits a vertex into two vertices:

1. Calculate edge betweenness of edges and split betweenness of vertices.
2. Remove edge with maximum edge betweenness or split vertex with maximum split betweenness, if greater.
3. Recalculate edge betweenness and split betweenness.
4. Repeat from step 2 until no edges remain.

Initially, the n-vertex network is treated as a single cluster, assuming it is connected. Eventually, step 2 causes the cluster to split into two components (clusters). Clusters continue to be split into two until only singleton clusters remain. The binary splits can be represented as a dendrogram, which is used to reconstruct a partition of the network into any desired number of clusters.

CONGA is the same as the GN algorithm [8, 18] except for the vertex splitting step, which allows overlapping clusters. Because of this, a vertex v may be split into i vertices (copies of v) distributed between j clusters ($1 \leq j \leq i$). When reconstructing the partition, these copies of v are replaced by v itself in each of these j clusters.

The edge betweenness of an edge e is the number of shortest paths, between all pairs of vertices, that pass along e. The split betweenness of a vertex v is the number of shortest paths that *would* pass between the two parts of v if it were split. Since there are many ($2^{d(v)-1}$-1, where $d(v)$ is the degree of v) ways to split v into two, we choose the *best split*: the one that maximizes the split betweenness.

In [9] we give an approximate algorithm for computing the split betweenness of a vertex from its pair betweennesses. The *pair betweenness* of v for $\{u,w\}$, where u and w are neighbours of v, is the number of shortest paths traversing both edges $\{u,v\}$ and $\{v,w\}$. It is straightforward to compute this while computing edge betweenness.

The GN algorithm has a worst-case time complexity of $O(m^2n)$, where m is the number of edges and n is the number of vertices. In CONGA, each vertex v can split into at most m/n vertices on average (i.e., $d(v)/2$), so the number of vertices after splitting is $O(m)$ instead of n. This makes the time complexity $O(m^3)$ in the worst case: there are $O(m)$ iterations, and both step 1 and step 3 are $O(m^2)$.

In practice, the speed depends heavily on the number of vertices that are split (which increases the network size) and on how easily the network breaks into separate components. This is because, in step 3, betweenness need be calculated only for the component containing the removed edge or split vertex, or for both components if step 2 caused the component to split.

2.2 Local Betweenness

Betweenness is expensive to compute because it counts *all* shortest paths in the network. One way to avoid this is to count only *short* shortest paths. We redefine the edge betweenness of edge e to be the number of shortest paths running along e whose length is less than or equal to h (a parameter of the algorithm). The *pair betweenness* of vertex v for $\{u,w\}$ is the number of shortest paths traversing $\{u,v\}$ and $\{v,w\}$ whose length is less than or equal to h. *Split betweenness* is derived from pair betweennesses in the same way as in CONGA.

Step 1 of the CONGA algorithm is performed by a breadth-first search from every vertex. Using local betweenness, the depth of this search (from each vertex) is limited to h, which is faster than traversing every edge in the network.

Local betweenness has an even greater effect on the speed of step 3: betweenness need not be recalculated for the whole network, but only locally: in a small subgraph around the edge that was removed, or the vertex that was split, in step 2. In Figs. 1 and 2 we illustrate how step 3 of CONGA can be optimized in this way, but first we need to define this small subgraph, which we call an h-region.

The *h-region* of edge $\{u,v\}$ — the region affected by the removal of $\{u,v\}$ — is the smallest subgraph containing all shortest paths no longer than h that pass along $\{u,v\}$. This is an induced subgraph with vertex set

$$V_{u,v,h} = \{w : d(u,w) < h \vee d(v,w) < h\} \tag{1}$$

where $d(u,w)$ denotes the shortest-path distance between u and v. The *h-region* of vertex v — the region affected by splitting v — is the smallest subgraph containing all shortest paths no longer than h that pass through v or start/end at v. This is an induced subgraph with vertex set

$$V_{v,h} = \{w : d(v,w) \leq h\} \tag{2}$$

For example, Fig. 1(a) shows a small network with the 2-region of edge {h,i} shown shaded. Fig. 2(a) shows another network, highlighting the 2-region of vertex u.

(a) Edge {h,i} selected for removal. 2-region of {h,i} is shaded.

(b) Shortest paths within region are found and subtracted from betweenness.

(c) {h,i} is removed. Shortest paths within region are found and added to betweenness.

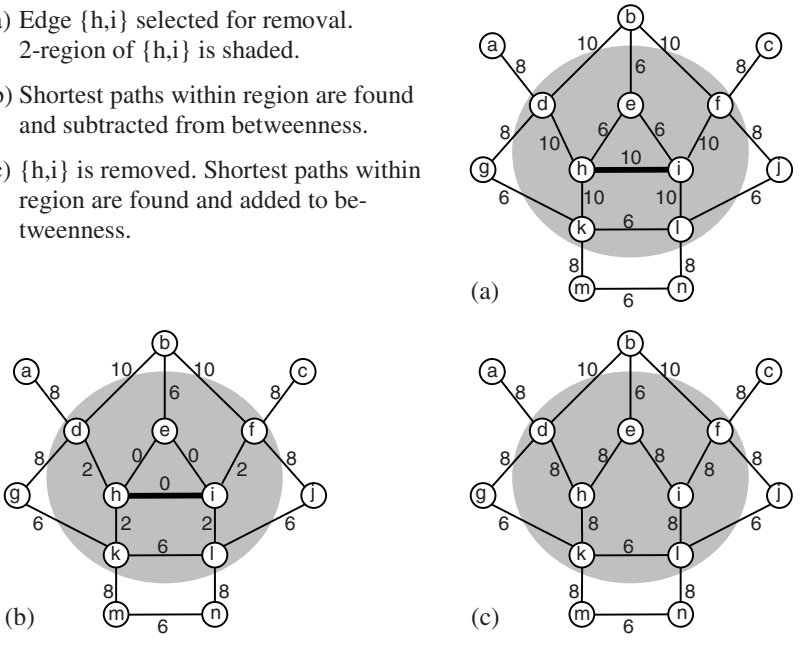

Fig. 1. Local recalculation of betweenness after removing an edge

(a) Vertex u selected for splitting. 2-region of u is shaded.

(b) Shortest paths within region are found and subtracted from betweenness.

(c) u is split. Shortest paths within region are found and added to betweenness.

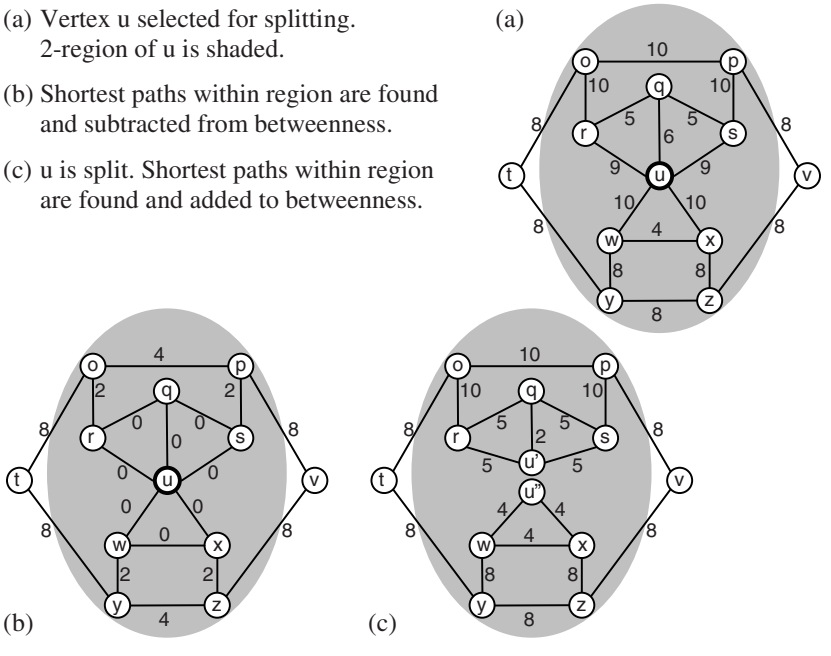

Fig. 2. Local recalculation of betweenness after splitting a vertex

In step 3, our new algorithm recalculates betweenness by a local method. It first "undoes" the betweenness of the *h*-region, by finding all shortest paths no longer than *h* that lie entirely within the region and subtracting their number from the (previously computed) edge betweenness of the edges they traverse and the pair betweennesses of the vertices they pass through. This has the effect of reducing the betweenness of the chosen edge (Fig. 1(b)) or vertex (Fig. 2(b)) to zero. After removing the edge or splitting the vertex, it again finds all shortest paths no longer than *h* within the region and adds their number to the edge betweenness of the edges they traverse and the pair betweennesses of the vertices they pass through; see Figs. 1(c) and 2(c).

2.3 The CONGO Algorithm

The CONGO algorithm is the same as CONGA (Section 2.1) but using local betweenness, explained in Section 2.2. The complete CONGO algorithm is as follows:

1. Calculate edge betweenness of edges and split betweenness of vertices.
2. Find edge with maximum edge betweenness or vertex with maximum split betweenness, if greater.
3. Recalculate edge betweenness and split betweenness:
   a) Subtract betweenness of *h*-region centred on the removed edge or split vertex.
   b) Remove the edge or split the vertex.
   c) Add betweenness for the same region.
4. Repeat from step 2 until no edges remain.

In practice, CONGO's execution time depends strongly on the structure of the network, but we can estimate its time complexity as follows.

For step 1, the time complexity for $h=\infty$ is $O(mn)$; this would reduce to $O(m)$ for $h=1$, which is of no practical use because the 1-betweenness of every edge is the same. For other small values of *h*, we make the simplifying assumption that all vertices have about the same degree, $2m/n$. Then, for each of the *n* vertices, the tree searched contains $O((m/n)^h)$ vertices. This makes the time complexity of step 1 approximately $O(m^h/n^{h-1})$, or $O(n)$ for a sparse network.

Making the same assumption for step 3, an *h*-region of an edge contains $O((m/n)^{h-1})$ vertices and $O((m/n)^h)$ edges; an *h*-region of a vertex contains $O((m/n)^h)$ vertices and $O((m/n)^{h+1})$ edges. Therefore, the time complexity of step 3 is approximately $O((m/n)^{2h+1})$, or $O(1)$ for a sparse network.

Step 2 takes $O(\log m)$ time, and the loop containing steps 2 and 3 is repeated $O(m)$ times. Therefore, the time complexity of the whole algorithm is $O(m \log m + m^{2h+2}/n^{2h+1})$, or $O(n \log n)$ for a sparse network.

3 Experiments

3.1 Experiments on Synthetic Networks

A common way to evaluate network clustering algorithm is by generating artificial networks based on a known community structure and comparing the known *communities*

with the *clusters* found by the algorithm. The comparison can be done in various ways, including the Mutual Information measure [7] and Rand Index [22]. We use the *F-measure*, defined as the harmonic mean of recall and precision, where:

- *recall*: the fraction of vertex pairs belonging to the same community that are also in the same cluster.
- *precision*: the fraction of vertex pairs in the same cluster that also belong to the same community.

(A pair of vertices are considered to "belong to the same community/cluster" if there exists *at least one* community/cluster that they *both* belong to. Because of overlap, there may be more than one of these.)

We randomly generated a set of networks containing n vertices divided into c equally-sized communities, each containing nr/c vertices. Vertices are randomly and evenly distributed between communities such that each vertex is a member of r communities on average. r is a measure of overlap, ranging from 1 (communities are disjoint) to c (communities each contain all vertices). The network is then constructed by placing edges between pairs of vertices randomly, with probability ip_{in} if there are i ($\geq$1) communities to which both vertices belong, and p_{out} otherwise. All networks used in the experiments are connected, and results shown are the average of 10 runs.

Below we compare our CONGO algorithm, for h=2 and h=3, with CONGA (which is equivalent to CONGO with h=∞). We also compare it with CFinder [1], based on the clique percolation algorithm of Palla *et al*. [20], one of the most efficient and best-known algorithms for finding overlapping communities. For CONGO and CONGA, the number of clusters is a parameter of the algorithm, so we set this to c, the known number of communities. This is impossible with CFinder, whose only parameter is k (cluster density). For fairness, we show the results from CFinder for *all* values of k.

Fig. 3 shows results for 256 vertices in 32 communities. The overlap is 2, meaning that each community contains 16 vertices. As p_{out} increases, the community structure becomes less evident and so CONGO's F-measure decreases, especially for h=2. In contrast, CFinder is relatively resilient to these intercommunity edges.

Fig. 4 shows the effect of increasing the density of intracommunity edges. The CONGO results are good and improve as p_{in} increases. Again, h=2 is worse than h=3, but only for small values of p_{in}. CFinder, for each k, peaks at a different value of p_{in}.

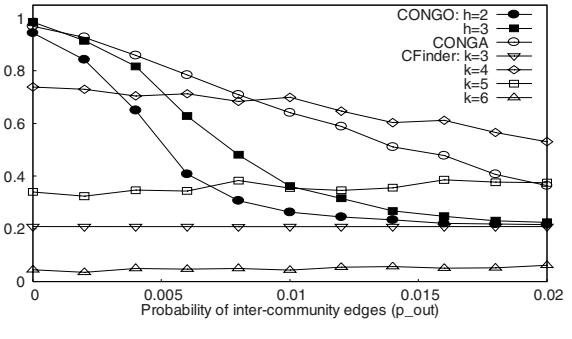

Fig. 3. F-measure for random networks with n=256, c=32, r=2, p_{in}=0.5, various p_{out}

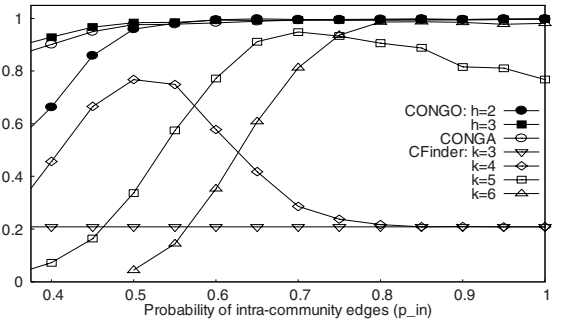

Fig. 4. F-measure for random networks with n=256, c=32, r=2, p_{out}=0, various p_{in}

In Fig. 5 we fix p_{in} and p_{out} and vary the overlap, r. CONGO's results decline as r gets larger, especially for h=2, while CFinder's results again peak at a different value of r for each k.

Fig. 6 shows the effect of varying the network size while keeping the community size constant. As expected, the F-measure results are quite stable, except for very small networks, because these contain very few communities.

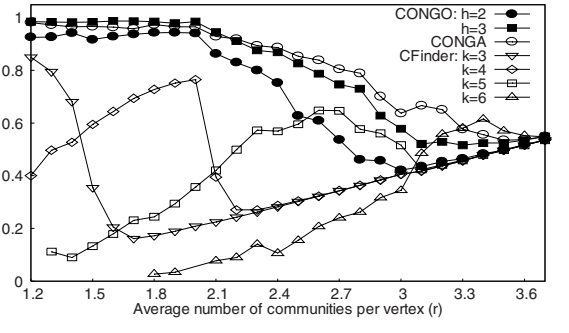

Fig. 5. F-measure for random networks with n=256, c=32, p_{in}=0.5, p_{out}=0, various r

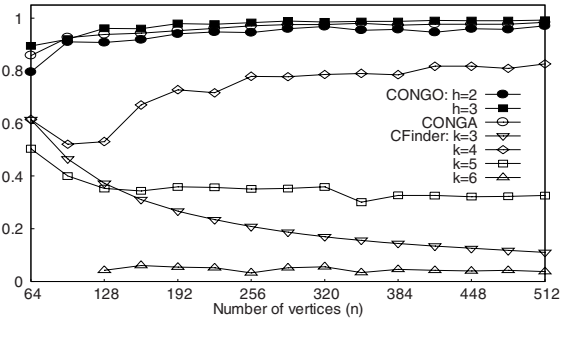

Fig. 6. F-measure for random networks with c=n/8, r=2, p_{in}=0.5, p_{out}=0, various n

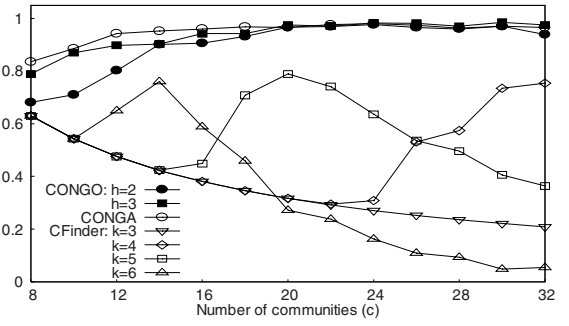

Fig. 7. F-measure for random networks with n=256, r=2, p_{in}=0.5, p_{out}=0, various c

In Fig. 7 we fix the size of the network but vary the number (and therefore size) of the communities. CONGO again performs well, but slightly less well for h=2.

In summary, on synthetic networks, CONGO's results are similar to those of CONGA: generally better than CFinder except where there are intercommunity edges. CONGO, like CONGA, treats an intercommunity edge as evidence that the two communities overlap. The difference between CONGO and CONGA is that a smaller value of h leads to slightly lower accuracy (F-measure).

Figs. 8 and 9 show the effect of local betweenness on execution time. For CONGO and CONGA, the plots show the time taken to compute the entire dendrogram and extract the clustering from it, using the author's implementation of the algorithms in Java, running on an AMD Opteron 250 CPU at 2.4GHz. For CFinder (v1.21), the times include the generation of solutions for all values of k, on the same machine.

Fig. 8 shows the time to cluster networks of varying size containing fixed-size (16) overlapping communities. The figure shows the approximately cubic time complexity of CONGA, which can only handle 2000 vertices in 20 minutes. CFinder is faster, taking only 15 minutes to cluster a 30000-vertex network. However, CONGO can cope with 500000 vertices in about 10 (h=2) or 20 (h=3) minutes. The CONGO results seem to confirm the O(n log n) time complexity that we predicted.

In Fig. 8 the community size is fixed and so the average degree is constant: about 5 (shown by the dashed line). This is not always the case in real networks. Fig. 9 shows how the algorithms scale in the extreme case: where there is an increasing number of vertices divided into a fixed number (12) of communities. Now, CONGO's execution time increases with the number of edges, but much more slowly than CFinder's.

3.2 Modularity

Evaluating an algorithm on real-world networks is challenging, because there is usually no known "correct" solution. The quality of clusterings must be assessed in a different way: for example, by modularity [17, 18], which measures the relative number of intracluster and intercluster edges. A high modularity indicates that there are more intracluster edges than would be expected by chance.

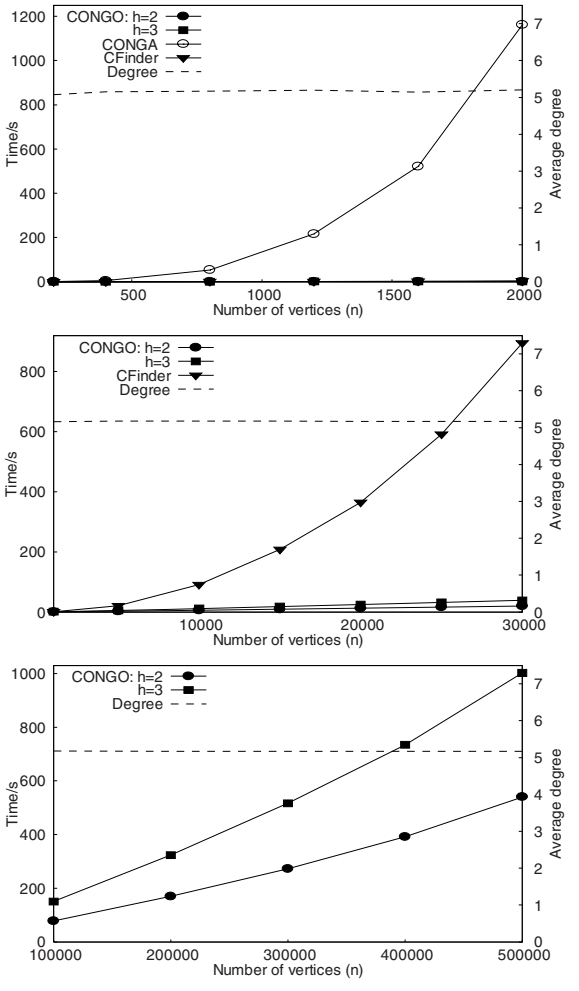

Fig. 8. Execution time for random networks $c=n/8$, $r=1.2$, $p_{in}=0.5$, $p_{out}=0$, various n

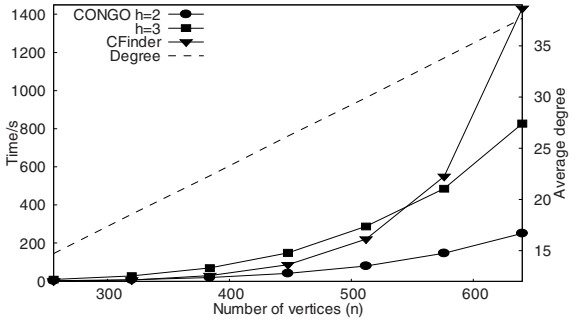

Fig. 9. Execution time for random networks $c=12$, $r=1.2$, $p_{in}=0.5$, $p_{out}=0$, various n

The original modularity measure, Q, is defined only for disjoint communities, but Nicosia *et al*. [19] have recently proposed a new modularity measure, Q_{ov}, which *is* defined also for overlapping communities. The definition of Q_{ov} is too long to reproduce here, but its main features are:

1. $Q_{ov} = 0$ when all vertices belong to the same community or all belong to singleton communities.
2. Higher values of Q_{ov} show stronger community structure.
3. Each vertex may belong to any number of communities with any *belonging coefficient*. For each vertex, the belonging coefficients for all communities sum to 1.

Although both old and new measures are named "modularity", they generally have different values even when applied to the same clustering and network. In the rest of this paper we use the term "modularity" to refer to Q_{ov}. We use it in this section to evaluate solutions on real-world networks. For our experiments we set the belonging coefficients of each vertex to $1/c$, where c is the number of communities it is in; i.e., equal membership of all communities.

It is sometimes assumed that the best clustering is the one that maximizes the value of modularity. The maximum value of modularity has even been used to compare clustering algorithms. However, as pointed out in [10], the peak value of modularity does *not* in general coincide with the correct, or best, clustering. This is true of both Q and Q_{ov} modularity measures.

To illustrate this, Fig. 10 shows the modularities of the solutions found by CONGO and CFinder for a synthetic network with 250 overlapping communities. For CONGO, modularity peaks at 0.822 between 45 and 53 clusters, where the F-measure is below 0.1. At 250 clusters, the correct solution, with F-measure 0.977 (h=3) or 0.891 (h=2), the modularity is only 0.701 (h=3) or 0.705 (h=2). CFinder finds a solution with 291 clusters with modularity 0.635 and F-measure 0.877.

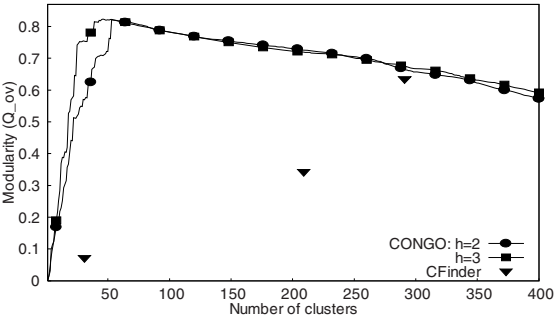

Fig. 10. Modularity of clusterings of random network: n=2000, c=250, r=1.2, p_{in}=0.5, p_{out}=0

Fig. 11 shows the results of a similar network with a larger overlap. CONGO's Q_{ov} at 250 clusters is 0.25 for both h=3 and h=2, and the F-measure is 0.998 (h=3) or 0.992 (h=2). Although Q_{ov} has a local maximum at 250 clusters, it is well below the global maximum between 63 and 79 clusters, where the F-measure is below 0.001. The closest CFinder solution is at 290 clusters, with Q_{ov} 0.142 and F-measure 0.842.

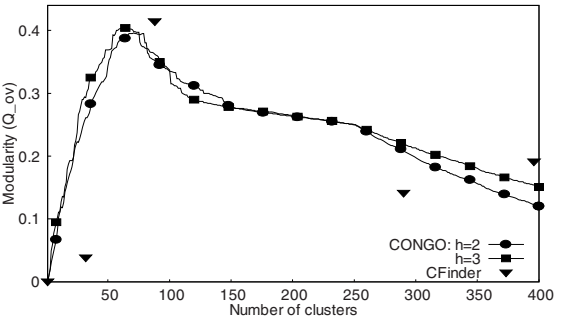

Fig. 11. Modularity of clusterings of random network: n=2000, c=250, r=2, p_{in}=0.5, p_{out}=0

We conclude that there is at best a tenuous relationship between modularity and correctness. Nevertheless, because modularity is widely used to assess clustering algorithms, we use it in the next section to evaluate the performance of CONGO (and CFinder) on some real-world networks. Because the peak value of modularity is meaningless, we plot the modularity of *all* solutions containing up to $n/5$ clusters.

3.3 Experiments on Real-World Networks

We have run the CONGO algorithm, and CFinder, on several real-world networks, listed in Table 1. The table shows the source of each network, its size, and the execution times for CONGO (to compute the entire dendrogram) and CFinder (v1.21) (to generate solutions for all values of k), running on an AMD Opteron 250 at 2.4GHz.

Table 1. Results on real-world networks

Name	Ref.	Fig.	Vertices	Edges	Runtime / s		
					CONGO		CF
					h=3	h=2	
netscience	[16]	12	379	914	1.4	1.3	0.3
cond-mat-2003	[13]	13	27519	116181	45110	1111	1140
blogs	[26]	14	3982	6803	33.5	6.1	3.2
blogs2	[26]	15	30557	82301	11702	286	405
PGP	[4]	16	10680	24316	636	82	35022
word_association	[12]	17	7205	31784	12026	172	97
protein-protein	[20]	18	2445	6265	94.5	8.2	2.9

"netscience" (Fig. 12) and "cond-mat-2003" (Fig. 13) are collaboration networks of coauthorships, of different sizes. The first of these is small enough for CONGA to handle, so its modularities are plotted along with those of CONGO. CONGA finds high-modularity solutions for small numbers of clusters, but is otherwise similar to CONGO for h=2 and h=3. For both networks, the modularities of the h=2 and h=3 solutions are

quite similar. CFinder finds several solutions, one of which, for a relatively large number of clusters, has a higher modularity than CONGO's.

"blogs" (Fig. 14) and "blogs2" (Fig. 15) are networks of communication relationships between owners of blogs on the MSN (Windows Live™) Spaces website. "blogs2" is much larger than "blogs" and has a higher average degree. "PGP" (Fig. 16) is yet another type of social network, representing PGP key signing. For all three networks, the modularities are quite similar to those of Figs. 12 and 13.

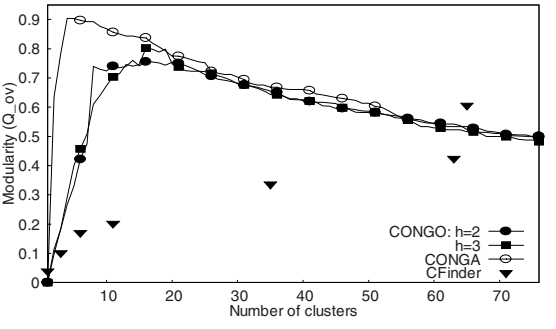

Fig. 12. Modularity of netscience network

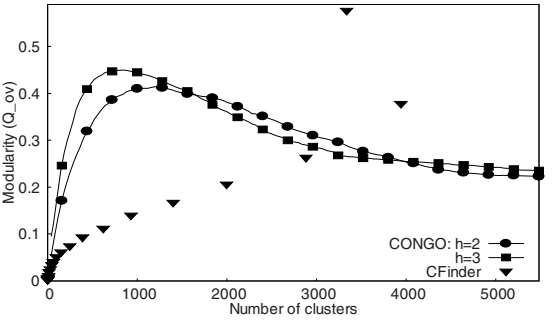

Fig. 13. Modularity of cond-mat-2003 network

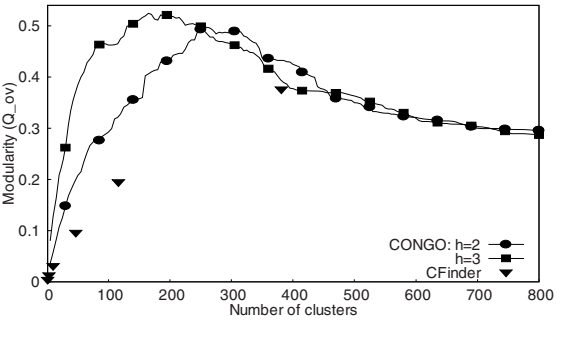

Fig. 14. Modularity of blogs network

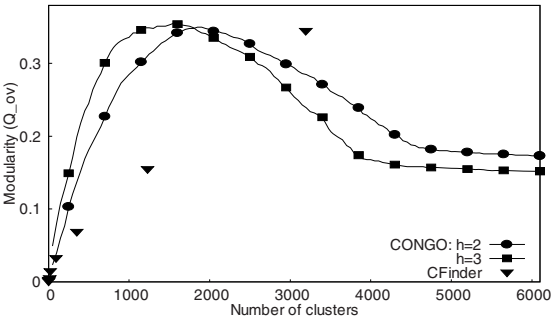

Fig. 15. Modularity of blogs2 network

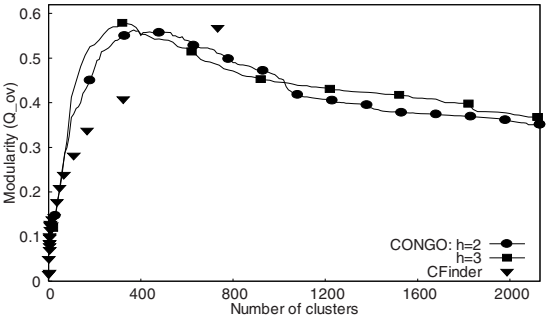

Fig. 16. Modularity of PGP network

Figs. 17 and 18 show two non-social networks, from psychology and biology, respectively. "word_association" is a word association network from [20], converted from an original directed, weighted version [12]. "protein-protein" is a yeast core protein-protein interaction graph provided by [20]. For the first of these, CFinder finds a higher modularity solution than CONGO for a small number of clusters.

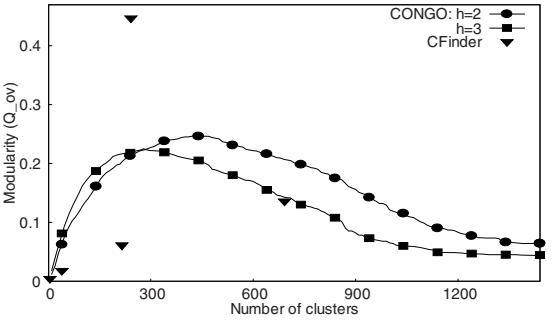

Fig. 17. Modularity of word_association network

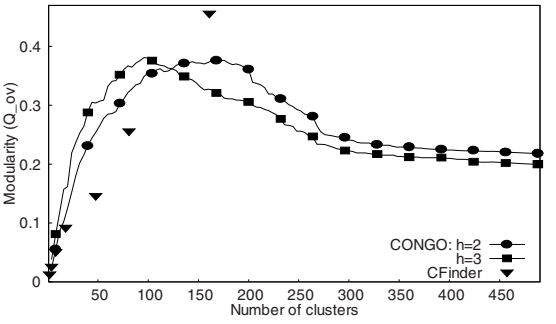

Fig. 18. Modularity of protein-protein network

Finally, we present an experiment on a real-world example with a known solution. Wirz [25] has constructed an ego-graph of his "friends" on Facebook, the social network website. An *ego-graph* of v is a network whose vertices are the "friends" of v and whose edges are the "friend" links between these vertices; v itself is excluded from the network in this experiment. The network has 84 vertices, in five components, and 351 edges. Wirz has manually classified the 84 vertices into ten communities and three isolated vertices, based on personal judgement and without knowledge of the network structure. CONGA, CONGO, and CFinder were run on this network.

Table 2 shows the results for ten clusters (the correct number), seven clusters (the clustering that maximizes modularity), and six clusters (at which CFinder found its highest-modularity solution, for k=3). CONGO, for h=3 or more, has higher F-measure and modularity than CFinder; for h=2, CONGO performs slightly worse than CFinder, but has higher modularity for 11 or more clusters (not shown in the table).

Table 2. Results on ego-graph network

F-measure

	c=10	**c=7**	**c=6**
CONGA	0.353	0.310	0.225
h=3	0.345	0.310	0.225
h=2	0.281	0.209	0.225
CFinder			0.303

Modularity

	c=10	**c=7**	**c=6**
CONGA	0.704	0.912	0.886
h=3	0.722	0.912	0.886
h=2	0.607	0.834	0.886
CFinder			0.858

4 Conclusions

The results presented in Section 3 show that CONGO can be an effective and fast algorithm for detecting overlapping communities in networks. Compared with CONGA, it is substantially faster, especially for h=2. Indeed, almost all of the real-world networks used in Section 3 are too large for CONGA to process in a reasonable time. CONGO is slightly less accurate than CONGA in most, but not all, cases. As discussed in [10], we believe this is because local betweenness is unable to correctly identify communities whose diameter is much larger than h. When communities have a small diameter, CONGO can be at least as accurate as CONGA.

Compared with CFinder, CONGO has similar execution time. CFinder is faster for small examples but CONGO, with h=2, is faster for larger networks. For synthetic networks, CONGO appears generally more accurate than CFinder. For real-world networks, modularity is usually better for CONGO, but CFinder often finds one solution with high modularity: this is for k=3 in all cases except the "jazz" network. Looking at these high-modularity solutions in more detail reveals a few large clusters and many small (mostly 3-vertex) clusters; in addition, there are many vertices that appear in no clusters. The sizes of CONGO's clusters are generally more balanced. It is hard to say which type of solution is best, except perhaps by comparing the computed clusterings with a "ground truth" solution, as in the "ego-graph" network of Section 3.

In conclusion, we believe that CONGO with h=2 is ideally suited to finding overlapping communities in very large networks. For smaller networks, where a solution can be found quickly, h can be increased for more accurate results.

Future work includes improving the CONGO algorithm. One issue is the value of the parameter. h=2 is fast and usually effective, but sometimes a larger value is required; h=3 is sometimes more effective, but too slow in general. The best value for h seems to depend on the diameter of communities, which might vary widely in real-world networks. It may be better to allow the length of shortest paths to vary dynamically in different parts of a network to suit the diameter of each community.

Another possible improvement is to introduce "belonging coefficients", showing how strongly each vertex belongs to each cluster. This should make solutions more informative than in the current algorithm, in which vertices belong equally to all their clusters. For example, partitioning a network into {1,2,3} and {1,2,3,4,5,6} seems meaningless, but if vertices 1, 2, and 3 belong more strongly to the first cluster than the second, the solution is more like two clusters {1,2,3} and {4,5,6} that overlap.

Further information related to this paper, including the implementation, networks analysed, and results, can be found at http://www.cs.bris.ac.uk/~steve/networks/ .

Acknowledgements. I am very grateful to Peter Flach for comments on a draft of this paper. Thanks are also due to Vincenzo Nicosia for explaining the details of his extended modularity measure, and to Martin Wirz for providing his ego-graph data.

References

1. Adamcsek, B., Palla, G., Farkas, I., Derényi, I., Vicsek, T.: CFinder: Locating Cliques and Overlapping Modules in Biological Networks. Bioinformatics 22, 1021–1023 (2006)
2. Baumes, J., Goldberg, M., Magdon-Ismail, M.: Efficient Identification of Overlapping Communities. In: Kantor, P., Muresan, G., Roberts, F., Zeng, D.D., Wang, F.-Y., Chen, H., Merkle, R.C. (eds.) ISI 2005. LNCS, vol. 3495, pp. 27–36. Springer, Heidelberg (2005)
3. Blondel, V.D., Guillaume, J.-L., Lambiotte, R., Lefebvre, E.: Fast Unfolding of Community Hierarchies in Large Networks. Eprint arXiv:0803.0476v1 at arxiv.org (2008)
4. Boguña, M., Pastor-Satorras, R., Diaz-Guilera, A., Arenas, A.: Phys. Phys. Rev. E 70, 056122 (2004)
5. Clauset, A., Moore, C., Newman, M.E.J.: Structural Inference of Hierarchies in Networks. In: Statistical Network Analysis: Models, Issues, and New Directions, pp. 1–13 (2007)

6. Danon, L., Diaz-Guilera, A., Duch, J., Arenas, A.: Comparing Community Structure Identification. J. Stat. Mech., P09008 (2005)
7. Fred, A.L.N., Jain, A.K.: Robust Data Clustering. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 128–133. IEEE Press, New York (2003)
8. Girvan, M., Newman, M.E.J.: Community Structure in Social and Biological Networks. Proc. Natl. Acad. Sci. USA 99, 7821–7826 (2002)
9. Gregory, S.: An Algorithm to Find Overlapping Community Structure in Networks. In: Kok, J.N., Koronacki, J., López de Mántaras, R., Matwin, S., Mladenič, D., Skowron, A. (eds.) PKDD 2007. LNCS (LNAI), vol. 4702, pp. 91–102. Springer, Heidelberg (2007)
10. Gregory, S.: Local Betweenness for Finding Communities in Networks. Technical report, University of Bristol (2008)
11. Lancichinetti, A., Fortunato, S., Kertesz, J.: Detecting the Overlapping and Hierarchical Community Structure of Complex Networks. Eprint arXiv:0802.1218v1 at arxiv.org (2008)
12. Nelson, D.L., McEvoy, C.L., Schreiber, T.A.: The University of South Florida Word Association, Rhyme and Word Fragment Norms (1998), `http://w3.usf.edu/FreeAssociation/`
13. Newman, M.E.J.: The Structure of Scientific Collaboration Networks. Proc. Natl. Acad. Sci. USA 98, 404–409 (2001)
14. Newman, M.E.J.: Detecting Community Structure in Networks. Eur. Phys. J. B 38, 321–330 (2004)
15. Newman, M.E.J.: Fast Algorithm for Detecting Community Structure in Networks. Phys. Rev. E 69, 066133 (2004)
16. Newman, M.E.J.: Finding Community Structure in Networks Using the Eigenvectors of Matrices. Phys. Rev. E 74, 036104 (2006)
17. Newman, M.E.J.: Modularity and Community Structure in Networks. Proc. Natl. Acad. Sci. USA 103, 8577–8582 (2006)
18. Newman, M.E.J., Girvan, M.: Finding and Evaluating Community Structure in Networks. Phys. Rev. E 69, 026113 (2004)
19. Nicosia, V., Mangioni, G., Carchiolo, V., Malgeri, M.: Extending Modularity Definition for Directed Graphs with Overlapping Communities. Eprint arXiv:0801.1647v3 at arxiv.org (2008)
20. Palla, G., Derényi, I., Farkas, I., Vicsek, T.: Uncovering the Overlapping Community Structure of Complex Networks in Nature and Society. Nature 435, 814–818 (2005)
21. Pons, P., Latapy, M.: Computing Communities in Large Networks Using Random Walks. J. Graph Algorithms and Applications 10(2), 191–218 (2006)
22. Rand, W.M.: Objective Criteria for the Evaluation of Clustering Methods. J. Am. Stat. Assoc. 66, 846–850 (1971)
23. Salvetti, F., Srinivasan, S.: Local Flow Betweenness Centrality for Clustering Community Graphs. In: Deng, X., Ye, Y. (eds.) WINE 2005. LNCS, vol. 3828, pp. 531–544. Springer, Heidelberg (2005)
24. Wakita, K., Tsurumi, T.: Finding Community Structure in Mega-scale Social Networks. In: 16th International World Wide Web Conference, WWW 2007, pp. 1275–1276 (2007)
25. Wirz, M.: Personal communication
26. Xie, N.: Social Network Analysis of Blogs. M.Sc Dissertation. University of Bristol (2006)
27. Xu, X., Yuruk, N., Feng, Z., Schweiger, T.A.: SCAN: a Structural Clustering Algorithm for Networks. In: 13th International Conference on Knowledge Discovery and Data Mining, KDD 2007, pp. 824–833. ACM, New York (2007)
28. Zhang, S., Wang, R., Zhang, X.: Identification of Overlapping Community Structure in Complex Networks Using Fuzzy C-means Clustering. Physica A: Statistical Mechanics and its Applications 374(1), 483–490 (2007)