Graph sub-sampling for divide-and-conquer algorithms in large networks

Eric Yanchenko
Akita International University

September 12, 2024

Abstract

As networks continue to increase in size, current methods must be capable of handling large numbers of nodes and edges in order to be practically relevant. Instead of working directly with the entire (large) network, analyzing sub-networks has become a popular approach. Due to a network's inherent inter-connectedness, sub-sampling is not a trivial task. While this problem has gained attention in recent years, it has not received sufficient attention from the statistics community. In this work, we provide a thorough comparison of seven graph sub-sampling algorithms by applying them to divide-and-conquer algorithms for community structure and core-periphery (CP) structure. After discussing the various algorithms and sub-sampling routines, we derive theoretical results for the mis-classification rate of the divide-and-conquer algorithm for CP structure under various sub-sampling schemes. We then perform extensive experiments on both simulated and real-world data to compare the various methods. For the community detection task, we found that sampling nodes uniformly at random yields the best performance. For CP structure on the other hand, there was no single winner, but algorithms which sampled core nodes at a higher rate consistently outperformed other sampling routines, e.g., random edge sampling and random walk sampling. The varying performance of the sampling algorithms on different tasks demonstrates the importance of carefully selecting a sub-sampling routine for the specific application.

1 Introduction

Graphs, or networks[1], provide a simple and intuitive model for interconnected systems where objects or entities are represented as nodes and their relationships are denoted as edges. Graphs have been applied to many real-world applications, ranging from friendships (Girvan and Newman, 2002) to brain connectivity (Telesford et al., 2011). As their popularity has grown, so too has their size: it is not uncommon for researchers to now work with thousands, millions, or even billions of nodes and edges (e.g., Backstrom et al., 2006; Rozemberczki et al., 2019). Thus, modern network analysis methods must scale to such large datasets if they are to be practically useful.

[1]We will use these terms interchangeably in this work.

One popular approach for working with large data sets in general, and large graphs in particular, is sub-sampling. The idea is simple: first the researcher divides the data into smaller sub-sets before performing analyses on the sub-sets. The analyses are then combined to give a result on the original (larger) data set. Generally, performing an analysis multiple times on smaller data sets is much faster than applying it once to the entire dataset. While sub-sampling and divide-and-conquer approaches have found great success across various statistical, machine learning and computer science tasks (Jordan, 2012), there are some unique challenges to applying these techniques to graph-valued data. The main difficulty is that the nodes and edges which make up the graph are, by their very nature, dependent. As a graph can have a highly complex topology, it is unclear if these features will be preserved in the sub-graphs, and if not, it is unclear how that affects the analysis. Thus, it is paramount to understand the effect of the graph sub-sampling routine on the task of interest.

Despite its importance, graph sampling is still an open-problem, and, in particular, has received less attention from the statistics community. Leskovec and Faloutsos (2006) study the similarity between various measures computed on the original graph and their sub-graphs, including: clustering coefficients, degree distributions, largest eigenvectors and more. The authors find that while edge sampling performs poorly and random node sampling performs surprisingly well, random walk and forest fire samplers yield sub-networks most similar to the original network. Recently, Rozemberczki et al. (2020) develop a package to easily implement various sub-sampling routines. The new package is then applied to a variety of tasks, from computing average degrees to node classification. To our knowledge, their are not any works studying the effects of graph sub-sampling from a statistical angle, nor specifically studying their effect on divide-and-conquer algorithms.

In this work, we seek to close this gap by comparing graph sub-sampling methods on divide-and-conquer algorithms. We focus on two of the most important meso-scale structures: community structure (Newman, 2006), where nodes cluster into highly connected groups, and core-periphery structure (Yanchenko and Sengupta, 2023), where nodes are either in the densely-connected core or sparsely-connected periphery. Our main contribution is a thorough comparison, both theoretically and empirically, of seven sub-sampling routines on the performance of divide-and-conquer algorithms for identifying these structures. After introducing the methods, we recap the theoretical results from Mukherjee et al. (2021) before deriving novel theoretical results for the CP method of Yanchenko (2022). In particular, we derive the general mis-classification rate of the algorithm and give specific results for several sub-sampling methods. Finally, we apply these methods to various synthetic and real-world networks. We find that random node and random node neighbor yield the best community structure identification results, while routines which sample core nodes with a high probability, e.g., edge sampling and random walk, perform the best for the CP identification task. Indeed, the sub-sampling routines that perform well for one task, generally did not perform as well on the other, underscoring the importance of carefully choosing the sub-sampling algorithm for the specific problem.

The layout of the rest of the paper is as follows. In Section 2 we introduce the divide-and-conquer algorithms as well as the various sub-sampling routines. Section 3 is devoted to theoretical results while Sections 4 and 5 apply the methods to simulated and real-world networks, respectively. Finally, we share concluding thoughts in Section 6.

2 Methodology

We begin by discussing the divide-and-conquer algorithms for community structure and CP structure, followed by presenting seven graph sub-sampling algorithms.

2.1 Divide-and-conquer for community structure

Mukherjee et al. (2021) propose PACE, a divide-and-conquer algorithm to identify community structure in networks. The algorithm randomly samples sub-graphs, and then applies a community detection algorithm to each sub-graph. This step repeats many times before the more challenging step of "stiching" together the results from the sub-samples. Since community labels are not identifiable, this combining step is non-trivial. To avoid this problem, the authors propose an estimate of the clustering matrix C instead of directly estimating the community labels as the clustering matrix is unique. This clustering matrix can then be used to extract the community labels.

More specifically, let A be the $n \times n$ adjacency matrix corresponding to graph G with m total edges.[2] Let $Z \in \{0,1\}^{n \times K}$ correspond to the true community labels where $Z_{ik} = 1$ if node i is in cluster k for $k = 1, \ldots, K$ communities. We assume that K is known. We can define $C = ZZ^T \in \{0,1\}^{n \times n}$ as the clustering matrix where $C_{ij} = 1$ if nodes i and j are in the same community and 0 otherwise. Notice that C does not depend on the exact community labels (which are equivalent up to permutation) but only on the pair-wise relationship between nodes.

PACE begins by randomly sampling a sub-graph S_b of graph G with qn nodes where $q \in (0,1)$ is the proportion of nodes in the sub-sample. With sub-sample in hand, we can apply any community detection algorithm to this sub-sample, e.g., spectral (Ng et al., 2002), modularity maximization (Girvan and Newman, 2002), semi-definite programming (Cai and Li, 2015), etc. The resulting membership vectors of these clustering algorithms will allow us to construct $\hat{C}^{(b)} \in \{0,1\}^{n \times n}$ where $\hat{C}^{(b)}_{ij} = 1$ if both nodes i and j appear in sub-graph S_b and were assigned to the same community and 0 otherwise. We repeat this process for B sub-graphs, yielding $\hat{C}^{(1)}, \ldots, \hat{C}^{(B)}$.

The key step in this algorithm is combining the results from the B sub-graphs. Let N_{ij} be the number of times that both nodes i and j were chosen in the same sub-sample. Then the combined estimator $\hat{C} \in [0,1]^{n \times n}$ of the clustering matrix is defined by

$$\hat{C}_{ij} = \mathbb{I}(N_{ij} > \beta) \frac{\sum_{b=1}^{B} \hat{C}^{(b)}_{ij}}{N_{ij}} \tag{1}$$

where $1 \leq \beta \leq B$ is a tuning parameter. The combined estimator $\hat{C}$ is the proportion of sub-samples where nodes i and j were both sampled and assigned to the same community. We compute this quantity, however, only for the node pairs which were sampled a sufficient number of times ($> \beta$ times), otherwise the estimate is set to 0. Lastly, we can apply a clustering algorithm, e.g., k-means, to $\hat{C}$ in order to obtain the final community membership vector. The full algorithm is reported in Algorithm 1.

[2] In this work, we only consider undirected, unweighted networks without self-loops, but the ideas can easily be generalized.

Algorithm 1 PACE for community structure

Result: Community membership vector $\hat{\boldsymbol{c}}$

Input: $n \times n$ adjacency matrix A, number of communities K, proportion of nodes to sub-sample q, number of sub-samples drawn B, sub-sampling scheme $\mathcal{S}$, clustering algorithm $\mathcal{A}$, tuning parameter β

for B *times* **do**

  Randomly sample sub-network $S^{(b)}$ using $\mathcal{S}$

  Apply $\mathcal{A}$ to $S^{(b)}$

  Obtain clustering matrix $\hat{C}^{(b)} \in \{0,1\}^{qn \times qn}$

end

Compute N where N_{ij} is the number of times nodes i and j were sampled together in a subgraph.

Compute $\hat{C}$ where $\hat{C}_{ij} = \mathbb{I}(N_{ij} > \beta)\frac{\sum_{b=1}^{B} \hat{C}_{ij}^{(b)}}{N_{ij}}$

Perform k-means clustering on $\hat{C}$ to get final community membership vector $\hat{\boldsymbol{c}} \in \{1, \ldots, K\}^n$

The key advantage of this algorithm is that it estimates C, the clustering matrix, instead of directly estimating the community memberships Z, as the clustering matrix is independent of label permutations. Moreover, the estimate of $\hat{C}_{ij}$ is intuitive, namely the number of times nodes i and j were assigned to the same community divided by the number of times they were sampled together, given they were sampled together a sufficient number of times. Checking if $N_{ij} > \beta$ acts as a smoothing step by discarding the estimates for node pairs which only appeared in a few sub-graphs. Of course, β is a tuning parameter that must be chosen by the user. As suggested by the authors, we set β to be the 40th percentile of N_{ij}.

The main focus of this paper is the sub-sampling step of the divide-and-conquer algorithm, a topic that the authors of PACE give some attention too (see Section 2.3 of Mukherjee et al. (2021)). In particular, they suggest randomly sampling nodes, finding h-hop neighbors, onion neighborhoods or sub-graphs with roots at high degree nodes as possible sub-sampling schemes, but for nearly all experiments, they simply use random node sampling. Finally, note that Mukherjee et al. (2021) also propose another divide-and-conquer algorithm, GALE, but for simplicity we will only consider PACE. Indeed, in general the authors showed the superior performance of PACE compared to GALE, as well as GALE performing best with random node sampling.

2.2 Divide-and-conquer for core-periphery structure

We now turn our attention to the divide-and-conquer algorithm for detecting CP structures in large networks proposed by Yanchenko (2022).

2.2.1 Core-periphery metric

Before presenting the divide-and-conquer algorithm, we first discuss the metric from Borgatti and Everett (2000) (BE) used to quantify the CP structure of a network. The authors find the correlation between the observed network and a network with "ideal" CP structure. Let A again be the adjacency matrix for the observed network and $\boldsymbol{c} \in \{0,1\}^n$ be the CP labels

where $c_i = 1$ if node is in the core, and 0 otherwise. Then the BE metric is

$$\rho(A, \boldsymbol{c}) = \mathsf{Cor}(A, \Delta_{\boldsymbol{c}}) \tag{2}$$

where $\mathsf{Cor}(A, B)$ is the Pearson correlation of the vectorized upper-triangles of matrices A and B, and $\Delta_{\boldsymbol{c}}$ represents the ideal CP structure with $(\Delta_{\boldsymbol{c}})_{ij} = c_i + c_j - c_i c_j$. In words, $(\Delta_{\boldsymbol{c}})_{ij} = 1$ if either node i or j is in the core, and 0 otherwise. The metric in (2) can be optimized using the greedy algorithm proposed in Yanchenko (2022) to approximate

$$\tilde{\boldsymbol{c}} = \arg \max_{\boldsymbol{c} \in \{0,1\}^n} \mathsf{Cor}(A, \Delta_{\boldsymbol{c}}). \tag{3}$$

2.2.2 Divide-and-conquer algorithm

With a metric to quantify the CP structure of a network, we can present the divide-and-conquer algorithm. We slightly modify the algorithm in Yanchenko (2022) for reasons that will be plain later. The idea is to find the CP structure on small sub-networks of the network and then combine these results together to yield the CP labels on the entire network. Specifically, let A be the adjacency matrix with n nodes, $q \in (0, 1)$ be the proportion of nodes to sub-sample and B the number of sub-sample to draw. First, we sample a sub-graph with qn nodes using some graph sub-sampling routine, and find the optimal CP labels from (3). We repeat this process B times and output $\hat{\boldsymbol{c}} \in [0, 1]^n$ where $\hat{c}_i$ is the number of times node i was assigned to the core over the total number of samples B. The full algorithm is presented in Algorithm 2.

Algorithm 2 Divide-and-conquer for core-periphery structure

Result: Core-periphery proportions $\hat{\boldsymbol{c}}$

Input: $n \times n$ adjacency matrix A, proportion of nodes to sub-sample q, number of sub-samples drawn B, sub-sampling scheme $\mathcal{S}$

Initialize $\boldsymbol{x} = \boldsymbol{0}_n$ where x_i is the number of times node i is assigned to the core for $i = 1, \ldots, n$

for B *times* **do**

  Randomly sample sub-network $S^{(b)}$ using $\mathcal{S}$ with nodes $V = \{v_1, \ldots, v_{qn}\}$

  Obtain CP labels of $S^{(b)}$, $\hat{\boldsymbol{c}}^{(b)} \in \{0, 1\}^{qn}$

  $x_{v_i} = x_{v_i} + 1$ if $\hat{c}_i^{(b)} = 1$

end

$\hat{c}_i = x_i / B = \#$ of times node i was assigned to core / # of sub-samples

This algorithm is slightly different from the original paper. Originally, the output was the number of times node i was assigned to the core, divided by the number of times that node i was sampled. Indeed, this approach more closely resembles the stitching routine of the PACE algorithm. The reason that PACE divides by the number of times that nodes i and j were sub-sampled together, N_{ij}, and then only keeps the estimate if $N_{ij} > \beta$ is that the estimate $\hat{C}_{ij}$ would be unreliable if it was based only on a few sub-samples. For the CP task, however, if a node is sub-sampled infrequently, this gives us *more* information, rather than less. Indeed, core nodes are those that are more central or influential in the

network. Therefore, if a node is sampled infrequently, then it is unlikely to be important and also unlikely to be a part of the core. Thus, dividing by the total number of samples will decrease the estimate of $\hat{c}_i$ for infrequently sampled nodes. Implicitly, this means we should favor sub-sampling methods which have a high probability of sampling core nodes. As we will see, this small change not only makes proving theoretical statements easier, but also improves the empirical performance of the algorithm.

2.3 Sub-graph sampling

Finally, we discuss seven sub-sampling algorithms. We select a variety of sampling schemes including: node-based (random and proportional to degree); edge-based (random) and exploration (breadth-first search, depth-first search, random node-neighbor and random walk).

Random Node (RN): The simplest sampling method is to randomly sample nodes without replacement. Each node has an equal probability of being selected, and once all qn nodes are chosen, then the edges between these nodes make up the sub-graph. This method was originally considered in both Mukherjee et al. (2021) and Yanchenko (2022).

Degree Node (DN): In this method, nodes are again sampled at random, but now their probability of being sampled is proportional to their degree. Once the qn nodes are sampled, then the sub-graph is completed with all associated edges between sampled nodes. Mathematically, this means sampling without replacement from a distribution with probability mass function $\mathsf{P}(Z = j) = d_j / \sum_k d_k$ where d_j is the degree of node j.

Random Edge (RE): Instead of randomly sampling nodes, this approach samples edges uniformly at random. Edges are drawn with equal probability and the two nodes connected by the sampled edge are added to the sub-graph. This process continues until qn nodes have been sampled, at which point the remaining edges between sampled nodes are also added. Adding the remaining edges between sampled nodes is sometimes called the Induction Step (Ahmed et al., 2013) and will be used in all remaining algorithms as well. While edges can be selected with equal probability, if the graph has edge weights than this could be used to for a sampling scheme analogous to DN.

Breadth first search (BFS): The previous three methods randomly sampled nodes or edges. For the remaining methods, we will consider graph exploration approaches. The BFS algorithm begins by sampling a node at random. Then all neighbors of this sampled node are included in the sub-graph as well. Next, all neighbors of the neighbors are also included in the sub-sample. This process continues until we have sampled qn nodes.

Depth first search (DFS): DFS can be considered as the complement of a BFS algorithm. This method also begins by randomly sampling a starting node. Then instead of including all neighbors of this starting node, it randomly chooses one neighbor. From there, it randomly chooses one neighbor of the neighbor. The process continues until qn nodes

are sampled, or the search terminates as there are no more nodes to traverse. If the latter happens, a new node is sampled to start the process again until qn nodes have been reached.

Random node-neighbor (RNN): This approach shares similarities to BFS. As in BFS, a node is sampled at random and then all of its neighbors as also added to the sub-graph. Instead of looking at the neighbors of neighbors as in BFS, RNN then samples a new node and includes all of the new node's neighbors. This continues until qn nodes have been sampled.

Random walk (RW): In our final algorithm, we traverse the network through a random walk. Specifically, we begin by sampling a node uniformly at random. Then from this node, one of its neighbors is sampled at random, and this process continues until the walk has traversed qn nodes.

3 Theoretical Results

In this section, we present theoretical results for each algorithm. As this paper's focus is on the sub-sampling step, we will primarily focus on the role of the sampling scheme on the theoretical results. Additionally, Mukherjee et al. (2021) devoted significant attention to the theoretical properties of PACE, so we only briefly discuss these before devoting the majority of the space to new theoretical results on the CP divide-and-conquer algorithm.

3.1 PACE algorithm

The goal of the theoretical results for PACE is to show that the estimated clustering matrix $\hat{C}$, is "close" to the true clustering matrix C which generated the network. To formalize this notion, let $\tilde{\delta}(C^*, \hat{C})$ be the difference between the true matrix C^* and estimate $\hat{C}$ where

$$\tilde{\delta}(C, C') = \frac{1}{n^2}||C - C'||_F^2$$

and $||\cdot||_F$ is the Frobenius norm. Thus, $\tilde{\delta}(C, \hat{C})$ is the mis-clustering rate of the PACE algorithm. In the following theorem, the authors bound the expectation of this term under a general community detection algorithm and sampling scheme.

Theorem 3.1 from Mukherjee et al. (2021): *Given some network A with K communities where the largest community has $p_{max}n$ nodes, and some sampling scheme $\mathcal{S}$, let S be a randomly chosen sub-graph of the network. Moroever, let $\hat{C}$ be the estimated clustering matrix returned by* PACE*. Then the expected mis-clustering rate, $\mathbb{E}\tilde{\delta}(C, \hat{C})$ can be bounded by:*

$$\mathbb{E}\tilde{\delta}(C^*, \hat{C}) \leq \frac{B}{\beta n^2}\mathbb{E}||\hat{C}^{(S)} - C^{(S)}||_F^2 + p_{max}\max_{i,j}\mathbb{P}(N_{ij} < \beta) \tag{4}$$

where $\hat{C}^{(S)}$ and $(C^*)^{(S)}$ are the estimated and true clustering matrices, respectively, restricted to sub-graph S.

See Mukherjee et al. (2021) for the proof of this result. There are two main parts to the bound in (4). The first term quantifies the performance of the community detection algorithm on a randomly chosen sub-graph S which is mainly a function of the clustering algorithm. On the other hand, the second term measures how well the graph is covered by the sub-samples, and depends only on the sub-sampling procedure.

As our interest is on the sub-sampling step, we focus on this second term. Indeed, the authors of PACE provide two special cases of the results under the RN sampling scheme and Ego neighborhoods. As we do not discuss ego neighborhoods, we omit discussion of this result. The following gives the results for RN sampling.

Corollary 3.1 from Mukherjee et al. (2021): *Assume that nodes are chosen uniformly at random to construct the subgraphs. Then*

$$p_{max} \max_{i,j} \mathbb{P}(N_{ij} < \beta) = O(e^{-\kappa Bp}) \tag{5}$$

where $p = q^2(1 + o(1))$, $\beta = \theta Bp$ *for* $\theta \in (0, 1)$ *and* $\kappa = (1 - \theta)^2/2$.

This result shows that the mis-clustering error coming from the sub-sampling step decreases exponentially with B, the total number of sub-samples drawn, as well as q, the proportion of nodes in each sub-graph. As the authors note, deriving the specific result for other sub-sampling routines is difficult, so we do not consider them here.

3.2 CP algorithm

Using similar ideas to that of Mukherjee et al. (2021), we present novel theoretical results for the CP divide-and-conquer algorithm. Similar to the previous sub-section, we are interested in bounding the expected mis-classification error.

3.2.1 General mis-classification result

First, we derive the expression for the general mis-classification rate. Recall from Algorithm 2 our core estimates $\hat{\boldsymbol{c}}$, where

$$\hat{c}_i = \frac{1}{B} \sum_{b=1}^{B} \hat{c}_i^{(b)}$$

and $\hat{\boldsymbol{c}}^{(b)}$ are the CP labels from sub-sample b, i.e., $\hat{c}_i^{(b)} = 1$ if node i was assigned to the core in sub-sample b, and 0 otherwise. Then we define

$$\delta(\hat{\boldsymbol{c}}, \boldsymbol{c}^*) = \frac{1}{n} ||\hat{\boldsymbol{c}} - \boldsymbol{c}^*||_2^2 = \frac{1}{n} \sum_{i=1}^{n} (\hat{c}_i - c_i^*)^2$$

to quantify the mis-clustering of the algorithm. The following result is analogous to Theorem 3.1 from Mukherjee et al. (2021).

Theorem 3.2: *Given some network A and sampling scheme $\mathcal{S}$, let S be a randomly chosen sub-graph. Let $\hat{\boldsymbol{c}}$ be estimated CP labels using the CP divide-and-conquer algorithm. Then the expected mis-clustering rate, $\mathbb{E}\delta(\hat{\boldsymbol{c}}, \boldsymbol{c}^*)$ can be bounded by*

$$\mathbb{E}\delta(\hat{\boldsymbol{c}}, \boldsymbol{c}^*) \leq \frac{1}{n}\left(\mathbb{E}||\hat{\boldsymbol{c}}^{(S)} - (\boldsymbol{c}^*)^{(S)}||_2^2 - \mathbb{E}\sum_{i=1}^{n} y_i^{(S)}\right) \tag{6}$$

where $\hat{\boldsymbol{c}}^{(S)}, (\boldsymbol{c}^*)^{(S)} \in \{0,1\}^{qn}$ are the estimated and true CP labels, respectively, restricted to sub-graph S and $y_i^{(S)} = 1$ if node i is in the core and was not sampled in sub-graph S, and 0 otherwise.

Please see the Supplemental Materials for all proofs in this sub-section. Similar to the results of PACE, our error bound has two terms; the first which primarily depends on the CP algorithm, and the second which only depends on the sub-sampling routine. Indeed, if the expected fraction of mis-classified nodes and the expected fraction of core nodes that are not sampled both go to 0 for large n, then this entire term will converge to 0.

We focus on the second term which is the expected number of core nodes which are not sampled in a given sub-graph. First, this term shows the importance of using a sub-sampling routine which samples core nodes with high probability. Indeed, if the probability of a core node being sampled increases, then this term decreases. Secondly, this term notably differs from the analogous term in the PACE theory. The CP algorithm divides the estimates by B, the total number of sub-samples, instead of the number of sub-graphs for which a node was sampled as in the PACE algorithm. Because of this, an error is guaranteed if a core node is not sampled. This choice will make the theory easier to derive, but puts us at risk of greater error. The key observation is that in real-world networks, the number of the core nodes is typically much smaller than the number of peripheral nodes. Indeed, in practice we found that dividing by B instead of the number of times a node was sub-sampled led to improved performance.

3.2.2 Specific sub-sampling results

The results of Theorem 3.2 are agnostic to the network-generating model, CP detection algorithm, and sub-sampling scheme. By assuming a particular model, we can find the analytic expression for the second term in (6) for the majority of the sub-sampling schemes in Section 2.3.

We assume networks are generated with a CP structure from the stochastic block model (SBM) (Holland et al., 1983). For adjacency matrix A, let $A_{ij} \overset{ind.}{\sim} \mathsf{Bernoulli}(\varrho_n P_{ij})$ where P corresponds to a SBM that does not depend on n, and $\varrho_n \to 0$ is a sparsity-inducing parameter. Furthermore, if $\boldsymbol{c}^*$ correspond to the true CP labels, then

$$P_{ij} = \begin{cases} p_{11} & c_i^* = c_j^* = 1 \\ p_{22} & c_i^* = c_j^* = 0 \\ p_{12} & \text{otherwise} \end{cases}.$$

If $p_{11} > p_{12} > p_{22}$, then this model generates networks with a CP structure, so we call it the CP-SBM. We may also assume that $\alpha_n = k/n$, the proportion of core nodes, goes to zero to model the empirical observation that the core is typically smaller than the periphery.

In the following Corollary, we find an analytic expression for the expected number of core nodes that are not sampled in each sub-graph. We assume that n is large enough such that the probability of sampling the same node twice is negligible. The result is in Corollary 3.2 with proof in the Supplemental Materials.

Corollary 3.2: *Let A be generated from a CP-SBM with n nodes and k core nodes. Let S be a sub-graph of A of size qn where $0 \leq q \leq 1$ sampled using $\mathcal{S}$ where $\mathcal{S}$ is one of the sub-sampling schemes from this paper. If $y_i^{(\overline{S})}$ is the event that node i is a core node and not sampled in S, then*

$$\mathbb{E}\frac{1}{n}\sum_{i=1}^{n} y_i^{(S)} = (1 - q\xi_n^{(\mathcal{S})})\frac{k}{n} \tag{7}$$

where $\xi_n^{(\mathcal{S})}$ takes the expression listed in Table 1.

$\mathcal{S}$	$\xi_n^{(\mathcal{S})}$	$\lim_{n\to\infty} \xi_n^{(\mathcal{S})}$
RN	1	1
DN	$\frac{n\{(k-1)p_{11}+(n-k)p_{12}\}}{k(k-1)p_{11}+2k(n-k)p_{12}+(n-k)(n-k-1)p_{22}}$	$\frac{p_{12}}{p_{22}}$
RE	$\frac{n\{(k-1)p_{11}+(n-k)p_{12}\}}{k(k-1)p_{11}+2k(n-k)p_{12}+(n-k)(n-k-1)p_{22}}$	$\frac{p_{12}}{p_{22}}$
RNN	$\frac{1+(k-1)p_{11}+(n-k)p_{12}}{1+\frac{k}{n}\{(k-1)p_{11}+(n-k)p_{12}\}+\frac{n-k}{n}\{kp_{12}+(n-k-1)p_{22}\}}$	$\frac{p_{12}}{\alpha_n p_{12}+p_{22}}$
RW	$*$	$*$

Table 1: Theoretical results for sub-sampling scheme in CP divide-and-conquer algorithm.

The expression for RW (*) appear in the Supplemental Materials.

The proof carefully computes the expected number of cores nodes that are sampled in each sub-graph. For RN, DN and RE, we express the number of core nodes sampled as a hyper-geometric distribution. For RNN, the number of core nodes sampled is conditional on whether the first node drawn is from the core or periphery, so we consider these two cases. Lastly, we use a recursive relationship to model the RW case.

In (7), we see the general result for the average number of core nodes that are not sampled in a given sub-graph. As expected, as size of the sub-graphs increases (q), this term decreases. Moreover, this expression depends on $k/n = \alpha_n$. Assuming that the periphery dominates the core, this entire term will go to zero regardless of the sampling scheme which theoretically validates our decision to divide the core estimates by B. Of course, the sub-sampling routine will still affect the performance for finite n.

In Table 1 we present the form of the expression in (7) for five of the seven routines from Section 2.3. From (7), the overall error is smaller when $\xi_n^{(\mathcal{S})}$ is larger so large values of $\xi_n^{(S)}$ are to be preferred. BFS and DFS are difficult to analyze so these are omitted, and

RW has a complicated expression that we relegate to the Supplemental Materials. For the other sub-sampling schemes, we first notice that DN and RE yield identical expressions. This means that these sampling schemes are equivalent in terms of the number of core nodes that are expected to sample in each sub-graph when the network is generated from a CP-SBM. This does not mean that they will perform identically as the sub-sampling scheme affects the topology of the sub-graph S which may affect the first term in (6). Next, the exact expressions for $\xi_n^{(\mathcal{S})}$ are a bit cumbersome, so it is helpful to look at the asymptotic expression. Indeed, for DN and RE, this becomes p_{12}/p_{22}, the ratio of the core-periphery edge probability to the periphery-periphery edge probability. In a CP-SBM, we have $p_{12} > p_{22}$ so $p_{12}/p_{22} > 1$. This means that DN and RE have a uniformly smaller expected error than RN. Additionally, as the strength of the CP structure of the network increases, the error will decrease because p_{12}/p_{22} increases. In other words, we expect a lower error rate if the network exhibits a stronger CP structure. Finally, for the asymptotic results of RNN, $p_{12}/(\alpha_n p_{12} + p_{22} < p_{12} < p_{22}$. Thus, DN and RE are expected to have a lower error than that of RNN. Moreover, if $(1 - \alpha_n)p_{12} < p_{22}$, then RNN is also expected to have a worse error rate than that of RN.

It is important to highlight that these results and discussion apply only to the second term in (6). It is possible that a sub-sampling method may not sample core nodes as often, but has a lower mis-classification rate on the sub-graph, leading to a smaller value of the first term in (6), or vice-versa. Studying this term is more difficult, especially for methods besides RN, so we leave this as an avenue of future work.

4 Simulation Study

In this section, we carry out extensive simulations of both divide-and-conquer algorithms. The goals of this section are to compare the performance of the algorithms under different sub-sampling schemes and see if there exists an optimal sub-sampling method. Additionally, we also hope to empirically validate the CP theoretical results. All experiments in this and the subsequent section were carried out in `R` on a MacBook Pro with Apple M3 chip, 16 GB of memory parallelized using 10 cores. All code is available on the author's GitHub: `https://github.com/eyanchenko/subsamp`.

4.1 Community detection simulations

We begin by comparing the performance of PACE with different sub-sampling algorithms. By generating networks with a known community structure, we can compare the outputted membership vectors with the true clusters.

4.1.1 Settings

There are six simulations settings. In each, we generate networks, apply Algorithm 1 and compare the estimated membership vector with the true labels using the adjusted rand index (ARI). ARI provides a measure of concordance between true and estimated labels with 1

Setting	n	p_{11}	κ_1	B	qn
1	5000	$(\mathbf{0.02}, \mathbf{0.10})$	0.75	1000	250
2	$(\mathbf{1000}, \mathbf{10,000})$	0.04	0.75	1000	250
3	$(\mathbf{1000}, \mathbf{10,000})$	0.04	0.75	$\mathbf{\frac{1}{2}n}$	250
4	5000	0.04	$(\mathbf{0.50}, \mathbf{0.95})$	1000	250
5	5000	0.04	0.75	$(\mathbf{100}, \mathbf{10,000})$	250
6	5000	0.04	0.75	1000	$(\mathbf{100}, \mathbf{500})$

Table 2: Simulation settings for PACE divide-and-conquer algorithm. n is the number of nodes in the network; p_{11} is the intra-community edge probability; κ_1 is the proportion of nodes in community 1; B is the number of sub-graphs sampled; and qn is the number of nodes per sub-graph. The parameters which vary are highlighted in **bold**.

being the largest value and indicating a perfect match (up to permuting the label names). We repeat this process for 100 MC iterations and record the average ARI.[3]

We generate networks with a community structure by using a $K = 2$-block SBM. Recall from Section 3, if A is the corresponding adjacency matrix with data-generating model P, then $A_{ij} \overset{ind.}{\sim} \mathsf{Bernoulli}(P_{ij})$ where $P_{ij} = p_{c_i,c_j}$. We take $p_{11} = p_{22} > p_{12}$ to generate networks with assortative community structure where $p_{11} = p_{22}$ and p_{12} represent the intra-community and inter-community edge probabilities, respectively. We set $p_{12} = 0.01$ for all experiments in this sub-section.

An overview of the simulation settings is available in Table 2. Setting 1 looks at the effect of the intra-community edge probability where larger values indicated stronger community structure. Settings 2 and 3 increase the number of nodes in the network with B held fixed and increasing with n, respectively. The proportion of nodes in one community is the subject of setting 4 where greater imbalance in the community sizes should make the task more difficult. Finally, settings 5 and 6 look at the hyper-parameters in the PACE model, namely the number of sub-graphs drawn and size of sub-graphs, respectively. For all experiments, we use the fast greedy algorithm (Clauset et al., 2004) as our base community detection algorithm as in Zhang et al. (2022).

4.1.2 Results

The results are in Figure 1. In Setting 1, save BFS and DFS, all methods yield an increasing ARI as p_{11} increases. When $p_{11} \leq 0.04$, RNN has the largest ARI while RN does the best when $p_{11} > 0.04$. BFS improves at first before quickly dropping off, while DFS yields extremely low ARI values. Indeed, in all settings, it is clear that DFS performs poorly. Turning to setting 2, most methods' ARI increases with n until $n = 5000$ before decreasing. By $n = 9000$, these methods experience a steep drop off in performance. When B also increases with n as in setting 3, DN, RE, RN, RNN and RW all have increasing ARI, with no clear distinction.

[3]The run-time was also recorded but we do not report it. As some sub-sampling routines used optimized functions from `R` packages while others are written by hand, it is not a fair comparison. In general, the exploration-based sub-sampling methods will be slower.

The effects of the distribution of nodes between communities is evident in setting 4. Indeed, as the nodes are more disproportionately allocated between groups, each method decreases in performance with RN being the most negatively affected. As B increases in setting 5, all methods, save DFS, have an improvement in performance. Interestingly, while RN and RNN plateau at an ARI of 1, DN, RE, RW and BFS all begin to yield slightly decreasing values of ARI for $B > 2500$. Finally, setting 6 shows that as the number of nodes in the sub-samples increases, the communities are easier to detect for all methods except BFS and DFS. RN has the largest ARI followed by RNN for most values of q.

4.1.3 Discussion

From the results of these simulations, it appears that RN and RNN perform the best with RW, RE and DN also yielding good results. Even more clearly, the results demonstrate that BFS and DFS are poor samplers to use with the PACE algorithm. Setting 2 reveals the importance of increasing the number of sub-samples as n increases, even though the theoretical results depend on a fixed B. Settings 5 and 6 show that B and q must be carefully chosen, where small values can severely hinder performance. Finally, all methods suffer when the nodes are unequally distributed between communities. This is not surprising as the identification problem is much more challenging in this setting. Indeed, the presence of small communities likely requires the increase in the number of nodes per sub-sample in order to capture the structure.

4.2 Core-periphery detection simulations

Similar to the previous sub-section, we now compare the performance of the various sub-sampling routines from Section 2.3 on the CP divide-and-conquer algorithm.

4.2.1 Settings

We again look at six simulation settings. For each setting, we generate a network, apply Algorithm 2 and compute the AUC between the outputted proportions and true labels. We vary different aspects of the data-generating model as well as the algorithm hyper-parameters. All networks are generated from a CP-SBM. The networks are generated the same as in the community detection simulations except now $p_{11} > p_{12} > p_{22}$ where p_{11}, p_{12}, p_{22} represent the core-core, core-periphery, and periphery-periphery edge probabilities, respectively. Each setting is repeated for 100 MC replications and the average AUC is reported.

The simulation settings are summarized in Table 3. For each setting, we fix $p_{22} = 0.001$ and set $p_{12} = \frac{1}{2}p_{11}$. In setting 7, we vary p_{11} where larger values correspond to a greater strength of a CP structure. Settings 8 and 9 look at the effect of increasing the network size where the former fixes B while the latter allows B to increase with n. The effect of the core size is studied in setting 10. Choice of hyper-parameters are investigated in settings 11 and 12, varying the values of B and q, respectively.

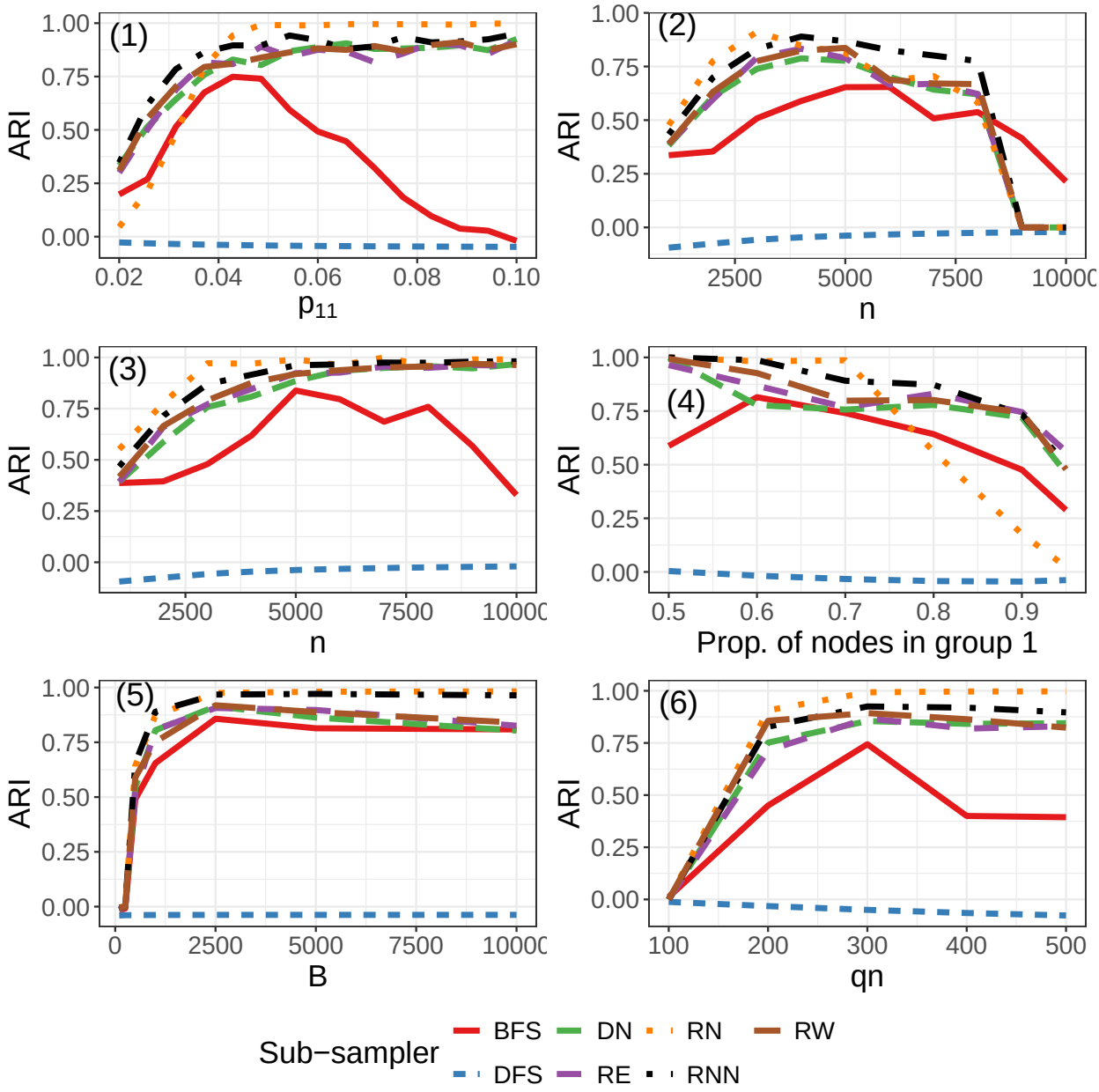

Figure 1: Community detection simulation results. The number in the upper-left corner corresponds to the simulation setting (1-6).

Setting	n	p_{11}	α_n	B	qn
7	5000	$(\mathbf{0.002}, \mathbf{0.020})$	0.01	1000	100
8	$(\mathbf{1000}, \mathbf{10,000})$	0.004	0.01	1000	100
9	$(\mathbf{1000}, \mathbf{10,000})$	0.004	0.01	$\mathbf{\frac{1}{2}n}$	100
10	5000	0.004	$(\mathbf{0.002}, \mathbf{0.30})$	1000	100
11	5000	0.004	0.01	$(\mathbf{100}, \mathbf{10,000})$	100
12	5000	0.004	0.01	1000	$(\mathbf{50}, \mathbf{500})$

Table 3: Simulation settings for CP divide-and-conquer algorithm. n is the number of nodes in the network; p_{11} is the core-core edge probability; α_n is the proportion of nodes in the core; B is the number of sub-graphs sampled; and qn is the number of nodes per sub-graph. The parameters which vary are highlighted in **bold**.

4.2.2 Results

The results are in Figure 2. In setting 7, as p_{11} increases, all sub-sampling schemes have an increasing AUC. BFS, DN, RE, and RW all perform nearly identically with DFS also performing the same once $p_{11} \geq 0.005$. On the other hand, RN and RNN clearly yield worse results. Next, it is helpful to compare settings 8 and 9 together since they both look at increasing n. Similar to setting 7, DFS yields the best results with BFS, DN, RE, and RW yielding similar performance, while RN and RNN consistently yield the lowest AUC values. In addition, we notice that if B is fixed and n increases (setting 8), the AUC values start to decrease for larger n, most notably for RN and RNN. In setting 9, however, where B increases with n, all methods save RNN have generally monotonically increasing AUC with increasing n.

Moving to setting 10, the performance of DFS drastically decreases as the core size increases. All other methods have roughly constant AUC values. In setting 11, we see that increasing the number of sub-sampled nodes beyond one or two thousand yields minimal returns for DFS, BFS, DN, RE, and RW. RN, however, has monotonically improving AUC with increasing B while RNN also improves with larger B. These trends are similar in setting 12. Increasing the number of nodes in the sub-samples only seems to noticeably improve RN and RNN; the other methods remain relatively unchanged, especially once $qn > 50$.

4.2.3 Discussion

There are several takeaways from these simulation settings. First, it is clear that BFS, DN, RE, and RW are the best sub-sampling methods for CP identification. DFS also performs well but its behavior for increasing core size is somewhat concerning. On the other hand, RN and RNN display the worst performance. Since the mis-classification rate is lower when core nodes are sampled more frequently, the best methods will have a higher probability of sampling core nodes. Indeed, both RN and RNN sample core nodes less frequently than BFS, DN, RE, and RW. This also aligns with the theoretical results which showed that RE and DN should perform similarly, and that both of these methods should outperform RN. Continuing on a theoretical thread, we note that the theory in Section 3 assumes that the number of sub-graphs, B, is fixed. In Setting 8, however, we saw that this assumption

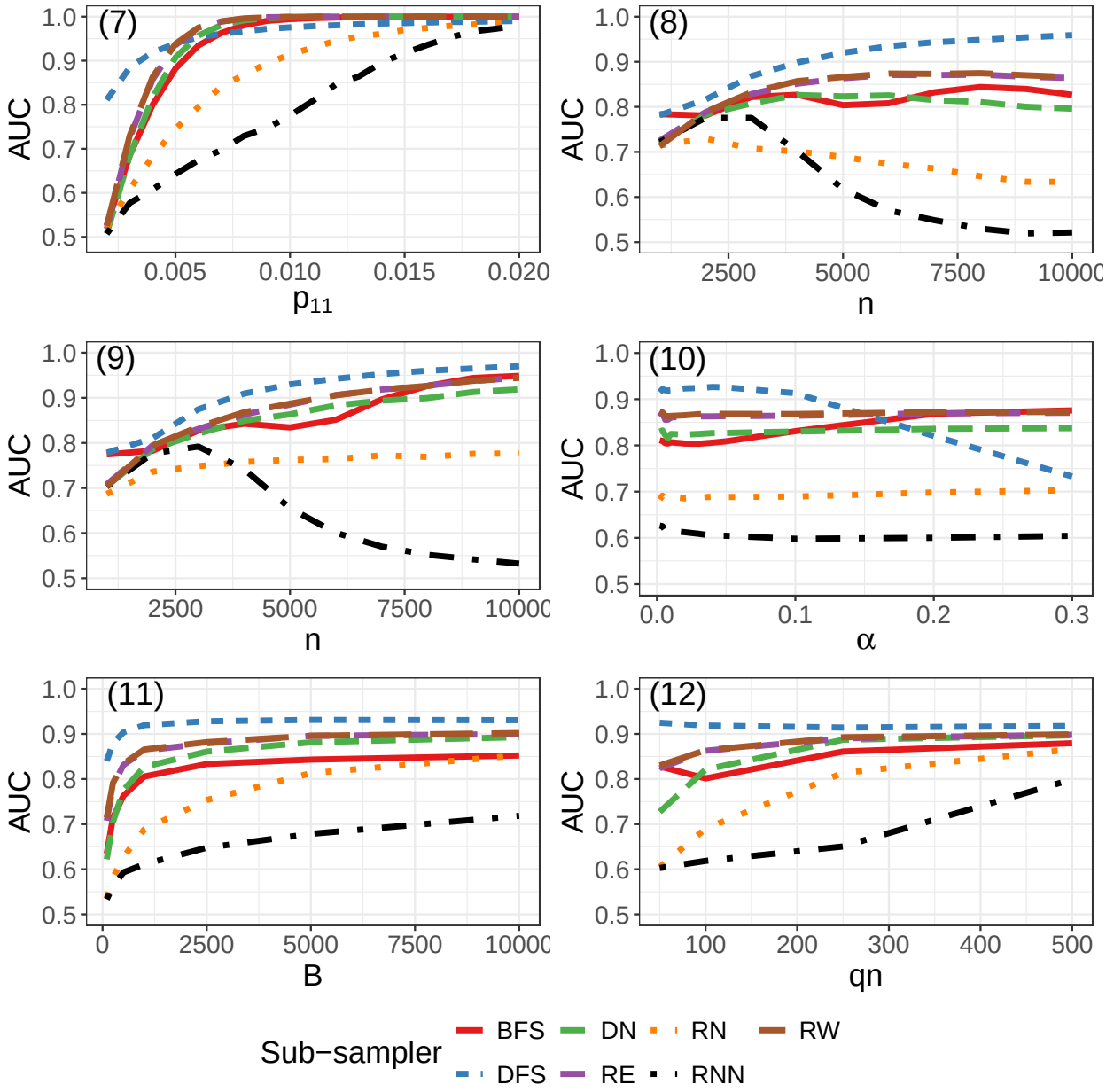

Figure 2: Core-periphery simulation results. The number in the upper-left corner corresponds to the simulation setting (7-12).

can lead to poor performance empirically. Indeed, settings 8 and 9 seem to indicate that B must be increased in larger networks, as there are more nodes to sample. Lastly, these results have implications on the choice of hyper-parameters. In settings 11 and 12, we found that increasing B and q only leads to marginal gains in performance, particularly for BFS, DN, RE, and RW. This means that these values can be set relatively small, increasing the computational speed. RN and RNN require larger B and q to match the other method's performance, further demonstrating their weakness for this task.

5 Real-data analysis

In this section, we implement both divide-and-conquer algorithms on various real-world networks.

5.1 Community detection

We apply the PACE algorithm to two different real-world networks. This first is *Political Blogs* (Adamic and Glance, 2005) ($n = 1490, m = 16,715$) where nodes represent blogs and edges arise if one blog references another. The blogs are divided between liberal and conservative, so we set $K = 2$, and keep only the largest connected component, yielding $n = 1222$. The second is *Facebook* (Leskovec and Mcauley, 2012) ($n = 4039, m = 88,234$) where nodes are users connected with an edge if they are friends. This is an ego network with ten egos so we set $K = 10$. We remove all edge weights and directions as well as self-loops for both networks.

We apply the PACE algorithm to each network and report the size of the largest community as well as the modularity of the returned node membership vector where a large value indicates a better community assignment. Since we have the ground truth labels (which we call the Oracle) for the *Political Blogs* network, we can also compute the ARI between the true and estimated labels. Finally, we set $q = 250/n$ for each network, and $B = 1000$ and 5000 for *Political Blogs* and *Facebook*, respectively.

The results are in Table 4. For *Political Blogs*, RN yields both the largest modularity value as well as most similar labels to the ground-truth. RNN does the next best, with RE also performing similarly. On the other hand, DFS yields substantially lower modularity and ARI values, likely due to its much greater imbalance in community labels. The results are similar for *Facebook*, where RN has the largest modularity and RNN provides the second-largest. In this example, both BFS and DFS provided imbalanced class labels, leading to lower modularity values.

5.2 Core-periphery detection

We now compare the sub-sampling routines using the CP divide-and-conquer algorithm on real-world networks: *Airport* (Csardi, 2013) is a network where nodes correspond to airports and edges to flights between airports ($n = 755$, $m = 4623$); and *Twitch* (Rozemberczki et al., 2019; Leskovec and Krevl, 2014) has nodes which are users and edges represent friendships

	Political Blogs			Facebook	
Method	Size	Q	ARI	Size	Q
Oracle	0.52	0.405	–	–	–
RN	0.53	0.425	0.810	0.46	0.743
DN	0.63	0.403	0.430	0.57	0.625
RE	0.66	0.414	0.495	0.64	0.550
BFS	0.62	0.388	0.463	0.86	0.041
DFS	0.90	0.075	0.061	0.99	0.003
RNN	0.63	0.419	0.584	0.66	0.694
RW	0.62	0.406	0.454	0.72	0.546

Table 4: Real data analysis results for PACE community structure divide and conquer algorithm.

($n = 168, 11$, $m = 6,797,557$). Edge weights and directions were again removed, along with self-loops.

For each network, we report the size of the core and corresponding BE metric value for the optimal core returned by the algorithm. In the simulation settings, we simply used the core proportions as a measure of a node's coreness, but for the real data, we want binary CP labels. To convert the core proportions to binary labels, we sort the nodes in descending order of their proportions. Then one by one, starting with the node with the largest proportion, we add this node to the core and compute the BE metric. We continue this process and select the core corresponding to the largest BE metric.

While the airport network is relatively small, this allows us to also find the CP labels using the vanilla BE metric algorithm without the divide-and-conquer step. We label this as "Full" in our results, and this allows us to compare both the core nodes and BE metric returned by the divide-and-conquer algorithm. Finally, we note the following values of q and B that we used for each network: *Airport*, $q = 100/n$ and $B = 5000$; and *Twitch*, $q = 500/n$ and $B = 25,000$.

In Table 5, we report the core size and BE metric for the detected cores, and in Figure 3, we compare the cores returned by the different algorithms. Specifically, for each sub-sampling pair, we compute the Jacaard coefficient (JC) between the core sets. The more similar the cores are, the larger the JC will be. For *Airport*, RE returns the most similar core as that of the full algorithm. All other methods have similar core sizes and BE metric values. Indeed, Figure 3 shows that each core is not only a comparable size, but the core nodes returned by each sub-sampling method have significant overlap. For *Twitch*, RE and RW yield the largest BE values, with RN, BFS and RNN also being similar. DN yields a vastly different core size and BE value, while DFS does not perform as well. For this network, RW and RE have the most similar cores.

6 Conclusion

Graph sub-sampling has long been recognized as an important task by network scientists and computer scientists, but it has yet to draw as much attention from statisticians. In this work,

	Airport		Twitch	
Method	k	BE	k	BE
Full	33	0.236	–	–
RN	29	0.236	275	0.077
DN	31	0.233	164	0.004
RE	31	0.233	170	0.079
BFS	35	0.234	88	0.076
DFS	27	0.225	155	0.066
RNN	30	0.234	91	0.076
RW	32	0.235	141	0.079

Table 5: Real data analysis results for core-periphery divide and conquer algorithm. k is the number of nodes assigned to the core and BE is the value of the Borgatti and Everett metric corresponding to the optimal labels.

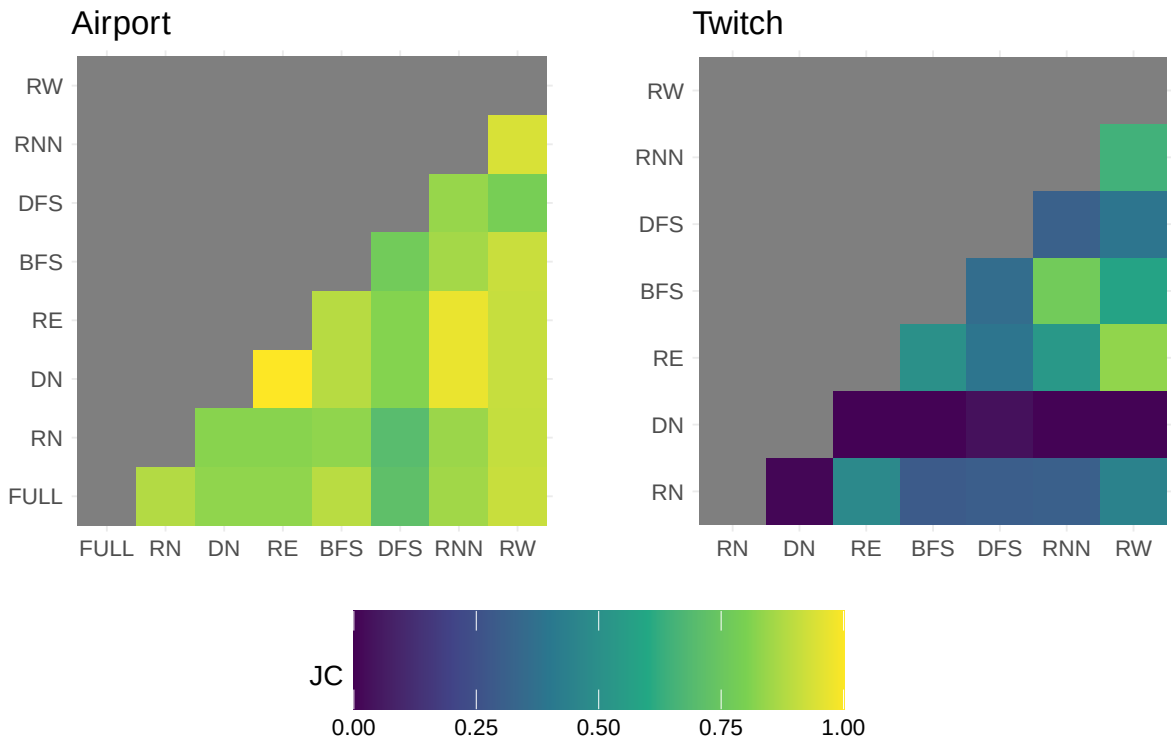

Figure 3: Comparison of cores returned using different sub-sampling algorithms. The color of the square corresponds to the Jacaard coefficient between the two core sets with lighter color meaning more similarity.

we offer an empirical and theoretical comparison of seven graph sub-sampling algorithms from a statistical angle by applying these methods to divide-and-conquer algorithms for two important meso-scale network features: community structure and core-periphery structure.

Based on our findings, we recommend random node sampling when trying to detect community structure. This method had some of the best performance in the simulated networks, as well as in the real-world networks. Moreover, random node sampling was studied theoretically in Mukherjee et al. (2021) and is the easiest sub-sampling method to implement. The reason for this good performance could be because random node sampling can easily sample across the network without getting "stuck" in any one part. Of course, random node sampling, as well as the other methods, will suffer if there are communities with only a few nodes. In this case, it is important to increase the size of the sub-graphs and/or the number of iterations. Randomly sampling nodes proportional to their degree as well as random node neighbor sampling also provided good results on the synthetic and real-world networks. Breadth-first and depth-first search algorithms, however, should be avoided for this task.

As for core-periphery identification, there was no clear winner, but samplers which selected core nodes with a higher probability consistently performed the best. Random edge sampling and random walk both yielded good results across the simulated and real-world data. On the other hand, random node sampling and random node neighbor performed quite poorly in these settings, aligning with our theoretical results. Finally, the real-data results show that many different cores can yield similar objective function values, implying the objective function surface is relatively "flat."

The contrasting performance of sub-sampling routines on the different identification task is revealing. Random node and random node neighbor sampling performed the best for the community detection problem, but were the two weakest methods for core-periphery identification. On the other hand, breadth-first and depth-first searches are good approaches for identifying core-periphery structure, but not for community structure. These results highlight the importance of tailoring the choice of sub-sampling algorithm to the particular problem. Indeed, our CP theory revealed the optimal feature for a sampler is a high probability of sampling core nodes. Deriving similar theory for future problems will help researchers choose the best methods, or even may even drive them to derive novel sub-sampling schemes for the specific problem.

References

Adamic, L. A. and Glance, N. (2005). The political blogosphere and the 2004 us election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, pages 36–43.

Ahmed, N. K., Neville, J., and Kompella, R. (2013). Network sampling: From static to streaming graphs. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(2):1–56.

Backstrom, L., Huttenlocher, D., Kleinberg, J., and Lan, X. (2006). Group formation in large social networks: membership, growth, and evolution. In *Proceedings of the 12th*

ACM SIGKDD international conference on Knowledge discovery and data mining, pages 44–54.

Borgatti, S. P. and Everett, M. G. (2000). Models of core/periphery structures. *Social Networks*, 21(4):375–395.

Cai, T. T. and Li, X. (2015). Robust and computationally feasible community detection in the presence of arbitrary outlier nodes. *Annals of Statistics*, 43(3):1027–1059.

Clauset, A., Newman, M. E. J., and Moore, C. (2004). Finding community structure in very large networks. *Physical Review E*, 70(066111).

Csardi, M. G. (2013). Package 'igraph'. *Last accessed*, 3(09):2013.

Girvan, M. and Newman, M. E. (2002). Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826.

Holland, P. W., Laskey, K. B., and Leinhardt, S. (1983). Stochastic block models: First steps. *Social Networks*, 5:109–137.

Jordan, M. I. (2012). Divide-and-conquer and statistical inference for big data. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 4–4.

Leskovec, J. and Faloutsos, C. (2006). Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636.

Leskovec, J. and Krevl, A. (2014). SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`.

Leskovec, J. and Mcauley, J. (2012). Learning to discover social circles in ego networks. *Advances in neural information processing systems*, 25.

Mukherjee, S. S., Sarkar, P., and Bickel, P. J. (2021). Two provably consistent divide-and-conquer clustering algorithms for large networks. *Proceedings of the National Academy of Sciences*, 118(44).

Newman, M. E. (2006). Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582.

Ng, A. Y., Jordan, M. I., and Weiss, Y. (2002). On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856.

Rozemberczki, B., Allen, C., and Sarkar, R. (2019). Multi-scale attributed node embedding.

Rozemberczki, B., Kiss, O., and Sarkar, R. (2020). Little Ball of Fur: A Python Library for Graph Sampling. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*, page 3133–3140. ACM.

Telesford, Q. K., Simpson, S. L., Burdette, J. H., Hayasaka, S., and Laurienti, P. J. (2011). The brain as a complex system: using network science as a tool for understanding the brain. *Brain connectivity*, 1(4):295–308.

Yanchenko, E. (2022). A divide-and-conquer algorithm for core-periphery identification in large networks. *Stat*, 11(1):e475.

Yanchenko, E. and Sengupta, S. (2023). Core-periphery structure in networks: A statistical exposition. *Statistic Surveys*, 17:42–74.

Zhang, S., Song, R., Lu, W., and Zhu, J. (2022). Distributed community detection in large networks. *arXiv preprint arXiv:2203.06509*.

Supplemental Materials

Theoretical proofs

Proof of Theorem 3.2: Let $\boldsymbol{c}^*$ be the true CP labels. Let $\hat{\boldsymbol{c}}$ be the estimated labels from divide-and-conquer procedure where

$$\hat{c}_i = \frac{1}{B}\sum_{b=1}^{B} \hat{c}_i^{(b)}$$

where $\hat{\boldsymbol{c}}^{(b)}$ is the CP labels returned from the bth sub-sample. Note that $\hat{c}_i^{(b)}$ is 1 if the node was sampled and assigned to the core and 0 otherwise. Now, notice that

$$\hat{c}_i - c_i^* = \left(\frac{1}{B}\sum_{b=1}^{B} \hat{c}_i^{(b)}\right) - c_i^* = \frac{1}{B}\sum_{b=1}^{B}(\hat{c}_i^{(b)} - c_i^*).$$

Thus,

$$\delta(\hat{\boldsymbol{c}}, \boldsymbol{c}^*) = \frac{1}{n}\sum_{i=1}^{n}\left\{\frac{1}{B}\sum_{b=1}^{B}(\hat{c}_i^{(b)} - c_i^*)\right\}^2$$

By Cauchy-Schwarz, we then have

$$\delta(\hat{\boldsymbol{c}}, \boldsymbol{c}^*) \leq \frac{1}{nB}\sum_{i=1}^{n}\sum_{b=1}^{B}\{\hat{c}_i^{(b)} - c_i^*\}^2 = \frac{1}{nB}\sum_{b=1}^{B}\sum_{i=1}^{n}\{\hat{c}_i^{(b)} - c_i^*\}^2$$

Now,

$$\frac{1}{n}\sum_{i=1}^{n}\{\hat{c}_i^{(b)} - c_i^*\}^2 = \frac{1}{n}\sum_{i=1}^{n}\mathbb{I}(\hat{c}_i^{(b)} \neq c_i^*)$$

is the mis-classification error of the algorithm on a single sub-graph. We can get an error here from one of two ways. First, if our sub-graph contains node i but incorrectly classify it. Second, if we don't sample node i and it's in the core. Thus,

$$\frac{1}{n}\sum_{i=1}^{n}\{\hat{c}_i^{(b)} - c_i^*\}^2 = \frac{1}{n}||\hat{\boldsymbol{c}}^{(b)} - \boldsymbol{c}^{(b)*}||_2^2 + \frac{1}{n}\sum_{i=1}^{n} y_i^{(b)}$$

where $\boldsymbol{c}^{(b)*}$ is the true core labels restricted to sub-graph b and $y_i^{(b)} = 1$ if node i is in the core and was not sampled in b, and 0 otherwise. Thus,

$$\delta(\hat{\boldsymbol{c}}, \boldsymbol{c}^*) \leq \frac{1}{nB}\sum_{b=1}^{B}\left(||\hat{\boldsymbol{c}}^{(b)} - \boldsymbol{c}^{(b)*}||_2^2 + \sum_{i=1}^{n} y_i^{(b)}\right)$$

Since each term in the sum is independent and identically distributed, we find

$$\mathbb{E}\delta(\hat{\boldsymbol{c}}, \boldsymbol{c}^*) \leq \frac{1}{n}(\mathbb{E}||\hat{\boldsymbol{c}}^{(S)} - \boldsymbol{c}^{(S)*}||_2^2 - \mathbb{E}\sum_{i=1}^{n} y_i^{(S)}) \tag{8}$$

where S is a randomly chosen sub-graph from our sub-sampling routine. This gives us the desired result.

Proof of Corollary 3.2: Here, we present the specific error for each of the sub-sampling routines. First, let $Y^{(b)}$ be the number of core nodes drawn in sub-sample b, i.e.,

$$Y^{(b)} = k - \sum_{i=1}^{n} y_i^{(b)}.$$

Without loss of generality, assume that the first k nodes in the network are the core nodes.

Random Node: For the random node sampling scheme, each node is equally likely to be selected. Thus, $Y^{(b)}$ has a hyper-geometric distribution with a total population of size n,"success" states in the population k, and draws qn, so

$$\mathbb{E}Y^{(b)} = \frac{qnk}{n} = qk.$$

Another way to see this is that the probability of a core node being drawn is k/n and we draw qn nodes so we get the qk. Thus, the expected number of core nodes which are *not* sampled is simply

$$k - qk = (1-q)k$$

Therefore,

$$\mathbb{E}\ \frac{1}{n}\sum_{i=1}^{n} y_i^{(b)} = (1-q)\frac{k}{n}. \ \square$$

For the following derivations, we note that $m = \sum_{i<j} A_{ij}$ is highly concentrated around its mean, $\sum_{i<j} P_{ij}$, and the error bound for m is much smaller than that of the other (appropriately scaled) random variables. Therefore, in what follows, we will ignore the randomness of m and consider it to be approximately equal to its expectation, i.e., if the data comes from a CP-SBM, then

$$2m \approx \mathbb{E}2m = k(k-1)p_{11} + 2k(n-k)p_{12} + (n-k)(n-k-1)p_{22}.$$

We also assume that the network is large enough such that the probability of sampling the same node twice is negligible.

Degree Node: For Random Node, each node had a $1/n$ probability of being selected. Now, a nodes' the probability of being drawn is proportional to its degree. We can model this as drawing from an urn with $2m$ balls where there are d_i balls labeled for node i, where d_i is the degree of node i. Then $Y^{(b)}$ has a hyper-geometric distribution where now there are $2m$ population elements, $\sum_{i=1}^{k} d_i$ success states, and qn draws. Thus,

$$\mathbb{E}Y^{(b)} = \mathbb{E}\mathbb{E}(Y^{(b)}|d_i) = \mathbb{E}\frac{qn\sum_{i=1}^{k} d_i}{2m} = \frac{qnk\{(k-1)p_{11} + (n-k)p_{12}\}}{k(k-1)p_{11} + 2k(n-k)p_{12} + (n-k)(n-k-1)p_{22}}$$

since

$$\mathbb{E}d_i = (k-1)p_{11} + (n-k)p_{12}$$

if node i is in the core. Thus,

$$\mathbb{E}\,\frac{1}{n}\sum_{i=1}^{n} y_i^{(b)} = \left[1 - \frac{qn\{(k-1)p_{11} + (n-k)p_{12}\}}{k(k-1)p_{11} + 2k(n-k)p_{12} + (n-k)(n-k-1)p_{22}}\right]\frac{k}{n}$$

For large n, this simplifies to

$$\left(1 - q\frac{p_{12}}{p_{22}}\right)\alpha_n. \;\square$$

Random Edge: We can again imagine an urn filled with balls where now the balls are labeled with edges instead of nodes. For each ball that we draw, we sample the two nodes which connect this edge, so we only make $qn/2$ draws. There are two types of edges that we are interested in: a core-core edge where both nodes are in the core, and a core-periphery edge, where one node is from the core. If $M_{cc}^{(b)}$ and $M_{cp}^{(b)}$ are the number of core-core and core-periphery edges drawn in sub-sample b, respectively, then

$$\mathbb{E}Y^{(b)} = 2\mathbb{E}M_{cc}^{(b)} + \mathbb{E}M_{cp}^{(b)}.$$

Let m_{cc} and m_{cp} be the observed number of core-core and core-periphery edges in the network. Then $M_{cc}^{(b)}$ has a hyper-geometric distribution with population size m, m_{cc} success states and $qn/2$ draws. Similarly, $M_{cp}^{(b)}$ has a hyper-geometric distribution with population size m, m_{cp} success states and $qn/2$ draws. Thus,

$$\begin{aligned}\mathbb{E}Y^{(b)} &= 2\mathbb{E}\mathbb{E}(M_{cc}^{(b)}|m_{cc}) + \mathbb{E}\mathbb{E}(M_{cp}^{(b)}|m_{cp}) \\ &= 2\mathbb{E}\frac{m_{cc}qn}{2m} + \mathbb{E}\frac{m_{cp}qn}{2m} = \frac{k(k-1)p_{11}qn}{2m} + \frac{k(n-k)p_{12}qn}{2m}\end{aligned}$$

Therefore,

$$\mathbb{E}\,\frac{1}{n}\sum_{i=1}^{n} y_i^{(b)} = \left[1 - \frac{qn\{(k-1)p_{11} + (n-k)p_{12}\}}{k(k-1)p_{11} + 2k(n-k)p_{12} + (n-k)(n-k-1)p_{22}}\right]\frac{k}{n}$$

which is the same result as in Degree Node. Again, for large n, this simplifies to

$$\left(1 - q\frac{p_{12}}{p_{22}}\right)\alpha_n. \;\square$$

Random Node Neighbor For Random Node Neighbor, we randomly sample a node and then include that node as well as all of its neighbors in the sub-graph. First, let's find the expected number of core nodes per draw. The expected number of core nodes drawn for a single iteration of this sampler depends on whether the initial node as in the core or periphery. Thus, assuming an SBM,

$$\begin{aligned}&\mathbb{E}(\text{core nodes}|\text{core node})\mathbb{P}(\text{core node}) + \mathbb{E}(\text{core nodes}|\text{periphery node})\mathbb{P}(\text{periphery node}) \\ &= \{1 + (k-1)p_{11}\}\frac{k}{n} + kp_{12}\frac{n-k}{n} \\ &= C_n\end{aligned}$$

So for each draw, we expect to include C_n core nodes in the sub-graph. But how many draws do we make? For each draw, the expected number of nodes sampled again depends on the first draw.

$$\begin{aligned}
&\mathbb{E}(\text{nodes}|\text{core node})\mathbb{P}(\text{core node}) + \mathbb{E}(\text{nodes}|\text{periphery node})\mathbb{P}(\text{periphery node}) \\
&= 1 + \{(k-1)p_{11} + (n-k)p_{12}\}\frac{k}{n} + \{kp_{12} + (n-k-1)p_{22}\}\frac{n-k}{n} \\
&= T_n
\end{aligned}$$

where the first 1 is from the node we initially drew. Thus, if we want to sample qn nodes, then we expect to make qn/T_n draws. And for each draw, we expect to sample C_n core nodes. Thus, the expected number of core nodes in our sub-graph is

$$qn\frac{C_n}{T_n} = qn\frac{\{1 + (k-1)p_{11}\}\frac{k}{n} + kp_{12}\frac{n-k}{n}}{1 + \{(k-1)p_{11} + (n-k)p_{12}\}\frac{k}{n} + \{kp_{12} + (n-k-1)p_{22}\}\frac{n-k}{n}}$$

Therefore,

$$\mathbb{E}\ \frac{1}{n}\sum_{i=1}^{n} y_i^{(b)} = \left[1 - qn\frac{\{1 + (k-1)p_{11}\}\frac{1}{n} + p_{12}\frac{n-k}{n}}{1 + \{(k-1)p_{11} + (n-k)p_{12}\}\frac{k}{n} + \{kp_{12} + (n-k-1)p_{22}\}\frac{n-k}{n}}\right]\frac{k}{n}.$$

For large n, this simplifies to

$$\mathbb{E}\ \frac{1}{n}\sum_{i=1}^{n} y_i^{(b)} = \left(1 - q\frac{p_{12}}{\alpha_n p_{12} + p_{22}}\right)\alpha_n. \ \square$$

Random Walk While the calculations are much more involved, we can also find the expected number of core nodes that aren't sampled for Random Walk. Let E^l be the expected number of core nodes traversed on a random walk through a CP-SBM. Then, depending on whether the first node selected is a core or periphery node we have

$$E_l = \frac{k}{n}(1 + x_{l-1}) + \frac{n-k}{n}y_{l-1}$$

where x_l and y_l are the expected number of core nodes traversed if the starting node is in the core and periphery, respectively. We can set up a coupled recurrence relationship to solve for these terms. Again, the key observation is that the expectation just depends on whether the next step is to a core or peripheral node:

$$\begin{aligned}
x_l &= \underbrace{\frac{(k-1)p_{11}}{(k-1)p_{11} + (n-k)p_{12}}}_{\alpha}(1 + x_{l-1}) + \underbrace{\frac{(n-k)p_{12}}{(k-1)p_{11} + (n-k)p_{12}}}_{\beta} y_{l-1} \\
y_l &= \underbrace{\frac{kp_{12}}{kp_{12} + (n-k-1)p_{22}}}_{\gamma}(1 + x_{l-1}) + \underbrace{\frac{(n-k-1)p_{22}}{kp_{12} + (n-k-1)p_{22}}}_{\delta} y_{l-1}
\end{aligned}$$

with initial conditions

$$x_1 = 1 \text{ and } y_1 = 0.$$

So we can simplify this as

$$\begin{aligned} x_l &= \alpha x_{l-1} + \beta y_{l-1} + \alpha \\ y_l &= \gamma x_{l-1} + \delta y_{l-1} + \gamma \end{aligned}$$

This can be written in matrix notation as

$$\mathbf{v}_l = \mathbf{A}\mathbf{v}_{l-1} + \mathbf{c}$$

where $\mathbf{v}_l = (x_l, y_l)^T$, $\mathbf{c} = (\alpha, \gamma)^T$ and

$$\mathbf{A} = \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}$$

where $\mathbf{v}_1 = (1, 0)^T$. This has a general solution of

$$\mathbf{v}_l = \mathbf{A}^l \mathbf{v}_1 + \sum_{i=1}^{l-1} \mathbf{A}^i \mathbf{c}.$$

Then we just need to plug these solutions into our equation for E_l, noting that we have $l = qn$ for the length of our random walk.