Partitioning Large Networks Without Breaking Communities*

Anand Narasimhamurthy, Derek Greene, Neil Hurley, and Pádraig Cunningham

School of Computer Science and Informatics, University College Dublin
{anand.narasimhamurthy,derek.greene,neil.hurley,padraig.cunningham}@ucd.ie

Abstract. The identification of cohesive communities is a key process in social network analysis. However, the algorithms that are effective for finding communities do not scale well to very large problems, as their time complexity is worse than linear in the number of edges in the graph. This is an important issue for those interested in applying social network analysis techniques to very large networks, such as networks of mobile phone subscribers. In this respect the contributions of this paper are two-fold. First we demonstrate these scaling issues using a prominent community-finding algorithm as a case study. We then show that a two-stage process, whereby the network is first decomposed into manageable subnetworks using a multilevel graph partitioning procedure, is effective in finding communities in networks with more than 10^6 nodes.

1 Introduction

After several years of academic research on social network analysis (SNA), considerable commercial interest in exploiting SNA has recently emerged. The research reported in this work is motivated by the prospect of using SNA in large-scale applications, such as exploring online communities [1], mining web usage data [2], identifying research trends in bibliographic networks [3], and analysing relationships among mobile telephone subscribers. The identification of cliques or communities in network data has received a lot of attention in SNA research, where a number of promising techniques have emerged [4–6]. These techniques can identify subgraphs such that the density of edges within the subgraph is greater than the density of edges connecting the subgraph to the rest of the network. They can also identify overlapping communities, where an individual can belong to more than one group. This is an important facility for network analysis in many real-world domains.

Another highly significant issue in this area is that of scalability. Unfortunately, it is in the nature of the analysis entailed in many community-finding algorithms that they do not scale well to very large graphs. In fact, existing techniques in this area generally cannot handle graphs with more than a few

* The work was supported by Science Foundation Ireland Grant Nos. 05/IN.1/I24 & 08/SRC/I407 and Enterprise Ireland Grant No. PC/2007/010.

tens of thousands of nodes (*e.g.* [6]). In contrast, many real-world networks will be substantially larger. For instance, in a study reported by Abello *et al.* [7], a one-day telephone call graph at AT&T consisted of 53,767,087 vertices and more than 170 million edges.

To deal with this issue of scalability, we propose a pragmatic problem decomposition strategy: use a "top-down" graph partitioning technique to decompose the network into smaller subnetworks, on which it is then feasible to apply a more computationally intensive community-finding algorithm. To perform the initial partitioning, we use the multilevel method proposed by Dhillon *et al.* [8], referred to as Graclus. For the second stage of the process, the CFinder algorithm [5] is employed to discover small, overlapping communities. These two techniques complement each other as Graclus is very unlikely to break a community that would be uncovered by CFinder – the basis for this complementarity is explored in more detail in section 5.1. The remainder of the evaluation in this paper demonstrates the effectiveness of the proposed two-stage strategy, covering two key aspects of the community-finding problem:

1. **Scalability:** The two stage strategy is evaluated on synthetic data to demonstrate that it results in a significant reduction in running time, making it possible to discover communities in graphs much larger than can be tackled by a community finding algorithm operating on its own.
2. **Solution quality:** The evaluation also demonstrates that the two stage process produces good solutions – this is assessed according to two related criteria:
   - The extent to which graph partitioning preserves communities intact when applied to a large graph.
   - The agreement between the communities discovered when the community finding algorithm is run on the entire graph, and those discovered when using the subgraphs obtained via graph partitioning.

We perform evaluations on networks with community structures similar to those occurring in large-scale, real-world networks such as those pertaining to mobile phone subscribers – *i.e.* small, dense, localised communities embedded in a very large sparse graph. We show that we can discover communities in these networks in a computationally efficient manner, without adversely affecting the "quality" of these communities.

The remainder of this paper is organised as follows. The next section provides a summary of existing methods, specifically community-finding algorithms and top-down network partitioning algorithms, which are relevant in SNA. We describe our proposed problem decomposition strategy in Section 3. The datasets used in our experiments are presented in Section 4, and our experimental results are subsequently discussed in Section 5. The paper finishes with some conclusions in Section 6.

2 Related Work

2.1 Community-Finding Algorithms

Before discussing recent work in the area of SNA, it is important to say something about the terms "clique" and "community" that pervade this discussion. A "clique" has a formal meaning in graph theory – it refers to a set of nodes that are fully connected (*i.e.* an edge exists between all pairs of nodes). A common exercise when examining the characteristics of a graph is to enumerate all the cliques occurring within it. Although the problem of finding a maximum clique in a graph is NP-hard [9], a number of clique-finding techniques employing a variety of heuristics have been proposed in the graph theory literature [10–14]. Most would agree that this definition of a clique is too strict for SNA, as "communities" of dense connections will be of interest, even if not all pairs of nodes are connected. If the objective is not to discover maximum cliques but instead to discover *dense* structure in the network, then the question will arise of which criterion to optimise when searching for dense structures. It is interesting to see how this issue has been addressed by various authors in the SNA literature. Additionally, it is important to note that techniques for optimising an appropriate *connectedness* criterion often have poor time complexity, thus limiting their usefulness when working with very large graphs. We now outline three prominent algorithms that have been employed for community-finding in SNA tasks.

Newman & Girvan's method (GN). This is essentially a top-down hierarchical clustering strategy, which employs a novel partitioning criterion. This criterion is based on the *shortest path betweenness* measure [15], which is well-established as a measure of node centrality in SNA. In the GN algorithm, the "edge betweenness" refers to the number of shortest paths in a network that pass along an edge. Edges that score highly on this criterion are likely to be inter-cluster edges (*i.e.* edges that link adjacent clusters). It may seem surprising that a criterion for identifying cluster centrality can also identify links between adjacent clusters when applied to edges rather than nodes. In fact, these edges are *weak ties* in the sense of this term introduced by Granovetter [16]. That is, while they are important links connecting individuals across the network, they represent associations between casual acquaintances rather than close friends. The GN approach works by generating a successive partitioning of a graph, cutting the edge with the highest edge betweenness score at each stage. Clearly this will be computationally expensive, given the number of shortest-path calculations that must be performed. Newman & Girvan say the time complexity of this algorithm is $O(n^3)$ on sparse graphs, making it applicable to graphs of up to 10,000 nodes at the time of publication in 2003.

CONGA. A related community-finding approach from the recent literature that can deal with overlapping communities is the CONGA algorithm, proposed by Gregory [6]. Since it is based on the GN algorithm described above [17], this

approach also employs a divisive hierarchical partitioning strategy. In order to allow nodes to belong to more than one cluster, CONGA permits nodes to be split, i.e. a real node v will be split into $\{v_1, v_2\}$ with the incoming edges to v divided between v_1 and v_2, and a new *virtual* edge created to link v_1 and v_2. If this new virtual edge has a higher edge betweenness than any real edge, then the node should be split on this basis. The enhancement that CONGA brings to the GN algorithm is to add this node-splitting step as an extension to the partitioning phase of the algorithm. Gregory states that the worst case time complexity of this algorithm is $O(e^3)$, where e is the number of edges. In practice CONGA has been shown to handle networks of up to 4,000 nodes and 7,000 edges.

CFinder. In contrast to the two previous algorithms which operate in a top-down manner, CFinder [5] is a bottom-up technique that uses cliques as building blocks for large groups. The algorithm first extracts all maximal complete subgraphs (*i.e.* cliques) that do not form part of larger complete subgraphs, and composes these into larger structures through a process called clique percolation. Although an efficient approach for finding all cliques in a general network may not be feasible since determining a maximum clique is NP-hard, Palla *et al.* nevertheless claim that their approach performs well on real networks. From these cliques, all k-cliques are enumerated (*i.e.* complete subgraph of size k). The algorithm then proceeds to find *k-clique-communities*, which are defined as the union of all k-cliques that can be reached from each other through a series of adjacent k-cliques. This approach can discover overlapping and nested communities in binary networks (*i.e.* an unweighted, undirected graph). The authors also note that it can be applied to graphs with weighted edges, as any weighted network can readily be transformed to a binary network by using a weight threshold. Depending on the desired resolution at which the network is to be inspected, the "correct" threshold changes. Palla *et al.* point out that, given the nature of the algorithm, it is difficult to formally analyse the time complexity of CFinder. However some discussion of the algorithm's complexity is provided in [18], where it was observed empirically that the algorithm's running time will often depend very strongly on the structure of the input data.

2.2 Graph Partitioning

It is not always easy to distinguish between the community-finding algorithms reviewed in the previous section, and the graph partitioning algorithms discussed here. However, we can make several broad distinctions. Notably, community-finding algorithms are designed to find regions of dense structure that are not well-connected with the rest of the network. These regions will often exhibit a certain degree of overlap. In contrast, the graph partitioning problem involves finding the optimal division of a graph into k disjoint parts according to a chosen criterion, such that the partition covers the entire graph. Common applications for graph partitioning algorithms include VLSI module placement, load balancing for parallel processing, and image segmentation [19].

Of the vast array of approaches that have been proposed in the literature, two of the most important classes are spectral clustering and multilevel partitioning. Spectral methods produce a partition based on the eigendecomposition of the graph. Usually the Laplacian matrix is used in this context, rather than the original adjacency matrix. The reader is referred to [20] for a detailed discussion on the Laplacian matrix, and the connection between the eigenvalues of the Laplacian with many key invariants of the graph. Spectral approximations for a variety of partitioning criteria have been formulated, including the minimum cut [21], ratio cut [22], and normalised cut [19].

Many multilevel approaches for graph partitioning have been developed over the years, the Metis algorithm [23] being the most well-known example. In these approaches a sequence of successively smaller, coarser hypergraphs is constructed. A bisection of the smallest hypergraph is computed and is successively projected to the next finest level. At each level, an iterative refinement algorithm is used to improve the bisection. Most multilevel algorithms are based on the seminal work by Kernighan & Lin [24]. Given an edge weighted graph $G = (V, E)$, and an initial partitioning of the nodes $V = A \cup B$, the algorithm proceeds by finding equal-sized subsets of nodes $X \in A$ and $Y \in B$, such that exchanging X and Y reduces the total cost of edges between the old partitions A and B. Previously it was assumed that the initial partitions would be of equal sizes. However, a number of variants of the original algorithm have been proposed which support unbalanced clusters.

Recently, Dhillon *et al.* [8] developed a fast multi-level algorithm that directly optimises various weighted graph clustering objectives. The authors show that a general weighted k-means objective is mathematically equivalent to a weighted graph clustering objective, and they exploit this equivalence. The main advantage of their method is that it approximates graph clustering objectives without requiring an eigendecomposition, which can be computationally intensive for large graphs. Another advantage of this algorithm compared to other multilevel approaches is that it does not require the partitions to be of equal sizes.

3 Problem Decomposition Strategy

While community-finding algorithms have been aimed at SNA applications, these techniques are computationally expensive and the communities they discover are small – normally comprising some tens of nodes. When dealing with very large graphs, which may potentially contain hundreds of thousands or even millions of nodes, algorithms with running time $O(e^2)$ or $O(e^3)$ will not be practical. Therefore the question arises, how can we find small communities in such large networks?

The approach we take in this work is to employ a two-stage problem decomposition strategy as outlined below:

Stage 1: Given a large graph, apply a computationally tractable graph partitioning algorithm that minimises the number of links broken in the process. The resulting partitions will themselves form large subgraphs.

Stage 2: Once the original graph has been split into manageable subgraphs, apply a more computationally intensive community-finding algorithm to each subgraph, and subsequently combine the results.

For the first stage of the process we use Graclus[1] by Dhillon *et al.* [8]. We choose this approach as it allows us to optimise the objectives for spectral clustering, which have previously proved successful in other areas [19], but without requiring a costly eigendecomposition. Thus Graclus will provide a partitioning of a large graph with a small running time. For the second stage of the process, we employ the CFinder algorithm [5], as it supports the discovery of overlapping, localised and nested communities. A detailed discussion regarding the motivation for combining these two algorithms, based on the compatibility of their objectives, is provided later in Section 5.1.

4 Experimental Datasets

To evaluate the effectiveness of the proposed decomposition strategy, we performed a variety of experiments on both synthetic and real-world datasets. Synthetic data was used for two main purposes. Firstly, we wished to examine how community-finding techniques scale to large, sparse graphs, approaching the size of those occurring in real-world applications, such as the analysis of mobile subscriber networks. Secondly, we wished to assess the extent to which graph partitioning techniques preserve communities intact. In addition to using synthetic data, we also ran experiments on real network data, specifically unweighted and weighted graphs derived from the CORA bibliographic dataset [25].

4.1 Synthetic data

We performed experiments on two types of synthetically generated network graphs[2]. The generation and structure of these graphs is now described in detail.

Power law graphs. An important characteristic of networks is the distribution of edges. The most straightforward assumption about the distribution of edges is that all possible edges between the vertices are equally probable [26]. This defines a random graph where the degree of each node follows a Poisson distribution. It has been shown that for many real graphs the degree distribution follows a power law rather than a Poisson distribution [27]. An example of the distinction between the distributions is given in Figure 1. The important point about power law graphs is the increased probability of nodes with large numbers of edges. These nodes represent *hubs* which have a significant impact on the overall connectivity of the graph, since a connected set of hubs can provide short paths between a large proportion of the nodes in the network. Thus the average

[1] Available from `http://www.cs.utexas.edu/users/dml/Software/graclus.html`
[2] Available for download at `http://mlg.ucd.ie/datasets/graph.html`

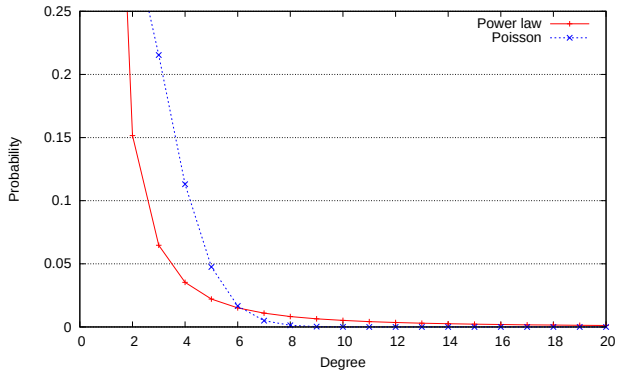


Fig. 1. A comparison of degree distribution for random and power law graphs - a common model for degree distribution in random graphs is a Poisson distribution.

distance between vertices in a power law graph will be small. Power-law graphs also tend to have a high cluster coefficient and exhibit scale-free characteristics, so the plot of degree distribution is self-similar at different levels of resolution.

The degree sequence of a graph is the set of node degrees $(d_1, \ldots d_n)$, ordered such that $d_1 \geq \cdots \geq d_n$. To construct synthetic graphs that approximate real-world power-law graphs, we generated random graphs with a given fixed input degree sequence, and chose the degree sequence to follow an exact power-law. That is, for $k = 1 \ldots n$, we choose d_k such that

$$d_k = \text{round}(ck^{\alpha}), \tag{1}$$

for chosen constants c and α. In particular, we select a reasonable value of α based on values reported from real-world graphs. The constant of proportionality c, was set by specifying the minimum degree $d_n = d_{\min}$, so that

$$d_k = \text{round}\left(\left(\frac{k}{n}\right)^{\alpha} d_{\min}\right) \tag{2}$$

Note that, provided the degree sequence is graphical, it is possible to generate a graph corresponding to that degree sequence by applying the constructive method proposed in [28]. Real-world graphs do not exhibit the perfect power-law characteristics of these synthetic graphs, but rather tend to exhibit a power-law behaviour in certain ranges of the node degrees. Nevertheless, this is a useful approximation that allows us to examine algorithms on large-scale graphs which exhibit some of the characteristics that can be found in real-world graphs. In the experiments described in Section 5, we use graphs with $d_{\min} = 8$ and $\alpha = 10$.

Graphs with embedded communities. While cliques and variants such as k-plexes are precisely defined, as noted previously, the term "community" is not clearly defined in the literature. In most cases, the term refers to a group of

nodes which have a high degree of connectivity between themselves, relative to the rest of the network. In order to capture this notion of a community, we use the measure *assoc*, which quantifies the connectivity between a community C and a graph G, where $C \subseteq G$:

$$assoc(C, G) = \sum_{u \in C, v \in G, u \neq v} w(u, v)\,, \tag{3}$$

where $w(u, v)$ is the weight of the edge connecting a pair of nodes u and v. Note that in the data described here, edge weights are either 1 or 0. In this case $assoc(C, C)$ is a count of the number of edges connecting nodes within C.

Following [4], we generate artificial graphs with known community structure, using a modification of the Erdős-Renyi (ER) random graph model. In an ER graph with parameter p, each pair of vertices is connected with probability p. We generate communities $C \subseteq G$, which are ER subgraphs of G with a fixed parameter $p = p_{in}$. This implies that for a community of size c (i.e. c vertices), the expected value of $assoc(C, C)$ is given by

$$\mathrm{E}\{assoc(C, C)\} = \frac{c \times (c-1)}{2} p\,. \tag{4}$$

To generate between-community edges, we introduce a parameter p_{out}, representing the probability that an edge connecting two vertices from different communities is present in the graph. Another measure of community quality that takes into account between-community connectivity is the ratio $r(C)$ of the within-community connectivity in C to the connectivity of the community's nodes to the entire graph:

$$r(C) = \frac{assoc(C, C)}{assoc(C, G)}\,. \tag{5}$$

If we use

$$\hat{r} = \frac{\mathrm{E}\{assoc(C, C)\}}{\mathrm{E}\{assoc(C, G)\}}\,, \tag{6}$$

to estimate $r(C)$, it follows that for a community of size c, generated using p_{in} and p_{out},

$$\hat{r}(c) = \frac{(c-1)p_{in}}{(c-1)p_{in} + 2p_{out}(n-c)}\,, \tag{7}$$

where n is the total number of vertices.

For evaluation purposes, we constructed "unit" graphs consisting of 1,000 nodes, containing embedded communities of different sizes, ranging from 10 to 40 nodes. These communities were non-overlapping, and all nodes were assigned to a community. Successively larger graphs were then generated such that the number of nodes and communities were scaled by the same integral ratio. For example, a graph of 2,000 nodes had exactly twice the number of communities of the same respective sizes as a graph of 1,000 nodes.

In order to better understand the nature of the synthetic data, some example communities consisting of ten nodes, but of different link densities, are shown

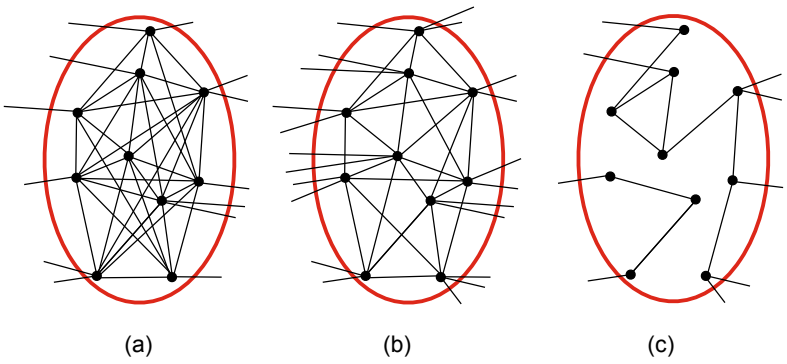


Fig. 2. This figure shows three example communities of 10 nodes in the synthetic data with different parameter values: in (a) $p_{in} = 0.7, r = 0.7$, in (b) $p_{in} = 0.5, r = 0.5$, in (c) $p_{in} = 0.2, r = 0.5$. For instance, in example (c) 20% of the possible links in the community exist, and half of all links are internal to the community.

in Figure 2. In Figure 2(a) a community of ten nodes with $p_{in} = 0.7$ and $r = 0.7$ is shown. At this density, on average 70% of all possible within-community edges exist, and 70% of all edges associated with the community are internal. In the evaluation in Section 5.3 it is clear that the community finding algorithms perform well when the community "signature" is this pronounced. In the scenario in Figure 2(b) the community signature is weaker with just 50% of possible internal links present and as many external as internal links in existence. The evaluation in Section 5.3 still shows good performance on this data. The situation in Figure 2(c) is effectively a pathological situation, as only 20% of all possible internal links are present. In this situation the community does not even form a single connected component. Perhaps unsurprisingly, most community finding algorithms will perform poorly in this context.

4.2 Real data

The CORA bibliographic dataset [25] contains information and annotations such as authors, cited papers and topic for over 50,000 research papers[3]. For our evaluation, we largely focus on the undirected binary paper-paper citation graph derived from CORA. Only the papers with available authors were selected, where the total number of these was 28,400. Of these, the largest weakly-connected component was used for the experiments described in the next section. This resulted in a graph consisting of 24,542 nodes. We suggest that this graph is a reasonable proxy for real-world data, such as mobile subscriber networks, since the communities are small despite the scale of the graph – typically containing 10 to 20 members. In Section 5.6 we also consider the weighted co-authorship graph derived from the CORA collection.

[3] See `http://www.cs.umass.edu/~mccallum/code-data.html`

5 Evaluation

The work presented in this section has two objectives: to highlight the scalability advantages of the two stage process, and to examine how the prior application of graph partitioning affects our ability to locate communities in a large graph. Additionally we compare the communities obtained from the entire graph with those discovered from the individual subgraphs obtained via graph partitioning. Before considering large graphs, in Section 5.1 we firstly examine the relation between the objectives of the two phases of the proposed decomposition strategy. Subsequently evaluations on synthetic data are presented in Section 5.2 and Section 5.3, on the power-law and embedded community graphs respectively (generated as described in Section 4.1). The results of evaluations on real data are then presented in Section 5.4 and Section 5.5.

5.1 Compatibility of CFinder and Graclus objectives

As will be demonstrated later, the two-stage problem decomposition strategy reduces run-time significantly in comparison to a single application of the CFinder algorithm on the entire graph. However, it is important to also examine whether the communities found by the two-stage process deviate significantly from those found by a single application of CFinder. Posing this question in another way, we must evaluate the likelihood that the partitioning stage breaks communities that would otherwise have been identified by CFinder. The CFinder algorithm attempts to enumerate all *percolation k-cliques* that exist in the graph. These are subgraphs consisting of a chain of *connected* cliques of k vertices, where two cliques are said to be connected if they have $k-1$ vertices in common. On the other hand, the Graclus software attempts to find a partitioning of the graph vertices set into m partitions $\{P_1, \ldots, P_m\}$ that minimises the *normalised cut*. This objective can be defined in terms of the measure *assoc* (Eqn. 3) as follows:

$$\sum_{i=1}^{m} \frac{assoc(P_i, G \setminus P_i)}{assoc(P_i, G)} \,. \tag{8}$$

If a minimum cut partitioning is likely to split a percolation k-clique, then the two objectives are not compatible and the two-stage strategy is likely to produce a significantly different set of communities than the single-stage strategy. As discussed in [5], in an ER graph with parameter p, for a given k, there exists a threshold probability at which a giant percolation k-clique component appears in the graph. The critical probability is given by

$$p^c(N,k) = \frac{1}{[(k-1)N]^{\frac{1}{k-1}}} \tag{9}$$

where N is the number of nodes in the graph.

Consider a graph of size N consisting of two communities C_1 and C_2, both of size $N/2$, generated as described in Section 4.1 using the parameters p_{in}

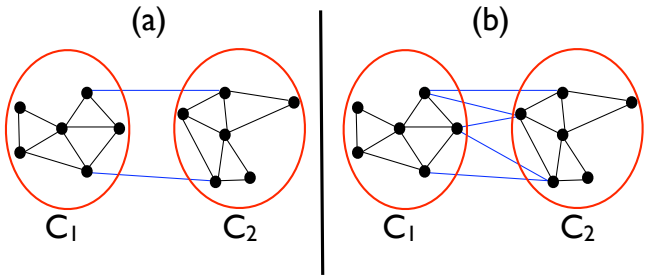


Fig. 3. C_1 and C_2 are two connected percolation k-cliques with $k = 3$. (a) When p_{out} is small, the two communities remain distinct as percolation k-cliques. (b) When p_{out} is sufficiently large, the two communities merge into a single percolation k-clique.

and p_{out}. The within-community connection probability p_{in} is chosen so that $p_{in} \gg p^c(N/2, k)$. Thus, with high probability, each of C_1 and C_2 is covered by a single percolation k-clique community, see Figure 3. Clearly, when $p_{out} = 0$, the two communities are unconnected; the CFinder algorithm will identify the two percolation k-cliques, C_1 and C_2 and the Graclus bi-partitioning algorithm will uncover the minimum normalised cut partitioning $\{C_1, C_2\}$. As p_{out} is increased, there is some critical probability p^c_{out} above which the two percolation k-cliques merge into a single percolation k-clique with high probability and thus CFinder returns just a single community, $C_1 \cup C_2$. Also, as p_{out} increases, it becomes more and more likely that the bi-partitioning that minimises the normalised cut deviates from $\{C_1, C_2\}$. If the normalised cut criterion can correctly identify the underlying community structure for all values of p_{out} up to p^c_{out}, that is for values of p_{out} at which CFinder identifies two distinct percolation k-cliques, then this suggests that the normalised cut criterion is compatible with the CFinder objective and that the first stage is unlikely to break percolation k-cliques.

A minimum normalised cut partitioning is unlikely to split very dense subgraphs [29]. As the density of percolation k-cliques increases with k, compatibility problems between the two objectives are most likely to manifest themselves for small values of k. Therefore, we restrict our analysis to $k = 3$. By construction, C_1 is fully contained inside a percolation k-clique, $K_1 \supseteq C_1$. To measure the extent of merging between the two communities, we compute $t = \frac{|C_1|}{|K_1|}$. When $p_{out} = 0$, $K_1 = C_1$ and $t = 1$. When p_{out} is sufficiently large that the two communities are completely merged, $K_1 = C_1 \cup C_2$ and hence $t = \frac{|C_1|}{|C_1 \cup C_2|} = \frac{1}{2}$. Values between these extremes ($0 < t < \frac{1}{2}$), may be interpreted as the extent to which the two communities C_1 and C_2 have merged. To measure the ability of Graclus to correctly identify the communities, we measure the overlap between community C_1 and a Graclus partition, using $s = \frac{|P_1 \cap C_1|}{|C_1|}$, where P_1 is the normalised cut partition with which C_1 has maximum overlap. Again, as p_{out} increases from 0, we expect that s decreases from 1.

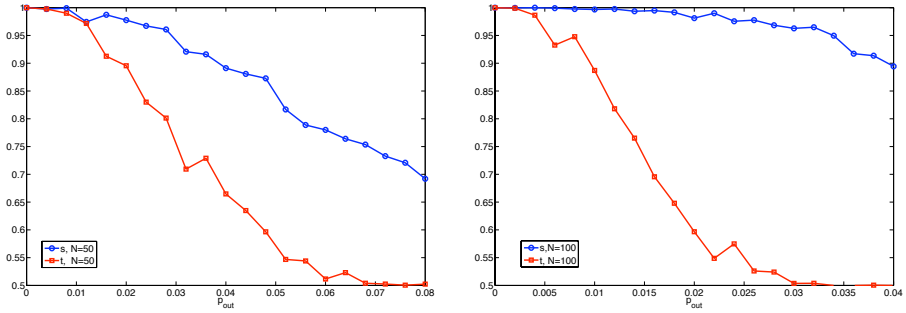


Fig. 4. Evaluation parameters $t = \frac{|C_1|}{|K_1|}$ and $s = \frac{|P_1 \cap C_1|}{|C_1|}$ plotted against p_{out} for two different values of N. The t measure is an agreement score for CFinder while s is an agreement score for Graclus. Each point of the plot is an average over 50 randomly generated graphs.

Thus the value t represents the fraction of the embedded community C_1 contained in the cluster produced by CFinder, while s represents overlap between embedded community C_1 and the normalised cut cluster P1 produced by Graclus. These can be viewed as two related measures of agreement between two clusters.

Figure 4 shows that, as p_{out} increases in value from 0 (*i.e.* the embedded communities C_1 and C_2 start to connect), the agreement between C_1 and the CFinder cluster decreases at a faster rate than the agreement between C_1 and the Graclus cluster. This trend becomes more pronounced as the number of nodes N increases – CFinder will fail to identify increasingly inter-connected communities before Graclus does. To put this another way, for large enough N (as is the case in real-world networks), a Graclus partitioning will correctly separate two percolation k-clique communities whenever CFinder recognises them as distinct communities. From this it may be concluded that the objectives of CFinder and Graclus are largely compatible and suitable for use together in the context of the proposed two-stage decomposition strategy.

To observe the behaviour as a function of N, in Figure 5, we plot s against N for the value of p_{out} at which $t \approx 0.8$, that is, when C_1 accounts for around 80% of the largest percolation k-clique that contains it. When $N = 20$ and C_1 is around 80% of the largest k-clique containing it, the Graclus partitioning also has nearly 80% overlap with C_1.

In summary, it may be concluded that the objectives of CFinder and Graclus are largely compatible. For large N, a Graclus partitioning correctly separates two percolation k-clique communities whenever CFinder recognises them as distinct communities. The likelihood that a normalised cut partitioning splits a percolation k-clique community increases as N decreases, although performance is still reasonably good, down to small communities of size $N/2 = 10$.

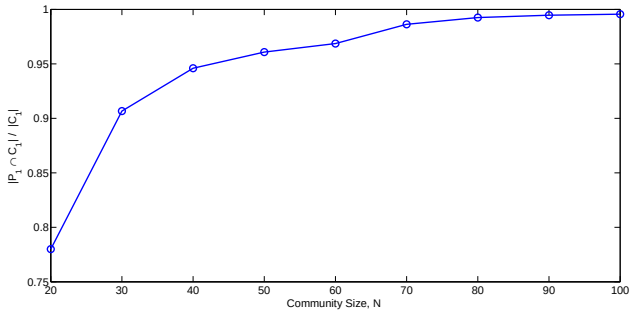
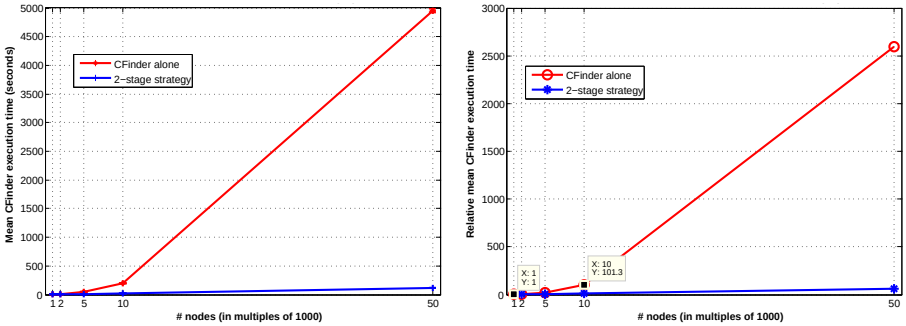


Fig. 5. Evaluation parameter s plotted against N, with p_{out} chosen s.t. $t \approx 0.8$. Each point of the plot is an average over 50 randomly generated graphs.

5.2 Scalability: Synthetic data

Our scalability analysis on synthetic data entailed comparing the running times of the proposed two-stage strategy with those of CFinder alone, when applied to the two types of graph generated as described in Section 4.1. In this analysis the graph partitioning stage was set to yield sub-graphs of about 1,000 nodes (*i.e.* a graph of 10,000 nodes is divided into 10 sub-graphs). The results for the power law graphs are shown in Figure 6, and those for graphs with embedded communities are shown in Figure 7. The results in the latter correspond to parameter sets $(p_{in}, r) = (0.2, 0.5)$, $(p_{in}, r) = (0.5, 0.5)$ and $(p_{in}, r) = (0.7, 0.7)$. These parameter values correspond to community signatures of the type shown in Figure 2. The results show average execution times computed over 50 runs.

In these figures, both the actual and relative execution times are shown. The graphs showing the actual execution times are included to illustrate the dependence of the CFinder execution time on the link density. It is clear that execution



(a) (b)

Fig. 6. Mean CFinder execution times (in seconds) versus graph size for synthetically generated power law graphs: (a) actual execution time, (b) execution time relative to 1000 node graphs.

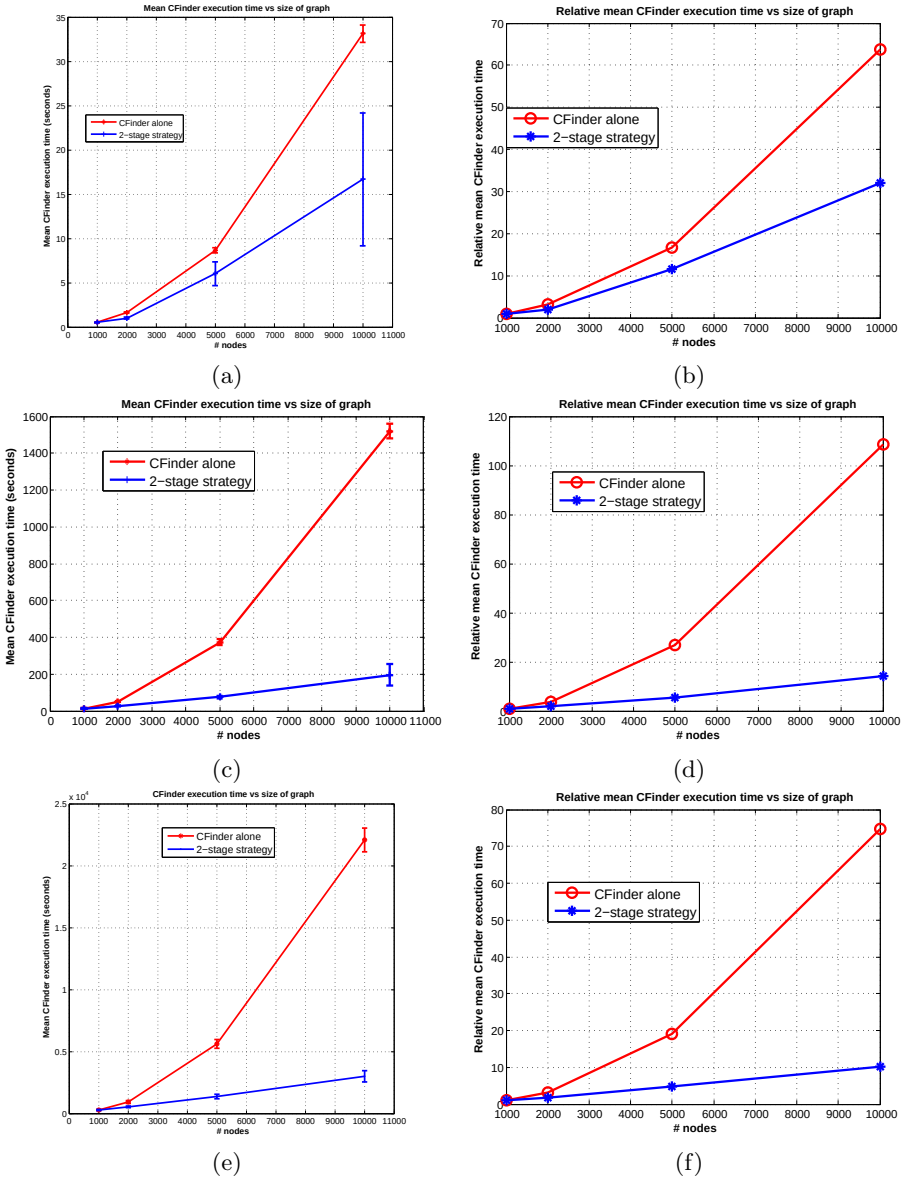


Fig. 7. Mean CFinder execution times versus graph size for graphs with embedded communities. Actual execution times are shown in (a,c,e) while relative execution times relative to a 1,000 node graph are shown in (b,d,f). The parameters for the graphs are $(p_{in}, r) = (0.2, 0.5)$ in (a,b), $(p_{in}, r) = (0.5, 0.5)$ in (c,d), $(p_{in}, r) = (0.7, 0.7)$ in (e,f).

time increases steeply as more links are added to the graph. For both the power law and embedded community graphs, we observe a steep increase in the execu-

Graph type	# Nodes	# Partitions	CFinder execution time	
			CFinder only	2-stage
Embedded communities $(p_{in}, Nconn_{avg}) = (0.5, 0.5)$	50k	50	40604.6	1227.94
Embedded communities $(p_{in}, Nconn_{avg}) = (0.5, 0.5)$	100k	10	189022.89	29967.89
Power law $(d_{min}, \alpha) = (8, 10)$	1,000k	1000	N/A	365.73

Table 1. Comparison of sample CFinder execution times (in seconds) for large synthetic graphs. Note that N/A indicates that CFinder did not terminate within a "reasonable" period of time.

tion times as the graphs grow in size even though the average "community size" is the same. For example, we can see from Figure 6(b) that the average execution time for a 10,000 node graph is more than 100 times that for a graph of 1000 nodes. The execution time is approximately $O(n^2)$ in the number of nodes. The running times for the two-stage strategy are also shown in the graphs. It is clear that this strategy results in a dramatic reduction in overall running time. The combined running times are computed as follows. Let the graph be partitioned into k parts. The total running time on a graph may be expressed as

$$T = G_k + \sum_{i=1}^{k} C_i \tag{10}$$

where G_k denotes the execution time for partitioning the graph into k parts using Graclus, and C_i denotes the execution time of CFinder on the i^{th} subgraph resulting from the graph partitioning. For our datasets, the running time for Graclus was usually a fraction of a second for graphs consisting of up to a few tens of thousands of nodes. Thus the total running time was dominated by the CFinder execution times on the individual subgraphs.

In Figures 6 and 7 we have shown the combined running times for graphs containing up to a few tens of thousands of nodes. However, the problem decomposition strategy permits discovery of communities in graphs of sizes significantly larger than possible using the community finding algorithm on its own. For instance Graclus can easily handle sparse graphs up to a few million nodes. Sample execution times for the two stage strategy on larger graphs are shown in Table 1. In these examples it is clear that the speedup rate is greater for 50 partitions than it is for 10 partitions. It is difficult to characterize this precisely as it is difficult to characterize the computational complexity of CFinder precisely [18]. However if CFinder is $O(n^2)$ (it is on most of the datasets described here) then the speedup will be proportion to p^2 where p is the number of partitions.

In cases where the subgraphs were larger than could be handled by CFinder, we further partitioned the subgraphs. The actual running time of CFinder on the entire graph is shown where available, although in many cases CFinder either

took an unreasonably long time or did not terminate at all. For our datasets we found that graphs of 10,000 nodes and $> 80,000$ edges were close to the limit of what can be handled by CFinder on a 2GHz dual core machine with 4GB RAM, running 64-bit Linux.

5.3 Solution quality: Synthetic data

Having established that the two-stage strategy could scale for graphs of up to 1,000,000 nodes, the next objective of our evaluation was to examine the quality of the results produced by the two-stage process. The first issue was to assess the extent to which the graph partitioning stage preserves communities when applied to a large graph. The second issue was to compare the sets of communities discovered by CFinder alone to those discovered by the two-stage process.

A few sample communities discovered by the CFinder software are shown in Figure 8. As mentioned before, CFinder enumerates all small cliques (specifically all k-cliques), and uses these as "building blocks" to enumerate communities. A k-clique-community is defined as the union of all k-cliques that can be reached from each other through a series of adjacent k-cliques, making this strategy useful in discovering overlapping as well as nested communities. Thus the communities discovered by CFinder can vary in size from a few nodes (Figure 8) to tens or even hundreds of nodes (Figure 9). Hence it is necessary to compare the

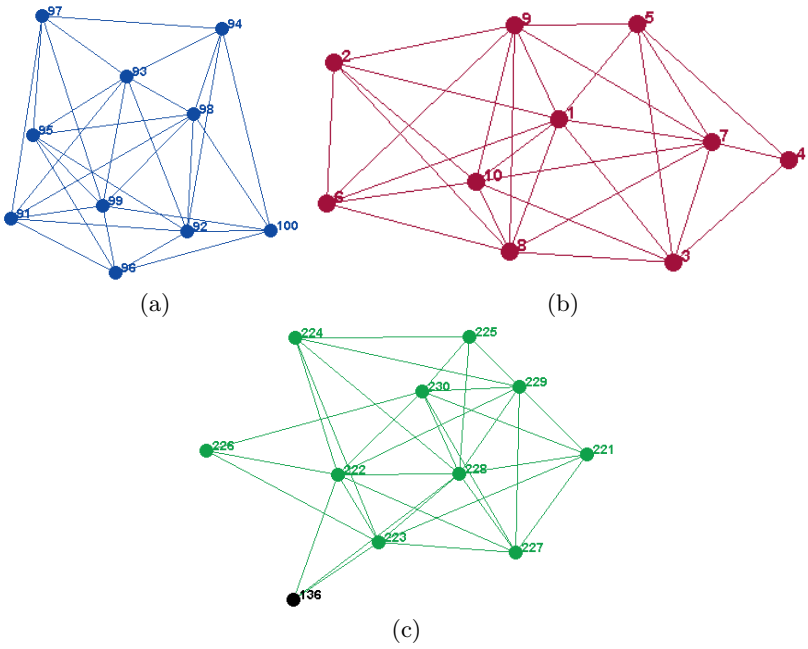
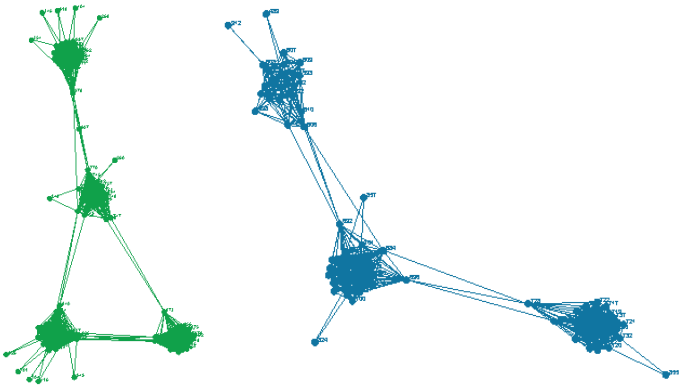


Fig. 8. Illustrative examples of smaller communities discovered by CFinder on synthetic data.



(a) (b)

Fig. 9. Illustrative examples of larger communities discovered by CFinder on synthetic data, which are comprised of smaller, interconnected communities.

communities obtained by CFinder to the true communities, specifically to what extent CFinder is able to faithfully discover the true communities. As an example, consider again Figures 8(a)-(c). In Figures 8(a) and 8(b), CFinder faithfully uncovers the communities (*i.e.* only community nodes with no other extraneous ones). Figure 8(c) is a case where an extraneous node (the node labelled 136, highlighted in a different colour) is chosen along with the other correct community nodes. A different scenario is shown in Figure 9, where the large communities presumably subsume a number of smaller communities. Again it is important here to assess whether or not the true communities are faithfully recovered.

Validation measures. Given the issues raised by Figures 8 and 9, we use *precision* and *recall* as measures of cluster quality in our evaluation process. Precision can be seen as a measure of relevance – *i.e.* the fraction of results that are *true positives* (although other relevant results may be missed out). Recall is a measure of completeness – *i.e.* the fraction of all relevant results that were retrieved (possibly including a number of irrelevant results). Formally these measures are defined as follows:

$$Precision = \frac{TP}{TP + FP} \tag{11}$$

$$Recall = \frac{TP}{TP + FN} \tag{12}$$

where TP is the number of true positives, FP is the number of false positives, and FN is the number of false negatives.

To calculate the actual validation scores for a given clustering, we compared each reference community from the "ground truth" with every one of the clusters discovered by the community finding algorithm as follows.

– Let V denote the set of all nodes in the graph.
– Let $R = \{D_1, D_2, \ldots, D_L\}$ denote the set of reference communities.
– Let $S = \{C_1, C_2, \ldots, C_K\}$ denote the set of communities obtained from the community finding algorithm.

Now let $prec(C_i, D_j)$ and $rec(C_i, D_j)$ denote the precision and recall of C_i with respect to the reference community D_j, where:

– $TP = |C_i \cap D_j|$
– $FP = |C_i \cap (V \setminus D_j)|$
– $TN = |(V \setminus C_i) \cap (V \setminus D_j)|$
– $FN = |(V \setminus C_i) \cap D_j|$

where V denotes the set of vertices and $\setminus$ denotes the set difference operator.

For each community C_i we compute $maxPrec(C_i)$, the maximum precision among all reference communities in R. The overall precision score for the communities in T is given by the average of the maximum precision scores:

$$maxPrec_{avg} = \frac{\sum_{i=1}^{K} maxPrec(C_i)}{K} \tag{13}$$

An overall score $maxRecall_{avg}$ can be calculated in the same way.

Discussion of validation results. The mean validation scores for synthetic graphs with embedded communities, containing 2,000 and 5,000 nodes, are shown in Figure 10 and Figure 11 respectively. We provide results for three sets of synthetic graph generation parameters, ranging from weak signatures ($p_{in} = 0.2, r = 0.5$) to strong signatures ($p_{in} = 0.7, r = 0.7$). The x-axis in these figures refers to the community "index", corresponding to their position in the list of all 120 communities as arranged in ascending order of size. The plots on the left hand side (a,c,e) correspond to precision scores, while those on the right hand side (b,d,f) correspond to recall scores. Note that, for the graph partitioning phase, we are only concerned with recall scores, as precision values are not meaningful in this context since the number of chosen partitions is usually much smaller than the number of communities. As long as existing communities are not broken up by graph partitioning, there remains the potential for them to be discovered by the community finding algorithm in the second stage of the two-stage strategy.

From Figure 10 and 11 we observe that the recall scores for the partitions obtained from Graclus are consistently high, especially for larger communities. This is a good indication that the first stage of the two-stage strategy is relatively successful in preserving communities intact. In fact, these scores are close to 100% for communities with reasonably well-defined signatures, as shown in Figure 10(c,e) and Figure 11(c,e). It is apparent that, when errors are made, they are most likely to be made on the small communities consisting of only ten nodes. The only situation where CFinder alone outperforms the two-stage strategy in terms of solution quality is in Figure 11(d). Here the recall of the two stage-strategy is slightly worse on the small communities of ten nodes. Note

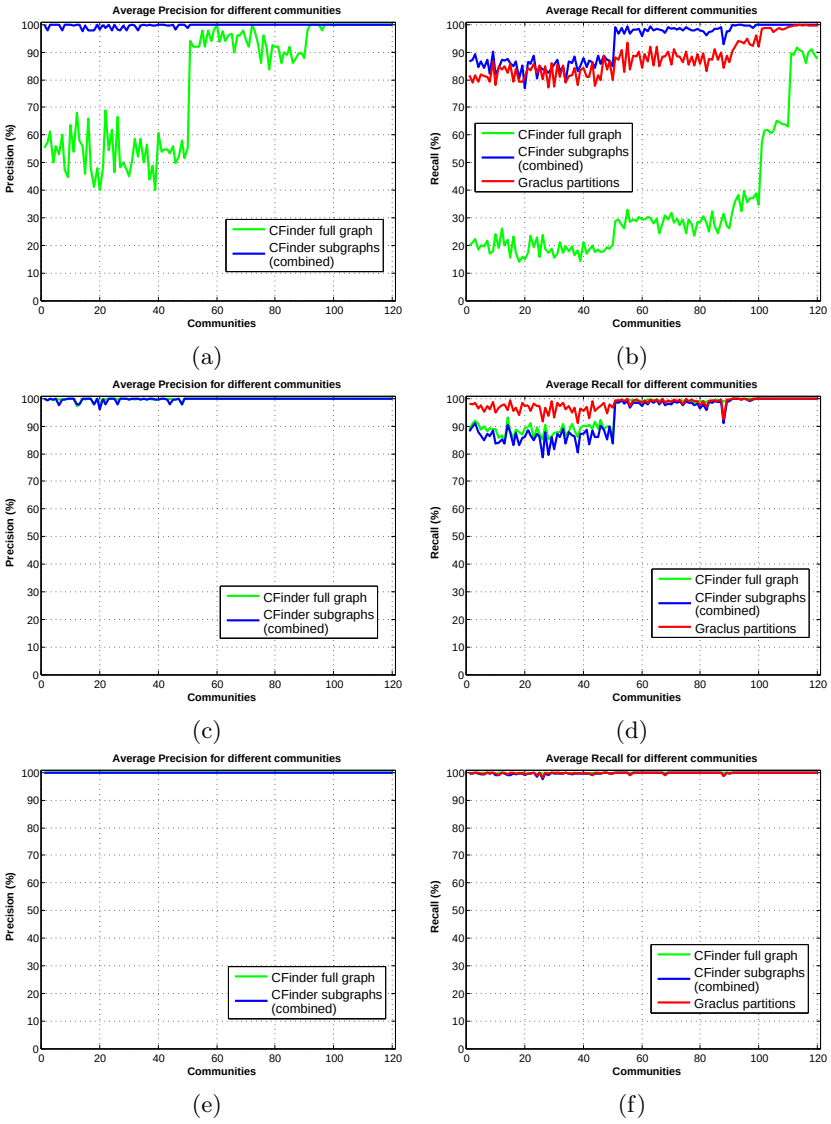


Fig. 10. Mean precision and recall scores for synthetic graphs of 2,000 nodes, generated with different parameter values: in (a,b) $(p_{in}, r) = (0.2, 0.5)$, in (c,d) $(p_{in}, r) = (0.5, 0.5)$, in (e,f) $(p_{in}, r) = (0.7, 0.7)$.

that the graphs in Figure 10(a,b) and Figure 11(a,b), with $(p_{in}, r) = (0.2, 0.5)$, represent a pathological case, where neither the two-stage strategy nor CFinder alone can effectively discover the communities embedded in the data, since the community signatures are not sufficiently well-defined.

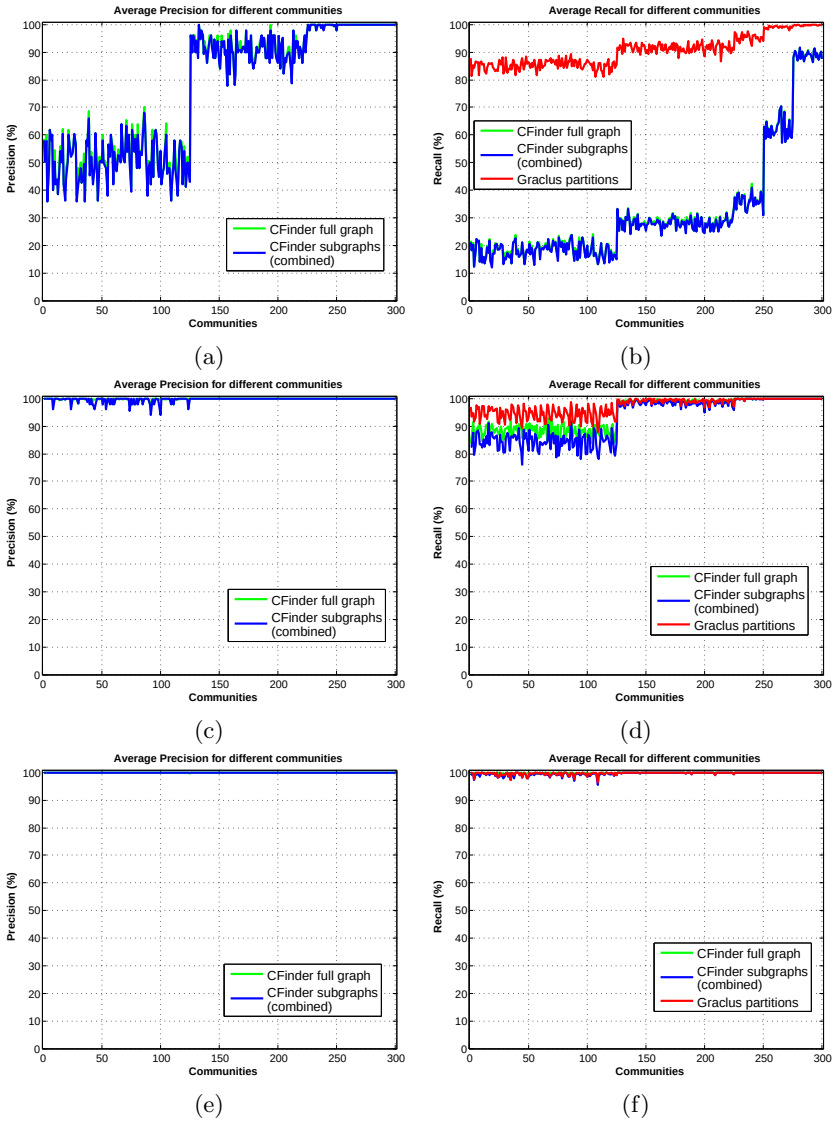


Fig. 11. Average precision and recall scores (Precision : (a),(c),(e). Recall : (b),(d),(f)) for synthetic graphs of 5,000 nodes generated with different parameter values: in (a,b)$(p_{in}, r) = (0.2, 0.5)$, in (c,d)$(p_{in}, r) = (0.5, 0.5)$, in (c,d)$(p_{in}, r) = (0.7, 0.7)$

5.4 Scalability: Real-world data

We now shift our attention to the CORA paper-paper graph. In the experiments performed on this graph, we wished to determine the degree to which communities discovered by running CFinder on the entire graph are uncovered when CFinder is run on the individual subgraphs obtained from graph partitioning.

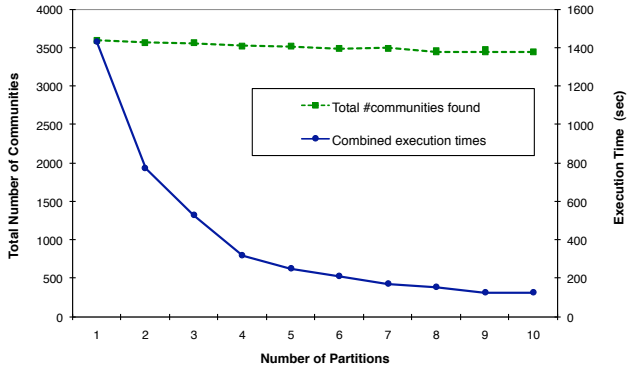


Fig. 12. Execution times for the two-stage community finding process on the CORA dataset. The first data point corresponds to the single stage process (*i.e.* CFinder alone), while the remaining points correspond to the combined times for the two-stage decomposition strategy. The total number of communities discovered is also shown.

For the latter case, we pooled together the communities obtained from the different subgraphs, removing any duplicates. We then compared the total number of communities obtained from the entire graph with those obtained from partitioning the graph into $k = \{2, 3, \ldots, 10\}$ subgraphs. Figure 12 shows that increasing the number of subgraphs had little effect on the total number of communities recovered, with a small decrease from 3,601 on the original graph to 3,448 for $k = 10$. In contrast, we clearly observe that the overall running time of the two-stage community finding process decreases substantially as the number of subgraphs k increases.

Next we compared the actual communities discovered by applying CFinder on the CORA dataset, with those recovered by the two-stage strategy. Our experiments showed a very considerable overlap. For example, the total number of communities obtained by running CFinder on the full graph was 3,601, and the total number after partitioning the graph into two subgraphs was 3,572. Of these, the number of communities in the two sets that were identical was 3,454. Of the remaining 147 communities obtained from the full graph, there were close matches for 118 communities. When the graph was partitioned into ten subgraphs, the total number of communities was 3,448, of which 2,967 had exact matches from among the communities of the full graph.

To further quantify these observations, Figure 13 shows the *recall* (as defined by $maxRecall_{avg}$ in Section 5.3) of the two-stage strategy relative to the 3,601 communities discovered by running CFinder on the full graph. These communities vary significantly in size, from 3 to 14,402 papers, with a mean size of 5 and median size of 12. Even for a partitioning of $k = 10$, CFinder subsequently managed to recover the same communities with a recall of $> 97\%$. In contrast, applying CFinder after randomly partitioning the graph leads to a significantly lower level of recall, which becomes more pronounced as the number of parti-

tions increases. Since it is apparent that Graclus rarely splits CFinder k-clique communities in practice, this suggests that an additional merging procedure to combine communities from different partitions will generally not be required at this point.

5.5 Solution quality: Real-world data

We now consider the issue of solution quality on the CORA paper-paper graph. Since clustering in bibliometrics is generally based on the assumption that a paper will tend to cite other papers in related topics of research, we focused on the ability of the two-stage strategy to recover the underlying topical groupings in the dataset. Specifically, we compared the output of CFinder after Graclus partitioning to the top-level topic annotations in the CORA dataset as a "ground truth". Figure 14 shows the overall distribution of these topics in the CORA dataset. It is clear that the topics are highly unbalanced in size, with the topic "artificial intelligence" alone accounting for 35% of the papers.

On first inspection Figure 15(a) appears to suggest that CFinder after random partitioning leads to communities with a marginally higher level of average *precision* (*i.e.* $maxPrec_{avg}$ as defined in Section 5.3) than achieved by CFinder after applying Graclus. However, taken in the context of Figure 15(b-d), we clearly see that the former approach is producing far fewer communities of smaller size that cover less of the papers in the dataset. These plots show that the problem becomes greatly exacerbated as the number of random splits increases. Conversely the proposed two-stage strategy manages to maintain approximately the same level of cluster quality – the values for *precision*, average cluster size, and the rate of paper assignment do not change significantly as k increases. This is to be expected as Figure 13 previously indicated that partitioning with Graclus did not interfere with the identification of CFinder k-cliques – and thus will not impinge on CFinder's ability to recover the ground truth topics in the

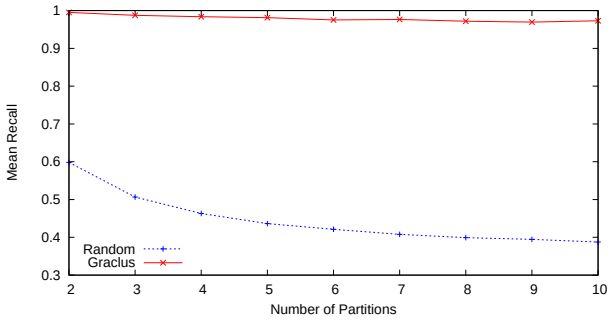


Fig. 13. Comparison (in terms of *recall*) of the ability of CFinder after Graclus partitioning to recover the same communities identified by CFinder on the full CORA graph, with the performance of CFinder after random partitioning (averaged over 40 runs).

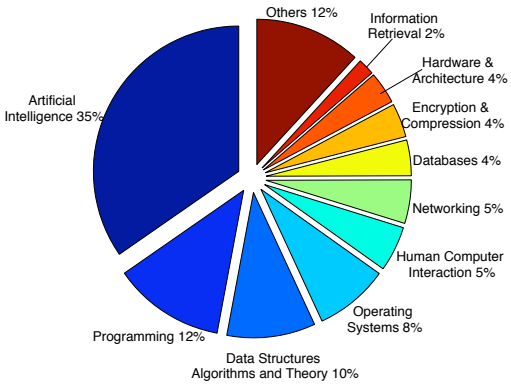


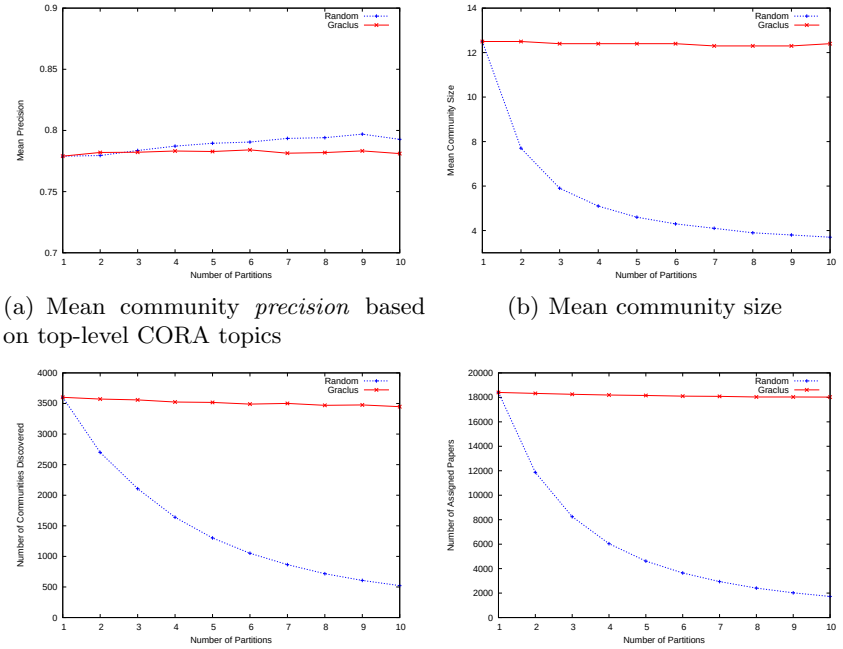
Fig. 14. Distribution of papers in top-level topics from the CORA dataset.

CORA graph. However it is interesting to note that the unequal distribution of topic sizes shown in Figure 14 did not affect the performance of the proposed strategy. As with the synthetic data, the strategy supports a significant reduction in computational cost without affecting our ability to accurately identify meaningful communities.

5.6 Weighted Graphs

We now consider the problem of searching for communities in large weighted graphs. As discussed in Section 2.1, a common approach used by standard community-finding algorithms in this context is to apply thresholding to transform a weighted network to a binary network. While increasing the threshold weight w can yield a more sparse graph with fewer edges, scalability is likely to remain an issue for networks with large numbers of nodes. Therefore we examined the effectiveness of the proposed problem decomposition strategy when working on a weighted network, specifically the CORA co-authorship network which consists of 24,951 unique authors.

In our experiments we considered threshold values of $w = 1$ and $w = 2$, so that an edge exists between two authors if they have co-authored at least w papers together. When applying CFinder on the two resulting graphs, the number of unique communities identified was 1,712 and 694 respectively (where 4,964 and 2,090 authors were assigned to at least one community). To evaluate the two-stage decomposition strategy on these graphs, we consider the ability of the strategy to retrieve the original CFinder communities as the number of Graclus partitions K increases. To provide a baseline we also applied CFinder after randomly partitioning the graphs. The plots in Figure 16 show the *recall* ($maxRecall_{avg}$) of both strategies on the two co-authorship graphs. As with the results on the paper-paper graphs discussed in Section 5.4, it is clear that the proposed two-stage strategy is highly successful at preserving CFinder com-



(a) Mean community *precision* based on top-level CORA topics

(b) Mean community size

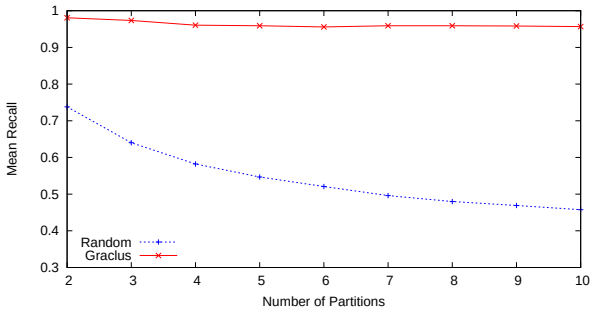
(c) Number of communities discovered

(d) Number of papers assigned to at least one community.

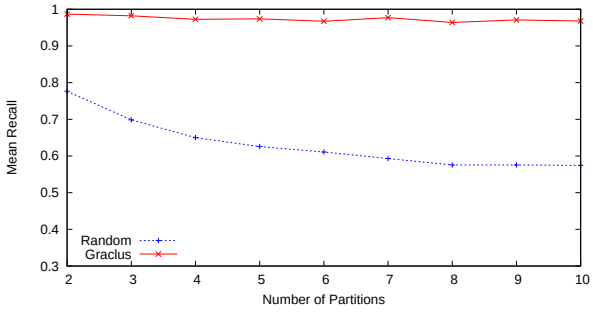
Fig. 15. Performance of CFinder after Graclus partitioning on the CORA paper-paper graph for increasing number of partitions K, compared with the performance of CFinder after random partitioning (averaged over 40 runs).

munities intact even as K increases. In contrast, applying random partitioning considerably impairs the ability of CFinder to find all k-cliques in the graphs, as reflected by the lower mean recall scores.

In these experiments we also observed behaviour similar to that shown in Figure 15, in terms of the number of unique communities found and the number of authors assigned to at least one community. For instance on the graph generated by setting $w = 1$, the proposed two-stage approach retrieves 1,388 communities covering 4,339 different authors at $K = 10$. For the same number of partitions the random approach only identifies 121 communities with assignments for 377 authors. These trends were even more pronounced for the case of $w = 2$. Here at $K = 10$ only 26 unique communities covering 83 authors were produced by CFinder after random partitioning, as opposed to 574 communities assigning 1,798 authors with our proposed strategy. Taken together with the results in Figure 16, these observations indicate that the combination of Graclus and CFinder is highly successful at retrieving the same communities produced by CFinder alone on thresholded weighted graphs.



(a) Threshold weight $w = 1$



(b) Threshold weight $w = 2$

Fig. 16. Comparison (in terms of *recall*) of the ability of CFinder after Graclus partitioning to recover the same communities identified by CFinder on the full thresholded CORA co-authorship graphs, with the performance of CFinder after random partitioning (averaged over 40 runs).

5.7 Summary

Since graph partitioning techniques are more scalable than community-finding algorithms, we have shown that the latter represents the "weak link" in the analysis process. Thus, our proposed strategy is feasible within some of the limitations of the community-finding approach. For instance, the authors of CFinder suggest that, if the most densely connected region of the network contains a large number of highly overlapping cliques, the computational performance of CFinder can suffer significantly. In our particular case (*i.e.* where CFinder is used as the community algorithm), the decomposition strategy can be somewhat limited in situations where communities grow in size over time. However, in many real networks, such as phone subscriber data, it will often be the case that more communities of similar size distribution emerge, rather than a "growth" in the size of communities. As illustrated by our evaluations on real and synthetic data, the two-stage strategy would be useful in this context, since it would permit discovery of communities in graphs larger than those which can be handled by CFinder alone. Our observations on both types of data clearly indicate that

applying Graclus prior to CFinder leads to a significant reduction in execution time, while also ensuring that there is also minimal loss of information overall on both unweighted and thresholded weighted graphs.

6 Conclusion

In this paper, we have proposed a two-stage problem decomposition strategy that facilitates the application of computationally expensive clique or community-finding algorithms to much larger networks than would be possible if these algorithms were applied on their own. The two-stage strategy involves using a top-down graph partitioning procedure to divide the network into smaller sub-networks, on which it is then feasible to apply a complementary, more computationally intensive community-finding procedure. We have demonstrated the usefulness of the proposed strategy in empirical evaluations on both artificial and real-world datasets, where the computational cost of the network analysis process was significantly reduced without adversely affecting the quality of the communities that were discovered. While we have considered the combination of Graclus and CFinder in this paper, it is possible that the two-stage strategy could be generalised to alternative combinations of algorithms. However, the effectiveness of alternatives would require evaluation.

References

1. Paliouras, G., Papatheodorou, C., Karkaletsis, V., Spyropoulos, C.D.: Clustering the users of large web sites into communities. In: Proc. 7th International Conference on Machine Learning (ICML'00). (2000) 719–726
2. Srivastava, J., Cooley, R., Deshpande, M., Tan, P.: Web usage mining: discovery and applications of usage patterns from Web data. ACM SIGKDD Explorations Newsletter **1** (2000) 12–23
3. He, Y., Cheung Hui, S.: Mining a Web Citation Database for author co-citation analysis. Information Processing and Management **38** (2002) 491–508
4. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Physical Review E **69** (2004)
5. Palla, G., Derenyi, I., Farkas, I., Vicsek, T.: Uncovering the overlapping community structure of complex networks in nature and society. Nature **435** (2005) 814–8
6. Gregory, S.: An algorithm to find overlapping community structure in networks. In: Proc. 11th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'07). (2007) 91–102
7. Abello, J., Pardalos, P.M., Resende, M.G.C.: On maximum clique problems in very large graphs. In: External Memory Algorithms. DIMACS Series in Discrete Mathematics and Theoretical Computer Science (1999) 119–130
8. Dhillon, I., Guan, Y., Kulis, B.: Weighted graph cuts without eigenvectors a multilevel approach. IEEE Transactions on Pattern Analysis and Machine Intelligence **29** (2007) 1944–1957
9. Karp, R.: Reducibility among combinatorial problems. Complexity of Computer Computations **43** (1972) 85–103

10. Bron, C., Kerbosch, J.: Finding all cliques of an undirected graph. Communications of the ACM **16** (1973) 575–577
11. Balas, E., Yu, C.S.: Finding a maximum clique in an arbitrary graph. SIAM J. Comput. **15** (1986) 1054–1068
12. Wood, D.R.: An algorithm for finding a maximum clique in a graph. Operations Research Letters **21** (1997) 211–217
13. Alon, N., Krivelevich, M., Sudakov, B.: Finding a large hidden clique in a random graph. In: 9th Symposium on Discrete Algorithms (SODA). (1998) 91–102
14. Östergård, P.: A fast algorithm for the maximum clique problem. Discrete Applied Mathematics **120** (2002) 197–207
15. Freeman, L.: Centrality in social networks: Conceptual clarification. Social Networks **1** (1979) 215–239
16. Granovetter, M.: The Strength of Weak Ties: A Network Theory Revisited. Sociological Theory **1** (1983) 201–233
17. Girvan, M., Newman, M.: Community structure in social and biological networks. PNAS **99** (2002) 7821–7826
18. Palla, G., Dernyi, I., Farkas, I., Vicsek, T.: Supplementary information: Uncovering the overlapping community structure of complex networks in nature and society. Nature (2005)
19. Shi, J., Malik, J.: Normalized cuts and image segmentation. In: Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR '97). (1997) 731–737
20. Chung, F.R.K.: Spectral graph theory. In: CBMS Conference on Recent Advances in Spectral Graph Theory. Number 92 in Regional Conference Series in Mathematics, California State University, Fresno (1994)
21. Pothen, A., Simon, H.D., Liou, K.P.: Partitioning sparse matrices with eigenvectors of graphs. SIAM Journal of Mathematical Analysis and Applications **11** (1990) 430–452
22. Chan, P., Schlag, M., Zien, J.: Spectral k-way ratio cut partitioning. IEEE Transactions CAD-Integrated Circuits and Systems **13** (1994) 1088–1096
23. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. **20** (1999) 359–392
24. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. The Bell system technical journal **49** (1970) 291–307
25. McCallum, A., Nigam, K., Rennie, J., Seymore, K.: Automating the construction of internet portals with machine learning. Information Retrieval Journal **3** (2000) 127–163
26. Erdős, P., Rényi, A.: On the evolution of random graphs. Bulletin of the Institute of International Statistics **38** (1961) 343–347
27. Barabasi, A.L., Albert, R.: Emergence of scaling in random networks. Science **286** (1999) 509–512
28. Hakimi, S.: On the Realizability of a Set of Integers as Degrees of the Vertices of a Graph,. SIAM J. Appl. Math. **10** (1962) 496–506
29. Narayanan, H., Belkin, M., Niyogi, P.: On the relation between low density separation, spectral clustering and graph cuts. Advances in Neural Information Processing Systems **19** (2007) 1025