DEANONYMIZING BITCOIN TRANSACTIONS
AN INVESTIGATIVE STUDY ON LARGE-SCALE GRAPH CLUSTERING

A senior thesis
submitted to the Department of Mathematics
in partial fulfillment of the requirements for
the degree of Bachelor of Arts

By

Yash Patel

Under the supervision of

S. Matthew Weinberg

Princeton University
Princeton, New Jersey

May 7, 2018

Acknowledgements

I would like to extend a profound thank you to the following people involved directly or indirectly in the writing of my thesis:

My thesis advisor for having spent countless hours explaining and re-explaining the relevant concepts for my research work, for helping guide me through the foreign land of research material for my junior IW and senior thesis alike, and for always being a motivated and enthusiastic voice to bounce ideas off of for bringing the project forward.

My parents for having constantly pushed me forward and supported me on my journey through my academic challenges at Princeton and onward into the future.

My sister for having been the best friend a brother could wish for, always reminding me to see the bright side of any situation as it comes.

My friends from Quad, Naacho, and the Princeton community at large, who always managed to turn every moment, from the most light to the most stressful, into a greatly memorable experience.

Contents

List of Figures

List of Tables

Abstract

Bitcoin has emerged from the fringes of technology to the mainstream recently. With speculation rampant, it has become more and more the subject of harsh criticism in ascertaining its use case. Unfortunately, much of Bitcoin's present use case is for transactions in online black markets. Towards that end, various studies have sought to partially deanonymize Bitcoin transactions, identifying wallets associated with major players in the space to help forensic analysis taint wallets involved with criminal activity. Relevant past studies, however, have rigidly enforced manually constructed heuristics to perform such deanonymization, paralleling an extensive union-find algorithm. We wish to extend this work by introducing many more heuristics than were previously considered by constructing a separate "heuristics graph" layered atop the transactions graph and performing a graph clustering on this heuristics graph. Towards that end, we explored the performance of various clustering algorithms on the SBM (stochastic block model) as a prototype of the heuristics graph and additionally tested graph preprocessing algorithms, specifically sparsification and coarsening to determine the extent they could speed up computation while retaining reasonable accuracies. We found hierarchical spectral clustering and METIS to have the best performance by the standard purity, NMI, and F-score clustering accuracy metrics. We also found sparsification and coarsening to result in little reduction in time with the former severely detracting from accuracies and the latter less so, suggesting the latter holds potential given implementation optimization in future studies. METIS was subsequently employed to cluster a subset of the full graph due to major time concerns with hierarchical spectral clustering. Several wallet clusters were identified as a result, though the accuracy of this could not be determined due to the limited ground truth available. Future extensions of this work should seek to refine the hierarchical spectral clustering algorithm for its time deficiencies and extend the ground truth available.

1 Introduction

Bitcoin (BTC), previously relegated as a technology for only the most technologically competent of computer scientists, has emerged into the mainstream in an enormous way. Ever since its explosive rise over the past year, technologists and investors both have sought to understand and exploit its technical possibilities. As illustrated in Figure 1, speculation on Bitcoin, both directly and in the form of associated ETF derivative markets [37], has become an undeniable sector of the modern technology world. While current speculations are high, many investors and technologists alike struggle to understand the fundamental use case of Bitcoin and its underlying technology: blockchain. The speculative bubble is largely the product of vacuous interest pooling into an asset of misunderstood potential, paralleling the financial situation surrounding the dot-com bubble.

Figure 1: The price of one Bitcoin, as tracked by Bloomberg. While technologists saw its promise many years prior, Bitcoin's rise in popularity in the general public has led to exponential speculative interest recently [21].

Prior to further discussing Bitcoin's use cases as a whole, it serves a purpose to understand the underlying technology first. Briefly, Bitcoin is a cryptocurrency, by which we mean a "digital unit of exchange that is not backed by a government-issued legal tender" [43]. The blockchain technology underpinning Bitcoin and other cryptocurrencies, as illustrated in Figure 2, is "a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes" [29]. Described plainly, blockchain seeks to solve a technical problem, namely that of achieving consensus in a distributed system, to arbitrate exchange of a common asset without the need of a trusted central authority. Since the desired end goal is transactions without a trusted third-party, it must be the case that any peer in this network can verify transactions as valid. This circumstance lends itself

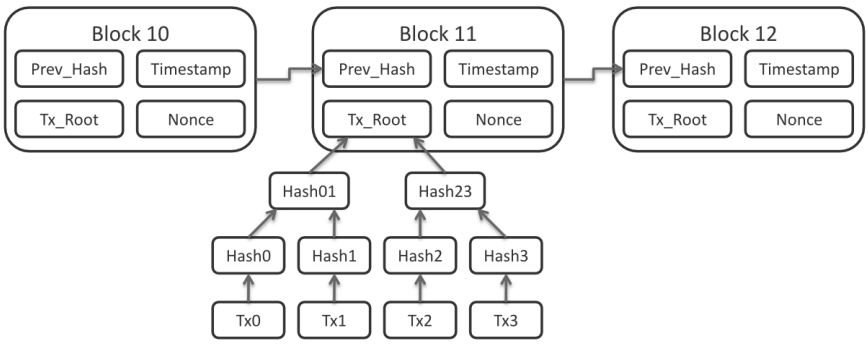

Figure 2: Blockchain, the underlying technology behind Bitcoin and all other cryptocurrencies, is simply a data structure consisting of an extended linked list of hash pointers with metadata. These pieces of metadata, in addition to carrying the actual BTC transaction information, more importantly serve data that can be used for efficient verification of double-spending in a distributed system [44].

naturally to the well-known concept of public-key encryption. It, therefore, should come as little surprise that this is how wallets are implemented in Bitcoin and other cryptocurrencies.

Specifically, as with any public-key encryption system, Bitcoin exchange revolves around participating members having two keys, one public and one private. For the specific use case of Bitcoin, the private key is used to sign any transactions messages being made to another wallet. In turn, any other account can verify it indeed originated from the claimed party using the signer's public key, as demonstrated in further detail in Figure 3.

Incidentally, these public keys serve as what people colloquially refer to as their "wallets." Specifically, "Bitcoin... is designed with pseudonymous identities. Account numbers are public keys of a specific asymmetric encryption system" [27]. This notion of Bitcoin being "pseudonymous" refers to this presence of an abstraction layer between real life identities and those used within the Bitcoin universe, despite the wallet IDs' activities being publicly visible. In a fully anonymous protocol, there would be no such information revealed in public channels.

Returning to the discussion of interest, Bitcoin, when viewed from this perspective, seemingly has no properties making it inherently more conducive to privacy apart from its abstraction of people to their public wallet addresses. This abstraction, once again, is seemingly comparable to that imposed by the current banking system, which abstracts a person's account number from his real life identity. The key differences, however, are twofold. In the first place, the abstraction layer instituted by banks is leaky. That is to say, while people generally do not know one another's account numbers, authorities within banks have this association, for they always require personal details be

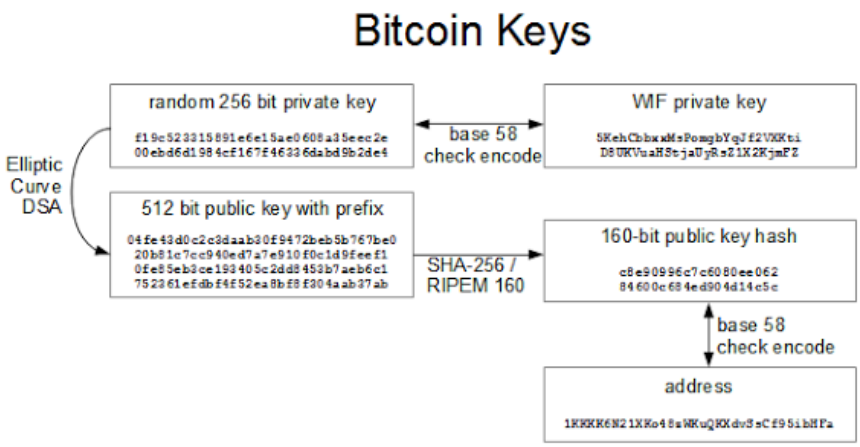

Figure 3: While public-key encryption is the basic idea behind the Bitcoin protocol, there are many additional details involved in making it a practical system to be used. In particular, additional hashing functions are used in an effort to make the underlying keys more manageable to deal with and store as an end user, while meeting sufficient security demands. These details, while interesting in their own right, however, are not of key important for this particular study [39].

provided in signing up for an account, per the famous "know your customer" adage. Bitcoin has no similar gatekeeper with identifying information by virtue it being a trustless protocol, meaning Bitcoin wallets can be procured without the need to link them to personal details. The second distinguishing feature is the ease with which such wallets can be created. In line with the lack of a central authority, there is no barrier in place preventing a single person or entity from creating many Bitcoin wallets. This follows as, in the process of making a wallet, there is *no* way to distinguish whether the creator is someone who has made a wallet or not due to the lack of associated identifying attributes. Thus, the properties of privacy that have come to be associated with Bitcoin have been born out of its lack of a central authority holding identifying links.

Returning to the discussion of the use cases of Bitcoin, many have been born out of this seeming impossibility in linking real-life identity with associated transaction wallets. It, therefore, seems natural that its use cases are generally of some illicit nature, as, if they were not, there would be no benefit of using Bitcoin over a traditional bank, as the former does not provide the typical fraud protections that are instituted by the latter. One government report discussed cryptocurrencies in the following light:

"[Cryptocurrencies] present particularly difficult law enforcement challenges because of their ability to transcend national borders in the fraction of a second, unique jurisdictional issues and anonymity due to encryption. Due primarily to their anonymity, virtual currencies have been linked

to numerous types of crimes, including facilitating marketplaces for: assassins, attacks on businesses, the exploitation of children (including pornography), corporate espionage, counterfeit currencies, drugs, fake IDs and passports, high yield investment schemes (Ponzi schemes and other financial frauds), sexual exploitation, stolen credit cards and credit card numbers, and weapons" [43].

To point to one particular example, the Silk Road, an "online market...[that] specialized in 'black market' goods, such as pornography, weapons or narcotics...corresponded to about 4.5% of all transactions occurring in exchange" during the early days of Bitcoin [8]. While Bitcoin's integration into the mainstream has greatly exploded since these statistics were measured, the fact remains that a significant portion of Bitcoin's trading activity lies in illicit markets. It, therefore, becomes a question of interest whether such illicit activity can be tracked and retraced. By this, we mean to say, whether we can produce a map that links wallet addresses to real-life identities. What is meant by "real-life identities" is somewhat ill-defined, as to what level of granularity we wish to resolve. For example, labelling a cluster as "John Smith" is just as valid as labelling it "Coinbase," despite the former being far more specific.

Briefly, Coinbase is a platform employed by many BTC users that performs two primary purposes: serving as a platform of exchange for cryptocurrencies and providing a simple way to set up wallets. These wallets, however, are not "owned" by the end user in the traditional sense; the private key is owned by Coinbase. Internally, they simply have a map linking a particular private key to the user's account, meaning any activity desired by the user on Coinbase is simply relayed to the Coinbase API, which serves as a "middleman" to the end Bitcoin application layer. The main takeaway is that Coinbase serves as a central manager of the accounts of many end users, making it a clear target for deanonymization, as we further expound upon in Section 2.

Thus, the main objective of the study herein was to deanonymize Bitcoin transactions to as great an extent as possible. **By deanonymize here, we are specifically referring to being able to cluster wallets together that are associated with the same real-world entity, from which tainting of any one of the associated addresses reveals the entire cluster identity.** The outline of this paper is as follows:

- **Section 2:** Covers the major technical topics of interest that are to be used in the remainder of the paper. Most of the findings presented through this section entail a literature review of well-known works in the respective fields, although occasional novel material is presented when appropriate for sake of organization. We further note that a majority of figures presented

through this section were catered from previous research endeavors.

- **Section 3:** Walks through the structure of the experiments that were written and conducted. The code is freely available at: `https://github.com/yashpatel5400/anonychain`. We also provide the specifics of the expected structure of results from each experiment.
- **Section 4:** Presents the results of the experiments laid out in the previous section, all in the form of relevant visualizations.
- **Section 5:** Discusses the main takeaways from the graphical results as they relate to the research problem. Specifically, these discussions pointedly answer the extent to which various clustering techniques can be used for deanonymizing Bitcoin transactions.
- **Section 6:** Provides follow-up research ideas that could be pursued should readers feel inspired and interested in furthering the research conducted herein.

2 Background

To follow is the necessary technical background for the paper, of which a majority concentrates on material from graph theory. We begin by presenting the current state-of-the-art in this deanonymization space and subsequently discuss all relevant technical concepts as they relate to this study. Once again, the objective of this investigation was to cluster Bitcoin wallets, such that all the wallets clustered together correspond to the same entity. Extending from here to the full deanonymization is simply a matter of expanding the ground truth to associate specific wallets to real-world entities, upon which we elaborate in the sections that follow.

2.1 Deanonymization Techniques

We now present the main research works that entail the current state of the art in the field of Bitcoin deanonymization. The main goals for this sections are to familiarize the reader with:

- The two main works of study, "Fistful of Bitcoin" and "Automatic Bitcoin Address Clustering," and how they both rely on the same principle of carefully catering manual heuristics to determine whether two addresses correspond to the same real world entity.
- How these investigations **both** rely on a rudimentary union-find algorithm run on the original BTC transactions graph to perform clustering, which results in a proliferation of error, specifically due to information needing to be cut from the manually catered heuristics to avoid false-positives from collapsing the overall graph.
- How the study herein seeks to alter the means by which such deanonymization has been hitherto sought, specifically by catering all the heuristics on a *separate* graph and employing graph clustering algorithms on this separate construction.

Currently, two main studies have come to define the space of deanonymization, whose results and limitations we discuss herein. The objective of both was exactly that we have for this study, presented as "our goal is not to generally deanonymize all Bitcoin users - as the abstract protocol design itself dictates that this should be impossible - but rather to identify certain idioms of use present in concrete Bitcoin network implementations that erode the anonymity of the users who engage in them" [26]. In other words, there is no way to go from the raw BTC address information to a real-world identity. However, as with any practical system, there are often leaks in information

due to sheer negligence by the end user. This parallels how many of the supposed "hacks" of email clients have little to do with the encryption of their internal systems but rather with the laziness of end users in procuring difficult-to-guess passwords.

In a similar fashion, end users often reveal information intended to be kept private in public channels. For example, people may post their wallet addresses on online forums, from which a link between the address and a real-world entity can be established. By exploiting such side-channel leaked information, we can gather a base set of truth regarding who owns a particular wallet, i.e. catering a list saying that address 1Mz7153HMuxXTuR2R1t78mGSdzaAtNbBWX is owned by John Doe and so on. This list, however, will be a very tiny subset of the addresses on the BTC network. It was, therefore, our ultimate objective to expand from this tiny seed of base information to determine the identities of as many other wallets as possible. While seemingly of little use, the purpose of this becomes clear in remembering that many people own multiple BTC wallets. In other words, John Doe, from the previous example, may own three other wallets other than 1Mz7153HMuxXTuR2R1t78mGSdzaAtNbBWX. Having tainted his privacy with this public information leak, it was ultimately our objective to associate these other three wallet IDs with his name.

Of course, gathering a comprehensive ground truth is critical for achieving this final goal of deanonymizing with names. **Towards that end, however, we consider the problem of clustering the wallets owned by a single entity as effectively equivalent to the final goal.** Understanding this point is of *utmost* importance, as this is what the majority of the remainder of the investigation revolves around. Specifically, the idealized end result of this investigation was a list of sets with each set corresponding to a *single* real-world entity, namely something of the form `set(1,3,4,6),set(2,8),set(5,7,9,10).` Given such a clustering, the final "names" for each cluster could easily be established when any one of the wallets gets a labelling. That is to say, if for the example result given it was found that vertex 3 corresponded to "Jill Doe," all of `set(1,3,4,6)` would now similarly be known to be owned by "Jill Doe."

Thus, when any one of these wallets has a public side-channel leak in the future, this information can be retroactively applied onto the clustering, from which the identities of a series of other wallets too is revealed. Thus, the act of grouping wallets as being owned by a single entity was the goal of this study, for which we use the procured ground truth as a method of verification. The specifics of this procedure are elaborated in Section 3.

An example as told by the authors of "A Fistful of Bitcoin" is as follows: "This clustering allows us to amplify the results of our re-identification attack: if we labeled one public key as belonging to Mt. Gox, we can now transitively taint the entire cluster containing this public key as belonging to Mt. Gox as well" [26]. For context, Mt. Gox served a very similar role to that which Coinbase currently fills. It should therefore be clear why they "consider de-anonymization of Bitcoin addresses as a clustering problem...Bitcoin address clustering is fairly different from classical clustering problems as there is no direct information about the objects' (addresses) such as coordinates or distances. The other peculiarity of the problem is the vastness of the Bitcoin blockchain, which requires designing computationally efficient algorithms for its clustering" [14]. In other words, there are two problems at hand that require solving. The first is of translating Bitcoin deanonymization into a natural clustering problem, and the second is solving this posed problem in a reasonable time frame given the scale of the data at hand.

We begin by discussing the first issue at hand, specifically the approach taken in "A Fistful of Bitcoin," which revolves around a simple union-find variant. We explain their approach and subsequently analyze its pitfalls and pose alternatives. While it may seem natural to simply cluster the transaction (tx) history, seeing as there is a clear canonical mapping from this transactions history to a weighted graph, two accounts having many transactions between them has no bearing on them being the same entity. After all, if account A has many transactions with account B, there is no reason to believe they correspond to the same *or* different people.

Prior to discussing their union-find strategy, we introduce the notion of a change address. "Change addresses are the mechanism used to give back to the input user in a transaction, as Bitcoins can be divided only by being spent" [26]. In other words, the Bitcoin protocol prevents an account that owns 10 BTC from only sending five of its BTC to another account. Instead, the account must spend the entirety of the 10 BTC, with five going to the desired recipient and the other five to a "change address" owned by the original owner though this change address may simply be the original wallet. Having said that, we can discuss the two heuristics used in "A Fistful of Bitcoin" for linking addresses.

Fact 2.1. *"If two (or more) addresses are inputs to the same transaction, they are controlled by the same user; i.e., for any transaction t, all $pk \in inputs(t)$ are controlled by the same user" [26].*

The main idea behind this is that all the inputs of a given transaction must sign off the transaction with their private key. We assume that end users are not sharing their private keys, for revealing

such information essentially gives the recipient full authority of the associated account, in terms of being able to send any BTC held in the account. Thus, as *all* input accounts must sign off the transaction, all the private keys must be present simultaneously to validate the transaction, which can only happen if a single real-world entity knows the private keys of *all* the associated accounts. This heuristic is likely the most trustworthy of the ones considered, meaning its percent of false positives is extremely low.

While this is the case, this heuristic is not remotely comprehensive. That is to say, there are *many* more wallets that are owned by the same entity as compared to those that appear together as inputs in a single transaction. Thus, it was of interest to present other heuristics that may have a lower robustness but still contain some information regarding the likelihood of wallets corresponding to the same real-world entity. The second heuristic presented from the paper was as follows:

Theorem 2.1. *"The one-time change address is controlled by the same user as the input addresses; i.e., for any transaction t, the controller of inputs(t) also controls the one-time change address $pkoutputs(t)$ (if such an address exists)" [26].*

In other words, we expect that change addresses, when identifiable as such, are controlled by the originator of the transaction. To identify change addresses, the following intuition was followed: "Working off the assumption that a change address has only one input (again, as it is potentially unknown to its owner and is not re-used by the client), we first looked at the outputs of every transaction. If only one of the outputs met this pattern, then we identified that output as the change address. If, however, multiple outputs had only one input and thus the change address was ambiguous, we did not label any change address for that transaction" [26].

Thus, "A Fistful of Bitcoins" performed union-find in accordance to these two heuristics. That is, for any pair of vertices that satisfied either of these two heuristics, the pair was condensed into a single cluster. By directly applying the two heuristics on the raw transaction graph, they produced Figure 4. The paper, however, faced many issues related to directly clustering the graph as defined by these two heuristics. The primary of which was that "falsely linking even a small number of change addresses might collapse the entire graph into large 'super-clusters' that are not actually controlled by a single user" [26]. With that in mind, we considered algorithms that tended towards clustering in favor for false negatives as compared to false positives with the idea that the latter would result in clustering with little sematic value. In fact, the original iteration of their clustering "ended up with a giant super-cluster containing the public keys of Mt. Gox, Instawallet, BitPay,

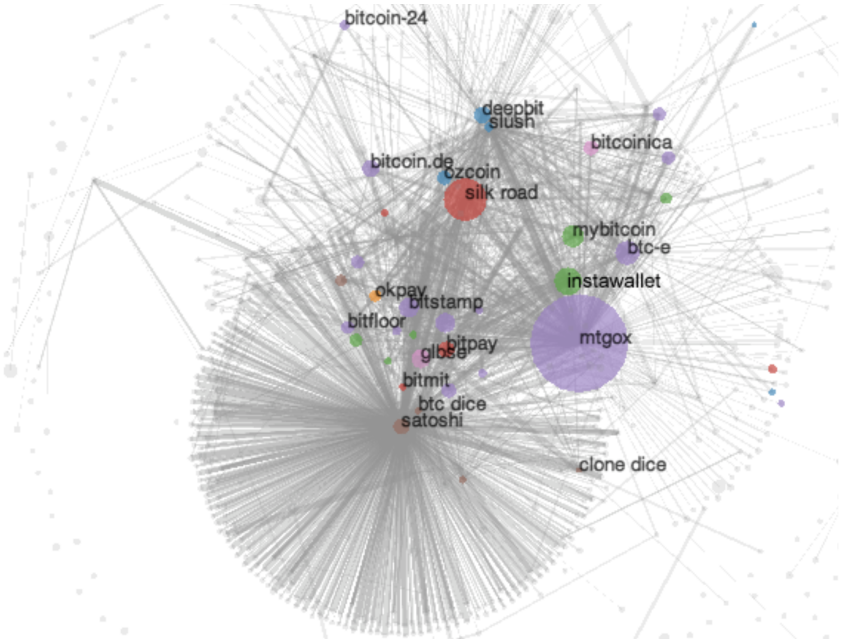

Figure 4: The deanonymization results from "A Fistful of Bitcoins." This clearly demonstrates that, using careful catered heuristics and large-scale graph clustering, certain portions of the network can be identified with high levels of confidence. The goal of this investigation, therefore, was to procure a graph similar in nature with an even greater number of addresses identified [26].

and Silk Road, among others" [26]. It was only after refining the second heuristic that this issue was resolved and there were distinct clusters formed.

As discussed in the paper itself, there were a few limitations of this investigation, some of which we sought to rectify herein. One such limitation was that "our new clustering heuristic is not fully robust in the face of changing behavior,... [i.e. in the face of] the gap [between actual and potential anonymity evolving] over time, and what users do to achieve stronger anonymity guarantee" [26]. One key limitation, therefore, is how these two heuristics have assumptions that fail to hold under some of the new anonymity services that have emerged atop the BTC network, such as BTC mixing.

These mixing services, also referred to as BTC tumblers, effectively mix up the input addresses to a new set of output wallets. In other words, mixing services act as trusted third parties, whereby participating members can transfer their funds to this trusted service wallet. The tumbler then distributes the money amongst three new wallets, which then randomly get assigned to the participating members. Of course, the value associated with the accounts involved in the mixing must be identical, else the owner of the accounts would be traceable through the mixing service. Assuming no collusion between the owner of the mixing service and any interested third parties, identities cannot

directly be traced between the input and output addresses. The heuristics catered and discussed above are unable to link addresses that have passed through such mixing services, since no pair of such addresses would appear in any connected transaction in the ways necessitated to form a link. This follows as it was the end user's intention in using the mixing service to disassociate the output from the input address, making it highly unlikely he would use the two in tandom in the future.

While this is the case, assuming the end user has some static characteristics, such as making transactions at only certain times of day or to only particular accounts, the original and mixed addresses should still be linkable, marking a clear deficiency in the heuristics catered above. We, therefore, sought to integrate a broader scope of heuristics to produce potential links between accounts. Of course, with this expanding set of heuristics arises the issue brought up within the "Fistful of Bitcoins" paper, namely that of procuring a series of false positives that ultimately collapse the overall graph into a small number of superclusters. We now delve further in depth as to particular methods employed to avoid said issue.

As previously mentioned, this issue was largely the result of their using union-find. In particular, the heuristics they catered were rigidly employed, meaning their involvement in clustering was completely binary. That is to say, if heuristic 1 was satisfied for a pair of vertices, the two were clumped together as being the same entity immediately and similarly for heuristic 2. This rigidness was the source of the collapse and resulted in the need to make refinements on the second heuristic. We wished to produce a means by which the heuristics could be procured with the system sufficiently flexible to automatically determine how effective the different heuristics were at linking two real-world entities and appropriately assign corresponding weights. **That is to say, rather than having a binary means of clustering addresses, we wished to expand to a continuous space.**

To do so, the natural extension was creating a **separate graph** from the transactions graph and running some full graph clustering algorithm on it. In other words, rather than using the heuristics **to directly cluster** the original graph, we sought to procure a **separate** weighted graph that embedded the information conveyed by the heuristics. Specifically, we would ideally have a graph with the property that if the edge connecting two vertices v_A, v_B had an associated weight we denote as w_{AB} and satisfied $w_{AB} > w_{BC}$, there would be a higher likelihood that the wallets associated with vertices A, B indeed correspond to the same real-world entity as compared to that for B, C.

The second main study in the realm of deanonymization, in fact, attempted to expand along

these lines. It specifically produced a probabilistic model that describes the probability two vertices in their graph correspond to the same real-world entity, where their graph contained data both from the on-chain heuristics and off-chain information. Examples of the former are exactly those from the "Fistful of Bitcoins" investigation whereas the latter are metadata associated with accounts of a particular entity. For example, "Satoshi Bones Casino uses 1change and 1bones prefixes and BTC-E exchange uses 1eEUR and 1eUSD prefixes" [14]. With the addition of this additional metadata "heuristic," however, came additional noise in the sense that there are plenty examples thereof that do not indicate the end-user is in fact the same. As discussed in their paper, "mostly informationless suffixes (for example, .com, .co, @gmail)" were ignored [14].

Towards that end, unlike the Fistful paper, the authors sought to procure a probabilistic model that effectively weighted the different heuristics based on their levels of confidence therein. In their words, their "proposed model is not intended to capture the probabilistic structure of the real world, but more to give an approach for systematical study of confidence trade-offs between different sources of information" [14]. This is similar in nature to the approach employed herein. However, their weights were manually tuned by the researchers such that the eventual graph clustering aligned with expected ground truths. Our investigation sought to perform such tuning automatically, in a manner that parallels modern machine learning algorithms, specifically neural networks.

All in all, the current degree to which deanonymization has been accomplished demonstrates it is in fact a viable route of investigation. Yet, their limitations point to a gap in the heuristics hereto employed, specifically regarding the rigidity with which they were used in clustering.

2.2 Data Procurement

The main goal for this section is to familiarize the reader with:

- How Blocksci, a investigative framework developed by [19], was well-suited towards constructing the desired heuristics graph.

As discussed in the previous section, the two pieces of data needed to be procured for this investigation, were (1) a subset of the vertices labelled with their real world identifications, serving as the ground truth against which performance of the deanonymization could be checked, and (2) the heuristics graph on which the clustering would be performed. For the purposes of this investigation, the former data were gratefully provided by the authors of the "A Fistful of Bitcoins" paper. As for the latter, however, another relevant investigation was conducted, from which a tool dubbed

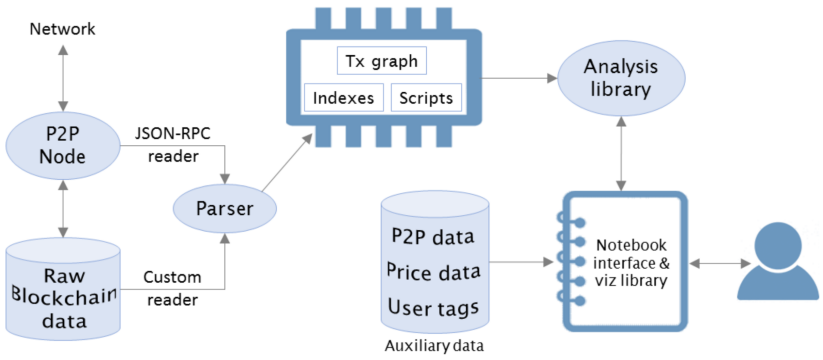

Figure 5: The main points of relevance in this figure are the parser and analysis library. As a brief summary, the P2P node refers to the BTC node running on the client, which is constantly getting updates as they are written to blockchain by miners. The raw blockchain data sitting on the client computer is parsed and streamlined to produce the optimized iterable data structure, which gets updated as new information arrives from the P2P node. With this iterable struture, an analysis library can be layered atop, to perform work manyfold, one example of which is procuring heuristics [19].

"Blocksci" was produced. Blocksci was born out of the need to perform analysis on public blockchain data. Per Blocksci's paper, "Bitcoin's blockchain alone is 140 GB as of August 2017, and growing quickly. This data holds the key to measuring the privacy of cryptocurrencies in practice...Blocksci is 15x-600x faster than existing tools [and] comes bundled with analytic modules" [19]. In particular, Blocksci fills the void of being able to iterate through the entire blockchain of transactions, allowing for heuristics to be catered effectively in an online setting. That is to say, unlike heuristics procured in previous investigations, a heuristic could be procured in a reasonable timeframe by employing Blocksci, allowing greater flexibility in their curation. Thus, one of the main bottlenecks in procuring more heuristics was eliminated through the development of Blocksci.

While specific details of its internal implementation are not greatly of relevance to this investigation, we present some to make clear the process by which heuristics are produced. This is more clearly elaborated upon in Figure 5. Specifically, "BlockSci's design starts with the observation that blockchains are append-only databases; further, the snapshots used for research are static. Thus, the ACID properties of transactional databases are unnecessary" [19]. In other words, by eliminating the unnecessary portions of database API, they streamlined the interface to end users.

In a similar vein, "The on-disk format of blockchains is highly inefficient for our purposes. It is optimized for a different set of goals such as validating transactions and ensuring immutability...whereas we aim for a single representation of the data that can it in memory" [19]. In other words, the parser for Blocksci implemented many optimizations revolving around making data struc-

tures not as bloated as they are when stored on-disk, as validation is no longer of primary concern. Using such streamlining and properties of caching, iteration through the blockchain data was significantly optimized. Thus, the main takeaway from this discussion was that Blocksci made the procurement of additional heuristics atop the transactions graph an efficient process.

2.3 Overview

Having established the current state of the art in the deanonymization space, we now turn to presenting the necessary technical background. We begin with the layout of the investigation, from which all the pieces to be presented fall into place. Specifically, using the capabilities of Blocksci as elaborated upon above, we procured a set of eight heuristics, about which there were varying degrees of confidence. For sake of completeness, the heuristics are laid out here, as they are presented in [18]. These heuristics, while necessary to initially construct the heuristics graph, are not critical to understand for the sake of this investigation, though brief explanations are presented for each for coherence. Specifically, many of these heuristics deal specifically with identifying change addresses as previously defined. The identification of change addresses very directly relates to the linking of wallets, as it can be assumed that the owner of a wallet and its associated change wallet are the same real-world entity. In other words, the identification of the link between a wallet and its change address(es) is effectively a subproblem of the overall real-world entity linking problem:

1. "Inputs spent to the same transaction are controlled by the same entity" [19], which exactly corresponds to that enforced in the original "Fistful of Bitcoins" investigation.

2. "If input addresses appear as an output address, the client might have reused addresses for change" [18]. In other words, there is seldom a circumstance where a user would wish to explicitly make a transaction to himself, making it plausible to assume that, if a wallet is seen as both an input and output of a transaction, this was an implicit artifact of the change address protocol.

3. "If all inputs are of one address type (e.g., P2PKH or P2SH), it is likely that the change output has the same type" [18]. Once again, the specifics of the inner workings of these different address types is not critical to understand. Specifically, "P2PKH (Pay to Public Key Hash) transaction is a simple spend to an address...[whereas] P2SH [Pay to Script Hash]...allows for a more complex and diverse set of transactions using the Bitcoin Script scripting language"

[31]. It naturally follows that an end-user would have similar behavior in terms of the types of transactions he wishes to partake in, suggesting change and input addresses would tend to be of the same wallet type.

4. "Most clients will generate a fresh address for the change. If an output is the first to send value to an address, it is potentially the change" [18]. In contrast to the second heuristic listed, this captures the more natural and common case, where the user does not make explicit the change address to be used. In turn, as change addresses are one-time uses in most circumstances, there is an extremely high probability they will have no transactions prior. Of course, the main hole in this heuristic is that the reverse does *not* necessarily hold. In other words, while nearly all change addresses have no prior transactions, it is entirely plausible that wallets with no prior transactions are not change wallets. All real-world entities too will have some transaction that is their "first" receipt, meaning false links would be procured in those circumstances. However, it seems reasonable to believe that the number of change addresses far exceeds the count of real-world entities using BTC, meaning this heuristic would still carry information.

5. "Bitcoin Core sets the locktime to the current block height to prevent fee sniping. If all outpus [*sic*] have been spent, and there is only one output that has been spent in a transaction that matches this transaction's locktime behavior, it is the change" [18]. Briefly, the locktime corresponds to an additional piece of information that can be baked into transactions that "allows signers to create time-locked transactions which will only become valid in the future, giving the signers a chance to change their minds" [5]. Hence, it follows that, if an output has activity that very closely aligns with the time-lock enforced on the transaction, its user was aware of when the time-lock would release, suggesting this output's owner is in fact the owner of the original input address.

6. "If there exists an output that is smaller than any of the inputs it is likely the change. If a change output was larger than the smallest input, then the coin selection algorithm wouldn't need to add the input in the first place" [18]. In other words, using a change address to receive more tokens than were originally in the account is not expected per the implementation in the BTC protocol, meaning such behavior would be indicative of a particular wallet *not* being a change address.

7. "If *tx* is a peeling chain, returns the smaller output" [18]. This idea of a "peeling chain"

was one introduced in the "Fistful of Bitcoins" paper, and essentially corresponds to the idea that a person can make a *series* of change address hops. Specifically "a single address begins with a relatively large amount of bitcoins...A smaller amount is then 'peeled' off this larger amount, creating a transaction in which a small amount is transferred to one address...and the remainder is transferred to a one-time change address. This process is repeated...until the larger amount is pared down" [26]. Hence, the smaller output that remains at the end of the chain is expected to correspond to the change address of the transaction originator.

8. "Detects possible change outputs by checking for... values that are multiples of 10^{digits}" [18]. This simply checks those addresses in the outputs that received a well-formed number of BTC on the same order of magnitude as the input, reflective of people's desire to maintain clean records of their funds.

With these heuristics, unlike previous investigations, we did not directly manipulate and collapse nodes of the original Bitcoin transaction graph. Instead, we procured a *separate* graph, in which nodes correspond to wallets and edges represent the presence of **a heuristic**. Note that this is a *huge* departure from previous studies, as was mentioned in passing. **That is to say, the heuristics themselves no longer represented *actions* to transform the transaction graph but rather *properties* of pairs of vertices.**

If multiple heuristics were found to be true between two nodes, the edge connecting said nodes had the sum of the weights associated with the corresponding heuristics. In other words, denoting the weights for these heuristics as $w_1, w_2, ..., w_8$ where w_i naturally takes some weight $\in [0, 1]$ corresponding to heuristic i and assuming these weights are normalized, such that $\sum_i w_i = 1$, if heuristics 2,3, and 4 came up as true between vertices A, B, the weight associated with edge e_{AB} would be $w_2 + w_3 + w_4$.

Upon curating this heuristics graph, we sought to subsequently perform clustering. Unlike the previous investigations, however, clustering was implemented in the traditional graph theory sense rather than the union-find means of previous papers. Such "standard graph clustering" techniques entail algorithms like k-means, spectral, and DBSCAN, the specifics of which are elaborated upon subsequently in the background. This eventual clustering represents our final result, whose performance was measured by comparing against known addresses using multiple notions of clustering accuracy. There are three points, however, that require major emphasis in presenting these research goals.

The first is that speed was of *critical* importance in choosing the clustering algorithms. The heuristics graph that was catered was on the order of scale of 180 million vertices and nearly 10 billion edges, making any traditional means of clustering quite infeasible. Thus, a majority of this research endeavor was spent on investigating methods of achieving clustering in *very* time efficient manner, albeit occasionally sacrificing some metrics of "correctness." Additionally, this endeavor marks a *dramatic* expansion over the previous studies in deanonymization. While they too focused on clustering addresses with heuristics, their results relied completely on a standard union-find algorithm. In turn, they had dramatic loss of information by being forced to discard heuristics that caused stray edges to form, whereas well-formed graph clustering algorithms can simply sift through such information automatically. They additionally could not cater additional heuristics easily. With the switch to a more dynamic clustering method of deanonymization, the clustering could also extract non-planar information and only draw *partially* from particular heuristics if they were determined to be noisy. In turn, with this expanded flexibility, future studies can easily try adding additional heuristics, with minimal changes to the underlying deanonymization procedure. Further, the goal of this particular investigation was **solely** to obtain clusters of BTC wallet addresses; in other words, while going from these clusters to real-world identities is trivial given seed points, this was not an explicit goal herein. In other words, as previously described, the structure of our output was to follow along the lines of saying: `set(1,3,4,6),set(2,8),set(5,7,9,10)` are the BTC wallet clusters, where each `set` is a single real-world entity, though its explicit identity would not necessarily be known. It follows that future investigations, namely when such information reveals itself through side channels, can take the resultant clusters found here and associate each with an explicit name.

2.4 Stochastic Block Model

The main goals for this sections are to familiarize the reader with:

- The concept of the stochastic block model (SBM), a standard graph model used to model the case of having many clusters whose nodes are more likely to be connected to one another than to nodes of other clusters.
- The need to have such a model as it relates to modelling the overall heuristics graph of the BTC network, namely serving as a prototype to understand the various clustering techniques and preprocessing steps investigated herein.

- Various properties relating to the ability to cluster the SBM, which may have potential ramifications in studying the associated clustering for BTC wallets.

That being said, this investigation began by modelling the heuristics graph as a simple SBM, on which much of the initial experimentation and testing was performed for their use on the final dataset. The use case of studying a model rather than the full dataset is twofold. The first is that it allows rapid testing. That is to say, by being able to control exactly the size of the test graphs, many runs could be conducted for a given set of algorithms of interest. In line with that, the second benefit is having a defined metric by which performance could be measured. While there is a subset of known data points in the entire dataset, by having access to the full truth for a simulation, specific pain points could be evaluated. For example, if it turned out a particular algorithm worked well as measured by metric A but not so by metric B and metric A was of greater relevance to the final investigative result, we could easily make use of said algorithm in the final clustering. In other words, such experiments made it such that the algorithms' strengths and weaknesses could be understood in greater detail, since trials could be procured that targeted different circumstances.

In line with that, we begin our discussion with an introduction to the stochastic block model, commonly shortened to SBM. The SBM is a "model for random graphs on n nodes with two equal-sized clusters, with an between-class edge probability of q and a within-class edge probability of p" [28]. We, instead, focus on an expanded view of the SBM. That is to say, rather than solely being between two clusters, we define the SBM to have n clusters such that for any clusters C_i, C_j, the probability a vertex in the first will be connected to a vertex in the second is p if $i = j$ and q if $i \neq j$, where $p > q$. In other words, this model produces a random graph comprised of many clusters, of respective sizes $n_1, n_2, ...n_k$, with vertices more likely to be connected with vertices of their cluster as compared to those of other clusters. This, therefore, roughly corresponds to our idealized form of the heuristics graph, namely where the heuristics would be more likely to be present between vertices that belong to the same cluster (i.e. same real-world entity). The main leak in this simplification arises from the fact that, in the case of the SBM, a single run will either have edges or not, with probability p or q depending on the vertices of interest whereas the heuristics graph *always* has the edge, just adjusted to a higher or lower weight. However, in expectation, namely when averaged over a series of n runs as $n \to \infty$, the adjacency matrix of SBM resembles that of the heuristics graph, assuming the the heuristic weights have been normalized, i.e. of the form:

$$C = \left[\begin{array}{ccc|c|c|ccc} p & p & p & & & q & q & q \\ p & p & p & Q & \cdots & q & q & q \\ p & p & p & & & q & q & q \\ \hline & Q & & P & \cdots & & Q & \\ \hline \vdots & & & \cdots & \ddots & & \vdots & \\ \hline & Q & & \cdots & \cdots & & P & \end{array}\right]$$

Where each of the block P, Q entries are simply referring to $(p, q)\mathbb{1}\mathbb{1}^T$ respectively, as indicated in some of the matrix entries. The main limitation in the SBM representation of the heuristics graph is that there is little to suggest different clusters will have similar weights in the heuristics graph. After all, there may be two sets of vertices that should both be clustered, where it is completely obvious that one set of vertices should be clustered, while the other has higher uncertainty. In this case, the first would likely have a higher weight due to more heuristics being marked "True" and the second less so. Naturally, these "weights" correspond to the entries in the weighted adjacency matrix. Thus, the natural extension to the SBM would simply involves having a different p_i for each cluster C_i. Yet, this complicates the model to a degree that gaining the desired intuition of the clustering algorithms would not come with the same ease as it would otherwise.

As a result, we sacrifice this piece of accuracy with the assumption that algorithm performance on this special case will scale appropriately to the general case. That is to say, we assume that, if algorithm A has better performance on the SBM than B, it should similarly perform better on the generalized case with varying p_i, though the two very likely will have degraded performance. The intuition reasoning behind said assumption is that the main detractor of performance comes when there are p, q values that are comparable. Consider the edge case where $p = q$; clearly, this case can have no algorithm that performs clustering with any better accuracy than 50%. Thus, the bottleneck of the performance in any SBM will be the lowest p_i value and highest q_{ij}, where q_{ij}

denotes the natural extension of the notion of q to arbitrarily many clusters, namely the probability that vertices from C_i, C_j will be adjacent. However, we can simply consider a simulation in which p, q are relatively near one another and evaluate the algorithms' performances on said case. Since this marks the worst/most difficult case, it follows that the algorithm that performs best here will similarly scale to be the best in the easier cases. This intuition, however, is confirmed through empirical testing in Section 3.

We finally present one noteworthy theorem of interest with relevance to SBMs, specifically the limitations with which they can be clustered. This makes more concrete the notion that if p, q are "sufficiently close," the graph clustering cannot be performed in any manner better than chance in the limit of large graphs (where $p = a/n, q = b/n$ for a graph with a total of n vertices):

Theorem 2.2. *If $a + b > 2$ and $(a - b)^2 \leq 2(a + b)$ then, for any fixed vertices u and v:*

$$\mathbb{P}_n(\sigma_u = +|G, \sigma_v = +) \to \frac{1}{2} \text{ a.a.s. [28]}$$

Where *a.a.s.* corresponds to asymptotically almost surely convergence. The specifics of this notion of convergence are not necessarily of relevance, though we put it here for sake of completeness. Namely, this corresponds to when "An event E...holds with probability $1-o(1)$, thus the probability of success goes to 1 in the limit $n \to \infty$" [42]. In other words, we cannot even determine whether two random vertices in the SBM are the same label or not better than random chance if a, b are not sufficiently different. As demonstrate in Figure 6, this corresponds to where p, q are quite close to one another. Note that this region is symmetric about the line $x = y$, since the notions of p, q can be flipped at any point and the complexity of clustering the model would remain the same. Since we are dealing with a generalized form of the SBM, we provide a straightforward extension of this theorem as follows:

Corollary 2.2.1. *Assuming $G = \cup_i G_i = \bigcup_i (n_i, \frac{a_i}{n_i}, \frac{b_i}{n_i})$, if for some choice i, $a_i + b_i > 2$ and $(a_i - b_i)^2 \leq 2(a_i + b_i)$ then the problem of clustering is not solvable as $n \to \infty$.*

Proof. The proof follows as a trivial consequence of the previously relayed theorem. A clustering problem is considered solvable when "one can *a.a.s.* find a [cluster] which is positively correlated with the" true cluster [28]. Suppose, now, for sake of contradiction that we could solve this generalized SBM clustering problem in the limit of $n \to \infty$. In solving the generalized clustering problem, it follows that we must further be able to successfully perform clustering for any subgraph of the

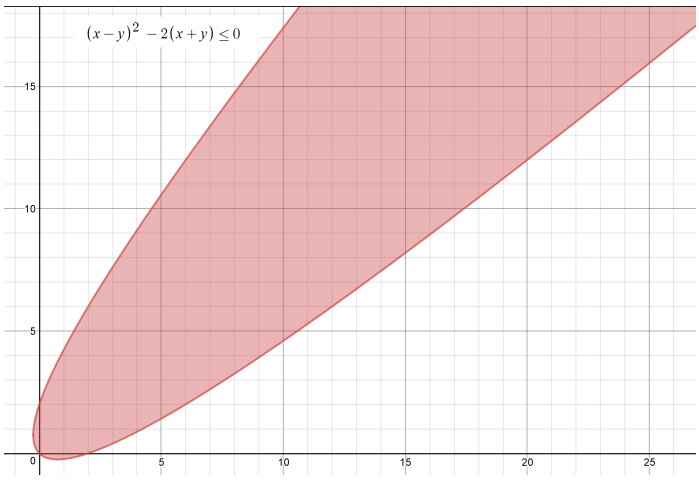

Figure 6: This marks the region in which no algorithm can perform clustering better than by chance. If the heuristics graph procured fell within this region, seeing as the graph was massive (i.e. could reasonably be seen as reaching the limit of $n \to \infty$ for practical purposes), it seems entirely infeasible that any algorithm would have reasonable performance.

original graph. In other words, if we have an algorithm capable of clustering G as $n \to \infty$, we must similarly have an algorithm to cluster any $S \subset G$ for $n \to \infty$.

Such an algorithm can simply be formulated as follows. Denote the original graph as G and its associated clustering algorithm as A. We wish to formulate some *subalgorithm* A_S that is capable of clustering $S \subset G$. Now define A_S to be the restricted output of A when run on G to the vertices of S. In other words, construct the canonical sub-clustering from that of the overall graph.

From here, therefore, we can consider $G_i \subset G$. It follows that, in being able to cluster G, we can similarly cluster G_i as $n \to \infty$. This result, however, is in contradiction to Theorem 2.2, meaning this overall clustering task is unsolvable. □

Hence the main takeaway from this section is that the SBM serves as a very pertinent simplification of the heuristics graph on which many of the properties of the clustering algorithms could be understood through relevant experiments.

2.5 Clustering Algorithms

Moving on from the SBM, we look to discussing the second topic at hand, specifically graph clustering. We present the relevant algorithms, followed by specific preprocessing extensions that we employed due to the massive scale of the data considered herein. This discussion of clustering is briefly summarized in a table presented following all the sections, namely Table 1. We further assume

that, when referring to graph clustering, we wish to separate the set V of vertices corresponding to G into sets $\{C_1, C_2, ..., C_n\}$, such that $\bigcap_i C_i = \emptyset$ and $\bigcup_i C_i = V$, where the objective is to minimize the "distance" between vertices in a single C_i and maximize that between any C_i, C_j for $i \neq j$, where "distance" often has a natural interpretation as related to edge weights in the graph. Thus, the objective of this overall section is to familiarize the reader with all the clustering algorithms, both in terms of their advantages and inner workings.

2.5.1 k-means

The main goal for this section is to familiarize the reader with:

- K-means clustering, a very standard and time-tested graph clustering algorithm whose main advantage is being very time-efficient. This clustering too served as the baseline comparison for accuracy metrics.

k-means is the canonical example of a clustering algorithm. Given its age and the continual improvements in efficiency, k-means can be run efficiently on massive datasets. Specifically, k-means works "given an integer k and a set of n data points in $\mathbb{R}^d$, the goal is to choose k centers so as to minimize ϕ, the total squared distance between each point and its closest center" [2]. The clustering of points, therefore, is simply according to which mean they are located nearest to. The determination of the best such k clusters, however, is an NP-hard problem, for which a heuristic has been developed and repeatedly confirmed in practice. This employed heuristic works as follows: it "begins with k arbitrary 'centers,' typically chosen uniformly at random from the data points. Each point is then assigned to the nearest center, and each center is recomputed as the center of mass of all points assigned to it. These last two steps are repeated until the process stabilizes" [2].

The result of this is presented in Figure 7, which corresponds to the canonical k-means visualization, dubbed a Voronoi diagram. Such diagrams effectively demonstrate how k-means divides the associated space in which the data are embedded into regions and associates to each region a particular label. From this visualization alone, it becomes clear that k-means is only capable of extracting planar clusters as, in producing this visualization without a loss of information, we have a *planar* presentation of the division of the data.

As with many of the other algorithms considered below in this paper, k-means requires prior knowledge of the number of clusters. We take an aside to address this point. While it may seem wholly arbitrary that the number of clusters be known ahead of time, as there is little way to

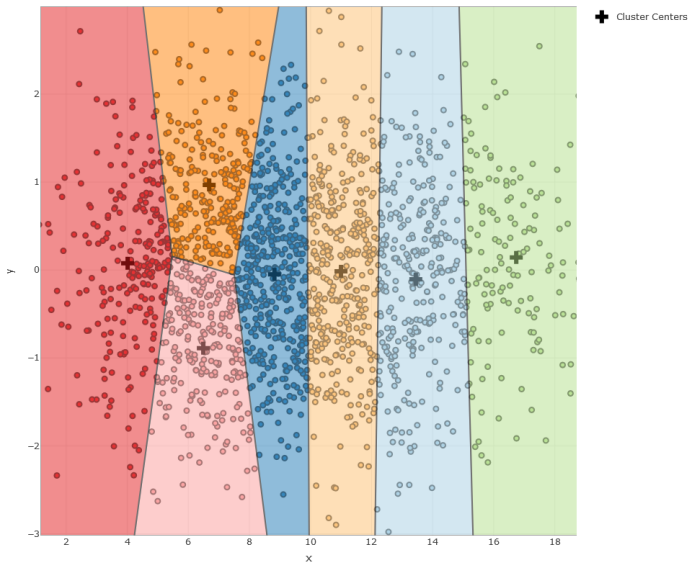

Figure 7: A Voronoi diagram is an effective and standard method of visualizing k-means clustering results, where the data are drawn in a 2D space and the entirety of the plane is divided into regions, with each one is associated with a single of k cluster centers produced from k-means. This notion of Voronoi diagram can easily be extended from its 2D form to higher dimensions if so desired [34].

ascertain ahead of time how many *unique* users of Bitcoin there are on the network, this serves as a means by which the granularity of results can be controlled. That is to say, given two identical networks, we could potentially extract more fine clusterings with higher levels of k. For example, in the edge case of $k = 2$, the clustering would likely resemble something along the lines of "Coinbase accounts" and "non-Coinbase accounts," namely a separation of a supercluster from the remainder of the graph. We can, therefore, imagine that as $k \to \infty$, clusters associated with individual people or small organizations would form out of the massive agglomerated "Non-Coinbase" cluster. In the limit, however, we would simply end up with singleton clusters, which would serve little purpose for identifying real-world entities. The selection of k, therefore, is a balancing act between getting fine grain separation of individual entities and uninformative results.

Thus, the main use case of k-means herein was to serve be as a baseline for comparing clustering accuracies. That is to say, we expected *all* other algorithms employed to have superior performance than k-means, though they may perform worse with respect to time complexity.

2.5.2 Spectral

The main goals for this section are to familiarize the reader with:

- Graph Laplacians, a simply defined construct whose eigenvalues and eigenvectors contain significant information regarding the clustering of its associated graph. We similarly present theorems as they relate to the eigenvectors of the Laplacian.
- K-means spectral clustering, which relies on embedding data points into a space associated with the Laplacian eigenvectors, followed by performing k-means clustering.
- Hierarchical spectral clustering, which relies on the embedding associated with a particular eigenvector (second) of the Laplacian and properties of its components.
- The comparison of the two spectral clustering techniques, with k-means being significantly more time-efficient and hierarchical capable of extracting more non-planar clusters.

Spectral clustering, as implied by its name, deals with clustering on the basis of eigenvalues and eigenvectors. Which eigenvectors are to be used for such clustering is not immediately obvious. However, it turns out to be those of a matrix that has fundamental importance to its corresponding graph: the Laplacian. While the Laplacian has differing definitions by context, in the realm of graph theory, it simply refers to:

$$L = D - W$$

Where hereafter we denote the Laplacian matrix as L; the degree matrix as D, which contains the degree of vertex i in position D_{ii} and 0s elsewhere; and the weight matrix as W, which contains the weight of edge e_{ij} in position D_{ij}. Since D, W are both symmetric matrices in the case of an undirected graph, L too is symmetric. Having presented the definition, we follow up with some relevant theorems regarding this Laplacian matrix [25]:

Theorem 2.3.

- *The smallest eigenvalue of L is* 0*. The corresponding eigenvector is the constant one vector* $\mathbb{1}$*.*
- *L has n non-negative, real-valued eigenvalues* $0 = \lambda_1 \leq \lambda_1 \leq \cdots \leq \lambda_n$*.*

From this theorem spawn several other results which have direct consequences in the use of eigenvectors for the purposes of graph clustering. The first theorem is as follows:

Theorem 2.4. *Let G be an undirected graph with non-negative weights. Then the multiplicity k of the eigenvalue 0 of L equals the number of connected components $A_1, \ldots, A_k$ in the graph. The eigenspace of eigenvalue* 0 *is spanned by the indicator vectors $\mathbb{1}_{A_1}, \ldots, \mathbb{1}_{A_k}$ of those components.*

While we refer to [25] for the whole proof of this result, we present a sketch of the primary portion of this proof from which an intuitive understanding arises. Specifically, consider the idealized case of a graph G containing clusters such that vertices within the clusters are fully connected and those in different clusters are not adjacent. The corresponding graph Laplacian for such a G would be:

$$\begin{bmatrix} L_1 & & & \\ & L_2 & & \\ & & \ddots & \\ & & & L_n \end{bmatrix}$$

Where each L_i is the Laplacian of the subgraph that only contains cluster i. From here, we notice that the eigenvectors of L are simply those that correspond to those of each of the L_i with 0s filling up the entries that do not correspond to the L_i matrix entries. By the preceding thereom, we know that each of these sub-Laplacians has an eigenvector of $\mathbb{1}$ corresponding to the eigenvalue 0, meaning that the multiplicity of the eigenvalue 0 relates to how many block entries there are in this Laplacian matrix, which precisely corresponds to the number of clusters there are.

We now wish to understand how to go about using such eigenvectors to accomplish clusterings. Specifically, to do so, consider once again the idealized case. In creating a matrix of these k eigenvectors as columns, we are left with a matrix of the form (where we are depicting the matrix as if the clusters are all of size 2):

$$M = [\mathbb{1}_{A_1}^T, \mathbb{1}_{A_2}^T, \ldots, \mathbb{1}_{A_k}^T] = \overbrace{\begin{bmatrix} 1 & 0 & \ldots & 0 \\ 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{bmatrix}}^{k}$$

From here, we simply notice that there are precisely as many rows as there were originally vertices in the graph, since each column corresponds to an eigenvector. Noticing that, it becomes clear that if we associate each vertex with its corresponding row, these vertices can now be trivially clustered using k-means with perfect accuracy. Thus, the main takeaway from this example is understanding that the eigenvectors corresponding to eigenvalues of lowest magnitude essentially help transform

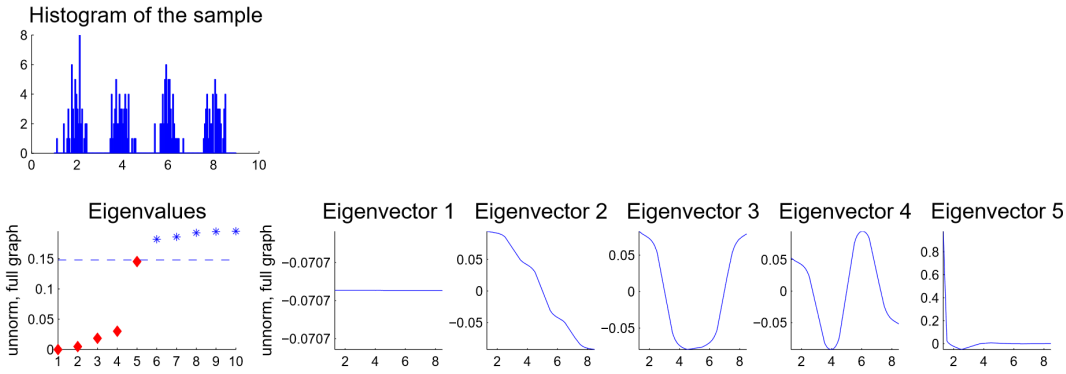

Figure 8: The top left is a histogram of the data spread, and the remainder of the picture corresponds to the eigenvalues and eigenvectors of the Laplacian, as discussed in the exposition. From this, it is clear that a well-separated dataset has a marked jump between eigenvalues that correspond to the the clusters and those that do not [25].

the vertices into a new space that much more closely correspond to their associated cluster, from which the k-means algorithm can trivially be employed to complete clustering. The eigenvectors with lowest eigenvalues are employed as they are the ones closest to the idealized $\mathbb{1}_{A_i}$ vectors.

An interesting addition supplied by spectral clustering arises in its eigenvalue roughly serving to indicate how separable certain subsets of the vertices are. As a result, we can estimate the number of clear clusters, namely by determining how many eigenvalues are near 0 prior to there being a jump in magnitude, as portrayed in Figure 8.

This procedure is summarized in the presentation of Algorithm 2. Note that their conception of spectral clustering involves beginning with a similarity matrix, namely a symmetric matrix where entry S_{ij} corresponds to some notion of "similarity" between vertices i, j. It is from this matrix that a similarity graph is *constructed* and its Laplacian determined. There exist many ways by which such a similarity graph can be constructed. As enumerated in [25], three such ways are as follow:

- **ϵ-neighborhood graph**: Connect vertices together if the similarity metric defined between the two of them exceeds some threshold parameter value of ϵ. That is to say, the adjacency matrix will have edges according to $I_{e_{ij}>\epsilon}$.

- **k-nearest neighbors**: Connect a given vertex to its k nearest neighbors in sense defined by the similarity metric, i.e. create an edge for the vertices corresponding to the k highest similarity measures. While this is normally asymmetric, simply avoid this problem by creating an undirected edge between the two vertices. That is to say, an edge between v_i, v_j will be

present if *either* v_j is one of the k-nearest neighbors for v_i or vice versa (or both).

- **Fully-connected graph**: Connect all edges, weighing the edges with some similiarity function, i.e. a Gaussian $G(v_i, v_j) = \exp\left(-\frac{||v_i - v_j||^2}{2\sigma^2}\right)$

For our particular case, we elected to use the third method of catering a similarity graph, although future elaborations on this work may expand this assumption and attempt other methods to determine their relative success.

Algorithm 1 Constructs Laplacian from the similarity matrix [25]

procedure ConstructLaplacian(S,K)
Input: Similarity matrix $S \in \mathbb{R}^{n \times n}$
Input: Cluster count k
    $G \leftarrow$ Similarity graph for S
    $W \in \mathbb{R}^{n \times n} \leftarrow$ Weighted adjacency matrix for G
    $L \leftarrow D - W$
Output: Laplacian matrix L

Algorithm 2 k-means based spectral clustering [25]

procedure SpectralClustering-Kmeans(S,K)
Input: Similarity matrix $S \in \mathbb{R}^{n \times n}$
Input: Cluster count k
    $L \leftarrow ConstructLaplacian(S, k)$
    $v_1, v_2, ..., v_k \leftarrow$ first k (sorted by λ_i) eigenvectors of L
    $V \in \mathbb{R}^{n \times k} \leftarrow [v_1^T, v_2^T, \ldots, v_k^T]$
    $(y_i)_{i=[1:k]} \leftarrow V_{i,:}^T$ (i.e. row i of V)
    $(C_i)_{i=[1:k]} \leftarrow kmeans((y_i)_{i=[1:k]})$
Output: $(C_i)_{i=[1:k]}$

In addition to this method of implementing spectral clustering, however, there is another which relies on a different principle, albeit still related to the spectrum of the Laplacian matrix. In this alternate approach, the "eigenvector u_2 corresponding to λ_2 (the second-smallest eigenvalue) is computed, and the vertices of the graph are partitioned according to the values of their corresponding entries in u_2" [17]. In other words, the second eigenvector is computed and used to partition the graph into two (or three) sections, specifically corresponding to the signs of the components of the eigenvector. This process can be subsequently repeated a number of times to achieve the desired number of clusters. Subsequent iterations bisect the cluster with the lowest second eigenvalue, since this effectively corresponds to how separable the cluster is, meaning after k iterations $k + 1$ clusters would exist.

Returning to the previous example at hand, namely the idealized case, intuition for this algorithm becomes clear. Specifically, we have that the eigenvector of the first eigenvalue (which *always* corresponds to an eigenvalue of 0) will simply be $\mathbb{1}$, meaning there is no information to be gleaned from its entries. Instead using the second eigenvector, we have something of the form $[\mathbf{0}_{A_1}, \ldots \mathbf{0}_{A_{k-1}}, \mathbb{1}_{A_k}, \ldots \mathbf{0}_{A_n}]$. In this case, we see that partitioning by sign would effectively separate a cluster (A_k in this particular instance) from the remainder of the graph. Presumably, in the following iteration, the second eigenvalue corresponding to the subgraph of $G \backslash A_k$ will be less than that of A_k, meaning it will be partitioned in the following iteration. Once again, the second lowest eigenvalue is chosen as it roughly demonstrates that the corresponding cluster is the closest to the idealized separable case of all those present in the clustering. We refer to this version of spectral clustering as "hierarchical," as it involves repeated bisection and employs a top-down separation method as contrasted with the k-means spectral implementation, which clusters all vertices at once. It seems natural, therefore, to expect hierarchical to be more accurate but significantly slower, due to its more careful handling of separation, than its k-means counterpart, a trend we investigated more fully empirically. This procedure is fully summarized in Algorithm 3.

Algorithm 3 Hierarchical based spectral clustering [25]

procedure SPECTRALCLUSTERING-HIERARCHICAL(S,K)
Input: Similarity matrix $S \in \mathbb{R}^{n \times n}$
Input: Cluster count k
    $C \leftarrow [G]$
    for $i = 1 : k - 1$ **do**
        $\lambda_{min} \leftarrow None$, $v_{min} \leftarrow None$, $j_{min} \leftarrow None$
        for $j = 1 : len(C)$ **do**
            $W \in \mathbb{R}^{n \times n} \leftarrow$ Adjacency matrix for $C[j]$
            $L_j \leftarrow D_j - W_j$
            $\lambda_2, v_2 \leftarrow$ 2nd eigenvalue, eigenvector of L_j
            if $\lambda_2 < \lambda_{min}$ **then**
                $\lambda_{min} = \lambda_2$
                $v_{min} = v_2$
                $j_{min} = j$
        $C_{new} = [Set(), Set()]$
        for $k = 1 : len(v_{min})$ **do**
            if $v_{min}[k] > 0$ **then** $C_{new}[0].add(v_{min}[k])$
            else$C_{new}[1].add(v_{min}[k])$
        $C[j_{min}] \leftarrow C_{new}$
Output: C

Thus, having discussed spectral clustering, we briefly discuss its role in this investigation. Spectral clustering represents one of the major departures from previous studies that we wished to bring

forward to this investigation, since it markedly involves attempting to perform clustering in a higher dimensional space, to capture non-planar clusters. This contrasts previous studies, whose clustering, by virtue of directly working with the transaction graphs, could not obtain any results that are non-planar. Hence, we wished to observe whether any clusters are more easily discernable under the mapping produced in spectral clustering as compared to their planar counterparts.

2.5.3 DBSCAN

The main goal for this section is to familiarize the reader with:

- DBSCAN, a graph clustering technique mostly reserved for large graphs, that attempts to achieve the high speed of k-means clustering and non-planar clustering of spectral methods.

DBSCAN is markedly different from the previous two algorithms in that it expressly seeks to be an algorithm that does not need a number of clusters to be supplied. Much in the way spectral clustering can be used to estimate the number of clusters using a "separation gap," DBSCAN similarly relies on two user-supplied parameters to effectively serve as the thresholds for vertices to be considered as part of the same cluster, though the manner in which these parameters serve as a threshold is slightly more involved. Prior to diving more in depth about its inner workings, it is noteworthy that DBSCAN additionally does not necessarily cluster all vertices. That is to say, many of the vertices may end in a state of being in an "undefined" category, outside of any clusters. This is quite natural, in that there are often circumstances where vertices are simply shoehorned into a particular premade cluster simply because it is the best of the available options when in reality it is clear said vertex does not fit well in *any* of these clusters.

Returning to the algorithm at hand, the user-supplied parameters are typically denoted as ϵ and $minPts$. From these, a set of "core points" $\{p_i\}$ are defined, where a core point is any point $p \in G$ such that there are $\geq minPts$ other points q such that the distance $d_{pq} < \epsilon$. That is to say, a core point essentially captures the intuitive notion of certain vertices being "central" in the graph, i.e. being well-connected to a significant number of other vertices in the graph. From there, we define one vertex v_B as being *directly* reachable from another v_A if the other vertex is a distance $d_{AB} < \epsilon$. We now define an arbitrary vertex v_B as being reachable from another v_A if there is a path of pairwise directly reachable core points that terminate in v_B. This is more clearly illustrated in Figure 9.

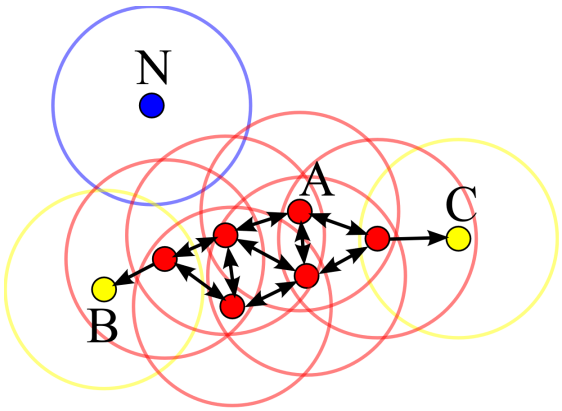

Figure 9: Here the $minPts = 4$. Red points correspond to the core points, yellow those that are reachable from core points, and blue that not reachable from any. Thus, in this case, all the vertices would be clustered together with the exception of the blue vertex [12].

Table 1: Summary of clustering algorithms [9].

Method name	**Parameters**	**Scalability**	**Use Case**
K-Means	number of clusters	Very large n_samples, medium n_clusters with Minibatch code	General purpose, even cluster size, flat geometry, not too many clusters
Spectral clustering	number of clusters	Medium n_samples, small n_clusters	Few clusters, even cluster size, non-flat geometry
DBSCAN	neighborhood size	Very large n_samples, medium n_clusters	Non-flat geometry, uneven cluster sizes

To finally construct the clusters, we simply go through the set of core points and find all other vertices that are reachable from these seeds. If a vertex is not reachable from any core points, it simply remains unclustered in the final result [11].

The main advantage of DBSCAN is that it is extremely well suited for large-scale clustering. DBSCAN additionally is capable of capturing non-flat cluster geometry, similar to spectral clustering, due to its ability to capture long-link separated clusters. Thus, it provides a means of clustering that effectively takes the superior speed of k-means and clustering flexibility of spectral into a single algorithm. Unfortunately, DBSCAN is extremely sensitive to parameter selection, and the choices are oftentimes non-obvious, as they hold little semantic meaning outside of the context of the algorithm itself.

To summarize, we present these strengths and necessary parameters are presented in Table 1. Note that this summary table was procured by `scikit-learn`, meaning there could be variations in implementation details of these algorithms that see parameter use differently.

2.6 Clustering Evaluation

The main goals for this section are to familiarize the reader with:

- The notion that evaluating graph clustering techniques is not a completely straightforward prospect, with different metrics being suited for different applications. Briefly, the intuitive notions captured by these metrics are as follow, where the "clusters" refer to those predicted by the algorithm and "classes" the underlying true groups to which the nodes belong:
  - **Purity:** How "mixed" the clusters are of nodes of different classes. Thus, a cluster containing nodes only of a single class has a purity of 1.0 whereas one with nodes of all types has a purity tending towards 0.0.
  - **Normalized Mutual Information (NMI):** How much knowing the guessed clusters reveals about the underlying classes. Thus, if the clusters were divided exactly as were the classes, knowing the cluster would provide full information of the underlying class, meaning NMI would be 1.0. If, however, the clusters were split in a manner not remotely paralleling that of the underlying classes, knowing the cluster a vertex were in would effectively reveal nothing about which underlying class it belonged to.
  - **Rand Index (RI):** Simply measures the "total correct" when considering *all* the pairwise decisions that are implicitly made in clustering vertices, i.e. the decision for any pair of vertices to be in the same or different clusters.
  - **F-score:** Similar to the RI, except this captures the notion that there are cases where we wish to penalize false positives and false negatives to different extents.
- The limitations of some of the metrics, such as rewarding clustering results that are too compartmentalized. For example, assigning each node to its own cluster achieves high purity, whereas NMI handles this issue.

Having discussed these clustering algorithms, it seems natural that there be metrics by which they can be compared. This seemingly trivial property, however, is not completely obvious in its definition as it may seem upon first glance. We assume for the sake of this discussion that there is some known ground truth clustering of the vertices. Unlike the DBSCAN assumption, we further assume that *all* vertices are associated with one (and exactly one) cluster of the ground truth.

Having said that, there emerge four natural metrics of evaluating clustering, per [15]. The notation employed by this resource, and that which we extend for the remainder of this section, is

"$\Omega = \{\omega_1, \omega_2, \ldots, \omega_K\}$is the set of clusters and $\mathbb{C} = \{c_1, c_2, \ldots, c_J\}$ is the set of classes" [15]. In other words, Ω corresponds to the guessed clusters while C is the set of ground truth clusters, which are referred to as "classes."

The first metric is referred to as *purity* and is defined as:

$$\text{purity}(\Omega, \mathbb{C}) = \frac{1}{N} \sum_k \max_j |\omega_k \cap c_j|$$

Roughly, the intuition behind purity is that a clustering with high purity has clusters such that the elements that are clustered together originate from the same class. Explained in words, we calculate purity by going through the clusters and associating each with a class. After all, the original clustering has no natural relation to the underlying classes. For example, we may be running a clustering algorithm on a graph with vertices $V = [1 : 10]$ and an underlying class separation of `1:set(1,6), 2:set(2,3,8), 3:set(4,5,7,9,10)`, where the integers represent the class associated with the proceeding set. The results of running our clustering algorithm may similarly be something of the form: `set(1,3,4,6),set(2,8),set(5,7,9,10).` However, there is no immediately obvious manner of associating each of our produced clusters to a *single* integer class as was originally the case. Purity effectively goes about doing this in the most natural manner, that is, by taking each cluster produced and seeing with which class it has the greatest overlap. Thus, returning to our example, `set(1,3,4,6)` would be associated with class 1, as $|$`set(1,6)` $\cap$ `set(1,3,4,6)`$| = 2$, whereas it $= 1$ for each of the two other classes. From here, we simply see what percent of the vertices were *in truth* associated with class to which they got assigned by this process.

Thus, a perfect clustering would have a purity of 1.0. However, it also follows that as $k \to \infty$, the purity will tend towards 1.0 and will be exactly so once $k = n$. In other words, once there are sufficiently many clusters that each vertex gets associated to its own cluster, the purity there and for any higher values of k will be 1.0. Thus, this metric, while useful in ascertaining the overlap with the true classification, suffers from generally rewarding high granularity.

Another metric is the "normalized mutual information or NMI:

$$\text{NMI}(\Omega, \mathbb{C}) = \frac{I(\Omega; \mathbb{C})}{[H(\Omega) + H(\mathbb{C})]/2}$$

[where] I is mutual information

$$I(\Omega;\mathbb{C}) = \sum_k \sum_j P(\omega_k \cap c_j) \log \frac{P(\omega_k \cap c_j)}{P(\omega_k)P(c_j)} \tag{1}$$

$$= \sum_k \sum_j \frac{|\omega_k \cap c_j|}{N} \log \frac{N|\omega_k \cap c_j|}{|\omega_k||c_j|} \tag{2}$$

where $P(\omega_k)$, $P(c_j)$, and $P(\omega_k \cap c_j)$ are the probabilities of a document being in cluster ω_k, class c_j, and in the intersection of ω_k and c_j, respectively...H is entropy...:

$$H(\Omega) = -\sum_k P(\omega_k) \log P(\omega_k) \tag{3}$$

$$= -\sum_k \frac{|\omega_k|}{N} \log \frac{|\omega_k|}{N} \tag{4}$$

where...the second equation is based on maximum likelihood estimates of the probabilities" [15]. Having presented this metric, it serves well to parse the definition. The intuition behind this metric becomes more evident in understanding the notions of mutual information and entropy separately. In fact, the latter is simply a special case of the former, where $H(\Omega) = I(\Omega, \Omega)$. Returning to mutual information, this measures the gain in information from being presented a new piece of information. Intuitively, it is the reduction in uncertainty of the state of some random variable B by learning that for some r.v. A. In this particular case, NMI measures the extent to which knowing our guessed clusters helps in knowing the class. Clearly, in the case of a perfect reconstruction of the true clustering, knowing our guess will give complete knowledge of the associated class, whereas an ill-formed guess will give no bearing as to what the true class is.

It, once again, follows that the mutual information increases to its max value as $k \to \infty$, namely once there are sufficiently many clusters to assign one per vertex. Thus, if we are able to recreate the ground truth clusters perfectly, the maximum value of MI is achieved, namely MI = 1.0. However, by looking at the definition, it becomes clear that if these clusters are further subdivided, the resulting MI remains the same. That is to say, if we have one additional cluster than the number present in the ground truth but the clusters $C_1, \ldots, C_{n-1}$ correspond exactly to those in the truth and C_n, C_{n+1} split the underlying nth true class, the MI will still be = 1.0. In particular, this means in the case of each vertex being assigned to its own cluster, as this is a fully refined clustering of the ground truth, the max MI will *also* be attained in this case. Thus, to combat this issue, we normalize by

the entropy, which "tends to increase with the number of clusters" [15]. Thus, by combining these two into a single NMI metric, we are able to effectively measure that the correct number of clusters were extracted in addition to the contents of the clusters being correctly formed.

The final two metrics are directly related to one another, with the third simply being a special case of the fourth. This metric, known as the Rand index, is the simplest to present and interpret. We simply consider all $n(n-1)/2$ pairs of vertices and calculate:

$$RI = \frac{TP + TN}{TP + TN + FP + FN}$$

Where TN, TP, FP, and FN are respectively the true negative, true positive, false positive, and false negative classifications that were made. Thus, if a pair of vertices were clustered together when they were in the same class in reality, this would be a TP, whereas, if they were not in reality, this would be an FP. The TN and FN are similarly defined if a pair of vertices are clustered separately. Thus, this metric is simply the "total correct" of the total pairwise predictions made. Clearly, the main limitation in this formulation of a metric is that it weighs FN and FP errors equally. While this may occasionally be the case, it is not so for us, as discussed by the previous works in deanonymization, wherein even a small number of false positives could cause the collapse of the entire graph into a single supercluster. It would, therefore, seem quite natural to assume FN should be penalized less harshly as contrasted to FP in our case.

Extending to include this simple adjustment is quite straightforward, using the well-known F measure from statistics:

$$P = \frac{TP}{TP + FP} \tag{5}$$

$$R = \frac{TP}{TP + FN} \tag{6}$$

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \tag{7}$$

Where false positives would be weighted more heavily for small values of β, namely $\beta < 1$ [15]. Having discussed these metrics by which clustering can be evaluated, we turn to specific preprocessing techniques employed for the study herein due to the massive scale of the data at hand.

2.7 Matrix Estimation

The main goal for this section is to familiarize the reader with:

- The notion of matrix sketches, which seek to approximate matrices, in the same sense as that targeted in performing Principle Component Analysis, without computing SVDs. The true motivation for this is to understand the online clustering methods subsequently presented.

As mentioned previously, the scale of the BTC transaction graph was massive, which made the application of vanilla clustering algorithms practically impossible. Note that, while the heuristics graph contained information of a different nature than the transaction graph, it was roughly the same order of magnitude of scale. While each heuristic only applied to a subset of the entire transactions graph, the introduction of all the heuristics populated it with a comparable number of edges as the transactions graph. To get a sense of scale, the graph to be clustered had roughly 200 million vertices and 10 billion edges, roughly corresponding to 180 GB of disk space. Thus, in addition to requiring far too much time to execute, handling the data all at once would have required all this be stored simultaneously in memory. Thus, approaches that either handled execution of clustering in some online setting or an approximated form were necessary to explore. We discuss the former in this and the following section and the latter in the subsequent two sections of the background.

We now briefly discuss the basic concepts behind matrix estimation. Though the main concepts presented in this section do not directly come up in the investigation herein, they form the basis for some of the online clustering algorithms presented in the following section, making this information still relevant to understand. Clearly, "the optimal low rank approximation for any matrix is obtained by its truncated Singular Value Decompositions (SVD)," as this is precisely the result obtained in solving the corresponding Rayleigh Quotient problem [24]. It, therefore, may seem unnecessary for any other estimation methods to be developed, seeing as the *optimal* estimation for any arbitrary lower dimension is already known. However, the need becomes apparent in observing "Data matrices...[that are] extremely large and distributed across many machines...render standard SVD algorithms infeasible" [24].

Towards this end, our particular application has the former property, wherein the data is too massive to perform an SVD. Matrix sketching is an alternate approximation method where time is prioritized over accuracy. "There are three main matrix sketching approaches...The first generates a sparser version of the matrix...The second approach is to randomly combine matrix rows...The third

sketching approach is to find a small subset of matrix rows (or columns) that approximate the entire matrix" [24]. In other words, all methods of matrix sketching seek to construct some smaller matrix by directly manipulating the original matrix rows. Unlike the procedure of extracting vectors from SVD, there are no additional computations required herein with the exception of manipulations performed on the initial original matrix rows. The paper additionally "proposes a fourth approach. It draws on the similarity between the matrix sketching problem and the item frequency estimation problem" [24].

The idea revolves around extending a clever approximation algorithm that solves the seemingly unrelated problem of estimating the count of m distinct items as they appear in a data stream without requiring a counter per item. Specifically, their matrix sketching technique views the rows of a matrix as "items." To gain an intuitive understanding, consider the simplified case where the matrix *only* contains e_i vectors, i.e. the standard $\mathbb{R}^n$ basis vectors. In this case, we can apply this "frequency counter" idea to determine those that most frequently appear and, in turn, can construct a matrix of these basis vectors to approximate the original matrix.

The main issue in this arises in considering a matrix with rows of differing magnitudes and multiple non-zero entries. For example, the frequency counter idea falls apart when being applied to a matrix whose first row is $\mathbb{1}_n$ and second is $2 \cdot \mathbb{1}_n$, as the two would be viewed as completely distinct, when they both point in the same direction. It, therefore, seems more reasonable that the two should contribute to some common counter. Towards that end, the authors therein considered a refinement of the frequency counter they dubbed "frequent-directions," which seeks to solve this very issue. We defer to their paper for full presentation of detail, as this understanding suffices for our purposes.

2.8 Online Clustering

The main goals for this section are to familiarize the reader with:

- Online spectral clustering techniques, specifically streaming and evolutionary implementations, where the former simply involves updating clusters as new data points come in and the latter an extension thereof where weights too can be changed.
- Specific limitations thereof that made these **not** applicable to our context, specifically in the restriction to changing weights for streaming clustering and need for temporal smoothness in the latter.

Having presented this brief background on matrix approximation, we turn to the first method considered in performing clustering on the massive heuristic graph, namely that of performing online graph clustering. Of course, we use the term "online" in the traditional statistical sense, wherein the analysis is to be performed as data arrives rather than all at once, after the data has been fully collected. This has a natural correspondence to our setting, assuming Blocksci is used to procure the data incrementally. In viewing our problem as an online clustering problem, we would have new data at each time step that either (1) add new vertices/edges, when we encounter a node who is exchanging BTC for the first time or activity linking an existing vertex to another, or (2) changes of weights of previously present nodes, when a heuristic is marked true for a node that was previously marked false or vice versa. Thus, the online scheme employed had to be capable of handling both the addition and changing of previously seen data points, though handling of deletion was unnecessary, as a vertex would permanently remain in the graph after first being encountered.

All the online algorithms presented herein relate to spectral clustering, as much study has been conducted on that front and as we believed spectral clustering held the greatest prospects in revealing results missed in previous union-find studies. We discuss two variations thereof, namely the streaming and evolutionary versions of spectral clustering. Much work has been done in the realm of streaming clustering, as fully discussed in [46] and [13]. Having said that, the reason these algorithms fail to meet the demands of the task at hand is because none of them support the change of weights required for our purposes. That is to say, while they perform clustering on data streams, they assume such streams are simply adding new data points rather than updating those that already exist.

Towards that end, an alternate dubbed "evolutionary clustering" has arisen that seeks to fill this gap where weights can be newly created or changed from preexisting values. The main work in this space is [30], which states "our approach is the first work accomplishing the task of incremental spectral clustering that can handle not only insertion/removal of data points but also similarity changes...Evolutionary clustering simultaneously optimizes two potentially conflicting criteria, i.e., the clustering should fit the current data as much as possible, while should not deviate dramatically from the historic context." This, therefore, fits our setting quite well, with one exception, namely that of temporally smoothness. While it is generally the case that temporal smoothness is a desirable property, especially where the data itself is modelling a process that itself is evolving in nature, this context does not fit that description. Specifically, the act of two vertices corresponding to the

same person is not a continuous process; it is simply a binary fact about which we have information arriving incrementally. In other words, it is very possible, and almost certainly the case, that up until some transaction t, vertices v_A, v_B will be thought to correspond to two different clusters/people but transaction t may reveal v_A, v_B both as being concurrent inputs, meaning (by Heuristic 1 previously discussed) we would want to *always* consider these two as being in the same cluster thereafter. In other words, there would be a discontinuous jump from **not** being in the same cluster to being clustered together at time t. From this property baked directly into the algorithm, its applicability to our context was largely removed. We, however, include the brief remainder of the literature review for sake of researchers who wish to further investigate this approach to the problem.

The algorithm, therefore, relies on starting with an eigenvector of L from the first data points of the stream and subsequently incrementally updating the eigenvalues and eigenvectors of the Laplacian with each new data point streamed in. We specifically refer to the results and algorithms catered in [30], which found:

$$\Delta v_{ij} = (K_{\mathcal{N}_{ij}}^T K_{\mathcal{N}_{ij}})^{-1} K_{\mathcal{N}_{ij}} \mathbf{h} \tag{8}$$

Where (for an eigenvector $\mathbf{v}$ and some fixed threshold τ):

$$K = L - \lambda D \tag{9}$$

$$\mathbf{h} = (\Delta\lambda D + \lambda \Delta D - \Delta L)\mathbf{v} \tag{10}$$

$$\mathcal{N}_{ij} = \{k | w_{ik} > \tau \text{ or } w_{jk} > \tau\} \tag{11}$$

$$\Delta\lambda = \Delta w_{ij} \frac{a+b}{1+c+d} \tag{12}$$

Where (v_i denoting the ith component of v and d_k is degree of vertex k):

$$a = (v_i - v_j)^2 - \lambda(v_i^2 + v_j^2) \tag{13}$$

$$b = (v_i - v_j)(\Delta v_i - \Delta v_j) - \lambda(v_i \Delta v_i + v_j \Delta v_j) \tag{14}$$

$$c = \Delta w_{ij}(v_i^2 + v_j^2) \tag{15}$$

$$d = \sum_{k \in \mathcal{N}_{ij}} q_k d_k \Delta q_k \tag{16}$$

Algorithm 4 Refinement of $\Delta\lambda$ and Δv [30]

procedure REFINE($\lambda, v, D, \Delta D, L, \Delta L, \Delta W$)
Input: Current eigenvalue λ
Input: Current associated eigenvector v
Input: Previous time-step degree matrix D
Input: Time-step degree matrix update ΔD
Input: Previous time-step Laplacian matrix L
Input: Time-step Laplacian matrix update ΔL
Input: Time-step weight matrix update ΔW
    $\Delta v \leftarrow 0$
    while $\Delta v, \Delta\lambda$ not stabilized **do**
        $\Delta\lambda \leftarrow$ Eqn. (12) using current Δv
        $\Delta v \leftarrow$ Eqn. (8) using current $\Delta\lambda$
    Output: $\Delta v, \Delta\lambda$

Algorithm 5 Iterative (online) spectral clustering [30]

procedure SPECTRALCLUSTERING-ITERATIVE(S_D,K)
Input: Data stream (individual vertices with heuristics) S_D
Input: Cluster count k
    $t \leftarrow$ time enough data points to define W, L, D
    $G \leftarrow \emptyset$
    for $i = 1 : t$ **do**
        $G \leftarrow S_D[i]$update vertices/edges
    $W \leftarrow W(G), L \leftarrow L(G), D \leftarrow D(G)$
    $[v_i], [\lambda_i] \leftarrow eigendecomp(L)$
    while $S_D[j]$not null **do**
        $i \leftarrow \{v_i | S_D[j]_i > 0\}$
        $\Delta W \leftarrow W - S_D[j]e_j$
        $\Delta D \leftarrow D - S_D[j]e_j$
        $\Delta L \leftarrow \Delta W + \Delta D$
        $\Delta v_i, \Delta\lambda_i \leftarrow Refine(\lambda_i, v_i, D, \Delta D, L, \Delta L, \Delta W)$
        $Update(W, D, L, v_i, \lambda_i)$
    $V \in \mathbb{R}^{n\times k} \leftarrow [v_1^T, v_2^T, \ldots, v_k^T]$
    $(y_i)_{i=[1:k]} \leftarrow V_{i,:}^T$ (i.e. row i of V)
    $(C_i)_{i=[1:k]} \leftarrow kmeans((y_i)_{i=[1:k]})$
Output: C

Thus, all investigated forms of online spectral clustering have some issue in being applied to the context of BTC wallet deanonymization. To be explicit, the streaming clustering techniques never handle the case of changing weights, which is hugely important in the changing evaluations of heuristics with new pieces of data. Similarly, the evolutionary techniques fundamentally revolve around temporal smoothness, which conflicts with the nature of the clustering on the heuristics graph. Future investigations may seek to further the work in either of these algorithms to rectify these shortcomings to allow their application to the context of deanonymization.

2.9 Graph Sparsification

The main goal for this section is to familiarize the reader with:

- The existence of graph sparsification algorithms whose main objective is to remove edges of the original graph while retaining a Laplacian matrix as close to that of the original as possible.

The second set of methods considered sought to deal with the scale of the graph by approximating the overall graph. Namely, rather than performing the clustering on the original heuristic graph G, we would employ preprocessing methods to procure some related graph G', on which clustering could be performed much more efficiently while additionally providing a natural method of extending back to the original G. Towards that end, we looked to both graph sparsification and coarsening; both seek to produce a G' that is both simpler than and similar to the original G. Sparsification seeks to produce a simpler graph by removing edges of the original graph, whereas coarsening seeks to do so by removing vertices.

As we discussed in the previous sections, namely in those on spectral clustering, information of the separability of clusters is embedded in the spectrum of the corresponding Laplacian matrix. It, therefore, seems reasonable to perform graph approximation with the express goal of retaining a graph whose Laplacian matrix is as close to that of the original as possible. What is meant by "close" is once again ambiguous. Towards that end, we use what has become the standard, namely the notion two matrices being $\sigma-$spectrally similar. Prior to defining this notion, we introduce some associated notation, namely:

$$A \preceq B \implies x^T Ax \leq x^T Bx \; \forall x$$

Having introduced that, "We say A and B are $\sigma-$spectrally similar if:

$$B/\sigma \preceq A \preceq \sigma \cdot B\text{" [3]}$$

.

It is not immediately obvious how this notion of two graphs being $\sigma-$spectrally similar, which is defined by simply seeing whether their associated Laplacian matrices are $\sigma-$spectrally similar, can be interpreted. We, therefore, discuss an equivalent formulation thereof that directly resolves this disconnect. Specifically, the "Courant-Fisher Theorem tells us that:

$$\lambda_i(A) = \max_{S:\dim(S)=i} \min_{x \in S} \frac{x^T A x}{x^T x}$$

Thus, if $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of A and $\widetilde{\lambda_1}, \ldots, \widetilde{\lambda_n}$ are the eigenvalues of B, then for all i, $\lambda_i/\sigma \leq \widetilde{\lambda}_i \leq \sigma \cdot \lambda_i$" [3]. Thus, any sparsification that retains $\sigma-$spectral similarity is greatly of interest for the purposes of obtaining a clustering that can directly applied to the original graph. Towards that end, a method dubbed spectral sparsification was procured that seeks to sparsify edges and produce a $\sigma-$spectrally similar graph, with σ tightly bounded from above. This process "involves assigning a probability $p_{u,v}$ to each edge $(u, v) \in G$ and then selecting edge (u, v) to be in the graph $\widetilde{G}$ with probability $p_{u,v}$. When edge (u, v) is chosen to be in the graph, we multiply its weight by $1/p_{u,v}$. This procedure guarantees that:

$$\mathbb{E}[L_{\widetilde{G}}] = L_G"[3]$$

In fact, the only step of this process that needs further detailing is the assignment of such probabilities $p_{u,v}$. Of course, assigning a lower probability will result in an overall sparser graph, yet doing so also loses information. Thus, the choice of $p_{u,v}$ represents balancing the desire to produce a simpler graph against that of producing one that contains most of the information that was present in the original graph. For this fact, however, a theorem was demonstrated in this same paper:

Theorem 2.5. *Suppose $\epsilon \in (0, 1/2)$ and $G = (V, E)$ is an unweighted graph with smallest non-zero normalized Laplacian eigenvalue at least λ. Let $\widetilde{G} = (V, \widetilde{E}, \widetilde{w})$ be a graph obtained by sampling the edges of G with probabilities:*

$$p_e = \min(1, C/\min(d_u, d_v))$$

for each edge $e = (u, v)$, where:

$$C = \Theta((\log(n))^2(\epsilon\lambda)^{-2})$$

and setting weights $\widetilde{w}_{(e)} = 1/p_e$ for $e \in \widetilde{E}$. Then, with probability at least $1/2$, $\widetilde{G}$ is a $(1+\epsilon)$-spectral approximation of G, and the average degree of $\widetilde{G}$ is $O((\log(n))^2(\epsilon\lambda)^{-2})$ [3]

Thus, using this theorem, we could employ spectral sparsification to the graphs at hand to obtain a graph that was simpler to deal with for clustering. The sparsification of this matrix substantially

reduces the memory consumption of the model, since there are much more efficient methods by which sparse matrices can be represented in most programming paradigms than the naive 2D array representations.

2.10 Graph Coarsening

The main goals for this section are to familiarize the reader with:

- The notion of graph coarsening as a preprocessing step for clustering, which parallels sparsification except revolves around removing graph vertices rather than edges.
- METIS, a developed algorithm that relies on a repeated series of coarsenings to reduce the original graph of interest to a reasonable size before performing clustering and projecting back to the original space with correction steps.

In addition to dealing with space issues, developing a time-efficient means to cluster this data was of great importance. By directly reducing the count of vertices, graph coarsening produces a smaller graph, potentially making certain procedures feasible to run, such as eigendecomposition. Graph coarsening, unlike graph sparsification, seeks to condense information into a single representation rather than removing unnecessary data outright. In other words, rather than simply removing vertices from a graph G outright to produce $\widetilde{G}$, the process, alternatively referred to as edge contraction, coalesces two vertices u, v such that any edges incident to either vertex is now incident to this combined vertex. In practice, this procedure is performed on weighted graphs, for which the collection of two edges into a single edge involves adding the weights for the initial two edges to find the resultant weight. Despite being a straightforward procedure, there exist two main variations on this process, referred to as SAG and WAG, respectively depicted in Figures 10 and 11. Roughly, the former corresponds to aggregating vertices by a strictly enforced threshold value, i.e. coalescing vertices i, j if "w_{ij} is comparable to $\min\{\max_k(w_{ik}, \max_k(w_{kj})\}$" [6]. The latter corresponds to "express[ing] the likelihood of nodes belong[ing] together; these likelihoods then accumulate at the coarser levels of the process" [6].

Having discussed the specifics of the coarsening itself, we consider algorithms that specifically employ this technique. The main one that falls into this category is dubbed METIS, whose overall structure is as follows: "The graph G is first coarsened down to a few hundred vertices, a bisection of this much smaller graph is computed, and then this partition is projected back towards the

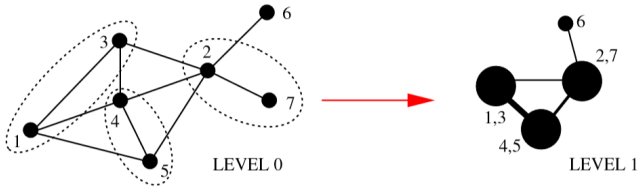

Figure 10: SAG (strict aggregation) assumes a strict threshold per which it is determined whether two vertices should be coalesced or not [6].

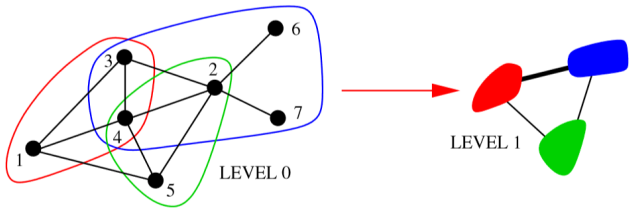

Figure 11: WAG (weighted aggregation), in contrast to SAG, determines the *likelihood* for different vertices to be clustered together, whereby vertices can be iteratively aggregated with further levels of coarsening [6].

original graph (finer graph), by periodically refining the partition" [20]. In line with that, there are three main facets of this algorithm that necessitate discussion, specifically the coarsening, eventual clustering, and refinement of partitions during uncoarsening.

To be precise, the objective of the coarsening phase is to construct a sequence of graphs $\{G_l\}$ such that $|V_l| > |V_{l+1}|$, where we denote the original graph $G_0 = (V_0, E_0)$ [20]. In line with this, "G_{l+1} is constructed from G_l by finding a maximal matching $M_l \subset E_l$ of G_l and collapsing together the vertices that are incident on each edge of the matching. In this process no more than two vertices are collapsed together because a matching of a graph is a set of edges, no two of which are incident on the same vertex" [20]. In line with the difference between SAG and WAG, there were multiple procedures discussed by which the maximal matching could be constructed, though it was found that the HEM "heavy-edge matching..., [which] computes a matching M_l, such that the weight of the edges in M_l is high...produces consistently better results than RM [random matching], and the amount of time spent in refinement is less than that of RM" [20].

Having discussed the coarsening, we turn to step two of METIS, namely the clustering, referred to in Figure 12 as the "initial partitioning phase." Specifically, the algorithm supports "four different schemes for partitioning the coarsest graph...Three of these algorithms are based on graph growing heuristics, and the other one is based on spectral bisection" [20]. For the purposes of this

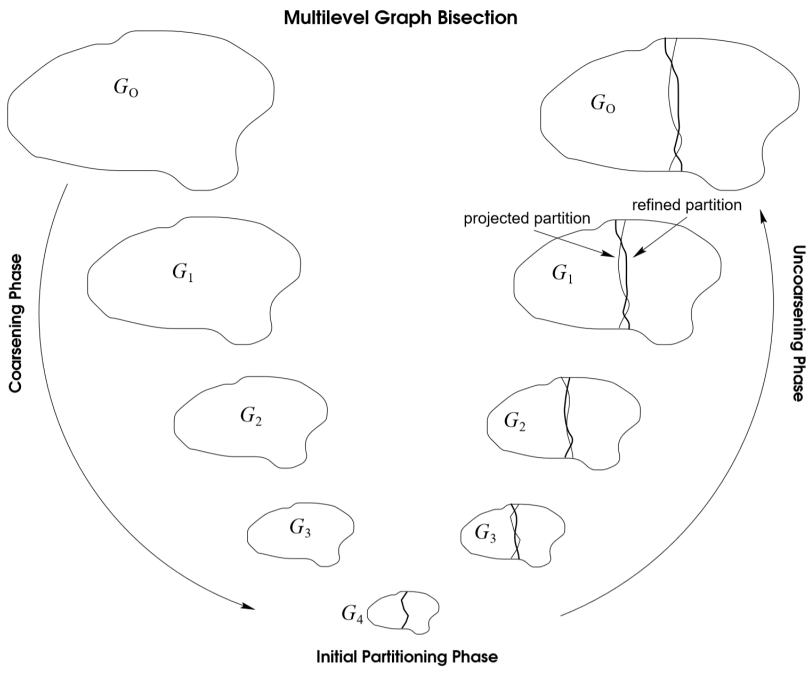

Figure 12: METIS relies on continual coarsening of the graph to a point where bisection or clustering is a relatively trivial problem and subsequently refining this partition as the graph is expanded back to its original size [20].

investigation, we decided to make use of spectral bisection, as this allowed for applying the results of experiments we conducted on the SBM initially.

From here arises the main spectacle of the METIS algorithm: the uncoarsening phase. Formally, this involves taking the partitioning sets $\{C_i^k\}$ obtained in the coarsest level that satisfy $\bigcup_i C_i^k = V_k$ and projecting them up through $G_{k-1}, G_{k-2}, ..., G_1, G_0$, such that each $\{C_i^l\}$ satisfies $\bigcup_i C_i^l = V_l$. In other words, taking the partitioning of the coarsest graph and attempting to efficiently construct a partitioning that eventually contains all the vertices of the original graph.

Towards that end, we can simply discuss how such a projection happens in a single step of the uncoarsening, from which the full process is fully defined. Specifically, a single step in this process begins with the natural projection up one level. That is to say, if two vertices u, v were coalesced into a supernode uv and said vertex uv was clustered into class A, the natural projection simply assigns u, v to A. However, "even if the partition of G_l is at a local minima, the projected partition of G_{l-1} may not be at a local minima. Since G_{l-1} is finer, it has more degrees of freedom that can be used to further improve the partition and thus decrease the edge-cut. Hence, it may still be possible to improve the projected partition of G_{l-1} by local refinement heuristics" [20]. In other

words, while it is the case that the clustering in the coarser level minimizes the crossing edges, this property is not necessarily retained in projecting up a level, due to the increased degrees of freedom. A "refinement heuristic" here refers to one that takes "two parts of the bisection $[A, B$ and]...selects $A' \subset A$ and $B' \subset B$ such that $A \backslash A' \cup B'$ and $B \backslash B' \cup A'$ is a bisection with a smaller edge-cut" [20]. The specific algorithm employed for this purpose was the Kernighan-Lin (KL) partition algorithm, though its details are not necessary to understand for our purposes.

In this way, METIS is capable of performing partitioning on enormous graphs, namely by creating a mapping from a large graph to a significantly reduced graph and one from this reduced graph's clustering to that of the original graph.

2.11 Notation

Prior to delving into the experiments themselves, we present a comprehensive table of the notation introduced in the background sections for the purpose of serving as reference for the remainder of the paper.

Table 2: Notation/abbreviations as used in this paper unless otherwise noted

Symbol	Interpretation
$G = (V, E)$	Graph with vertices V and edges E
D	Graph degree matrix, i.e. degree of vertex i in position D_{ii} and 0 elsewhere
W	Graph weighted edge matrix
L	Graph Laplacian, defined as $L = D - W$
λ_i	ith eigenvalue, when sorted in *increasing* order, i.e. the ith smallest eigenvalue
v_i	ith eigenvector, i.e. the eigenvector associated with λ_i
n	Number of nodes in the graph, i.e. $n = \|V\|$
$\{C_i\}$	Graph clusters, such that $\bigcup_i C_i = V$, $\bigcap_i C_i = \emptyset$, $C_i \subset V$
e_i	Standard ith basis vector of $\mathbb{R}^n$, i.e. $[0, 0, \ldots, 1, 0, \ldots, 0]$ (in position i)
SBM	Stochastic block model (planted partition model)
p, q	In- and out-cluster connection probabilities in SBM
NMI	Normalized mutual information
H	Entropy (probability/information theory)
F	Standard statistical F-score

3 Methodology

Having finished discussing the main relevant background, we turn to discussing the novel experiments/results conducted for this particular investigation. As was previously described, many of the results catered herein were performed on the SBM, from which relevant trends became apparent and their applicability to the full dataset more evident. Towards that end, the experiments are described here along with their intended purpose, and the associated results are presented in the corresponding subsection of Section 4.

3.1 SBM Clustering Trial

Given that the objective of this investigation was to produce a clustering on the full-scale heuristics graph, the main goal of these lead-up experiments was to ascertain which of the clustering techniques discussed in the background would be best suited for our purposes. Thus, this first experiment, as with all those that follow, involved taking three accuracy metrics and assigning a cumulative score to each technique. Specifically, the three metrics were those defined in the Section 2.6, namely the purity, normalized mutual information, and F-score (weighted rand index). Of course, the cumulative score was proportional to each of the three accuracy metrics, though weights were assigned on the basis of what was of greatest importance for our purposes. Thus, the final "evaluation score" is:

$$score = \frac{w_{purity}purity + w_{NMI}NMI + w_F F}{w_{purity} + w_{NMI} + w_F} \tag{17}$$

The specific algorithms that were tested against one another in this first set of experiments were all the offline or out-of-box (i.e. previously implemented) algorithms discussed, namely: hierarchicial spectral, k-means spectral, spectral (`scikit-learn`), k-means (`scikit-learn`), DBSCAN (`scikit-learn`), and METIS. Thus, the outputs of this section include the following:

1. **Metric plots:** For each of the three metrics in addition to the cumulative score, plots of performance by the different clustering algorithms for fixed values of p (as defined in the SBM) over varying levels of q.

2. **Time plot:** Each of the clustering algorithms were run on fixed choices of p, q while varying the size of the graphs, for which the time for completion of the clustering are plotted.

3. **Sample clustering:** The graphs themselves, after clustering was performed, are provided for sake of understanding the nature of the different clustering results.

Thus, it was the intention of this section to have a clear choice for an algorithm to be used for clustering on the final heuristics graph.

3.2 SBM Sparsification Trial

In addition to the offline algorithms, there were the various potential preprocessing techniques discussed in the background section whose effectivenesses were of interest, with respect to reducing the time expended while retaining reasonable accuracy. Amongst these was sparsification, which we tested on the two manually implemented spectral clustering algorithms against various accuracy metrics on the SBM. It was expected that sparsification would result in a slight degradation of accuracy, though ideally non-significant, and a decrease in computation time.

3.3 SBM Coarsening Trial

In addition to sparsification, preprocessing by coarsening was of interest, namely in seeing what time could be saved without causing any significant loss in accuracy, much in the same manner sparsification was tested. This coarsening trial, however, was somewhat moot, as METIS is itself a coarsening algorithm at heart. We, therefore, anticipated that performance would be of lower quality than METIS, though its application to other algorithms could be more clearly recognized as advisable or not by the results of these tests.

3.4 Full Dataset Trial

With the final determined algorithm, trials were conducted on a partial subset of the full dataset, with emphasis being placed on its performance measured in terms of time and sensibility of results. Unfortunately, no address was clustered that was known in the ground truth catered in "A Fistful of Bitcoins," as elaborated upon in the corresponding results section.

4 Results

We present the figures and tables as elaborated in the previous section in the following subsections. This section was the result of running the code available at `https://github.com/yashpatel5400/anonychain`, as described more fully in that repo (written in Python 3). The relevant discussion, i.e. interpretation of the figures, are given in the following section, namely Section 5.

For specifics on how the trials were run, we provide the summary below:

- **SBM Clustering:** For multiple values of p, i.e. holding p fixed, we chose a corresponding set of values of q to run trials on. For each pair (p, q), five identical trials were run. Each trial produced an SBM using these values of p, q. From here, each of clustering algorithms were run and their accuracies, across all metrics, and times were measured. After the five trials were run, an average score was computed, thus leaving a single data point per metric corresponding to the pair (p, q) across the various algorithms. After being computed over all relevant values of p, q, the values were plotted. Note that, across all the trials, the following were the cluster sizes used: `[63,53,32,25,20,18,14,12,8,7,7,5,4,3,3,2]`. These somewhat arbitrary sizes were chosen, as they roughly correspond to the heuristics graph nature, where a small subset of the graph has a large portion of the wallets (i.e. Coinbase and Mt. Gox) and the remainder have a reasonable number. In addition, the overall score was computed with $w_{purity} = 1.0, w_{NMI} = 3.0, w_F = 2.0$. These particular values of weights were chosen as NMI most closely captures the notion of determining the "overlap" with the underlying clusters while also penalizing separating vertices into their own compartments.

- For each of the accuracy metrics, DBSCAN's outliers (i.e. those vertices that were not associated with a particular class) were handled by simply considering them as belonging to a class that did not exist in the truth. That is to say, any "intersection" between a true class and the outliers would be empty. Additionally, in iterating through the clusters, as is the case of calculating purity, the outliers were simply ignored.

- For clarity, the labels as associated with the various clustering algorithms are laid out below:
  - **KMeans**: Standard k-means algorithm, as implemented in `scikit-learn`
  - **ManualHierarchical**: A manually implemented hierarchical spectral clustering algorithm, i.e. based on the principle of spectral bisection

- **ManualKMeans**: A manually implemented k-means spectral clustering algorithm, i.e. one which performs the final clustering on graph in a single iteration
- **METIS**: METIS algorithm as described in Section 2.10
- **MiniBatchKMeans**: Mini-batch k-means algorithm, a modified implementation of the k-means algorithm, which seeks "to reduce the computation time, while still attempting to optimise the same objective function. Mini-batches are subsets of the input data, randomly sampled in each training iteration. These mini-batches drastically reduce the amount of computation required to converge to a local solution. In contrast to other algorithms that reduce the convergence time of k-means, mini-batch k-means produces results that are generally only slightly worse than the standard algorithm" [9]
- **SpectralClustering**: Spectral clustering algorithm, as implemented in `scikit-learn`

- Coarsening was implemented much in the way it is in METIS. That is to say, rather than arbitrarily selecting edges, a matching of maximal weight was determined and used for the purposes of edge contraction. From there, the clustering was performed, and clustering results expanded to the original graph. As the purpose of this particular subset of experimentation was to tease out information regarding the loss in accuracy relayed by clumping nodes, no additional "refinement algorithm" was implemented, as is done in METIS. Instead, we simply expand the projected partition to the original graph as our final result. In addition, performing multiple levels of matching simply involved repeated edge contraction, clustering, and a direct expansion to the original graph, rather than the stepwise uncoarsening process employed by METIS.

- Coarsening was *not* extended to DBSCAN, since there was no straightforward or canonical way to extend the clusterings of outliers to the original graph. Of course, while there is the completely straightforward notion of associating the outliers of the coarsened graph to be outliers of the original graph, this makes little sense in terms of the algorithm, as it is quite likely all the original vertices condensed into a single vertex would be in the same neighborhood as one another, inherently altering the result of the DBSCAN output. Hence, rather than attempting to understand how the notion of outliers would extend in the process of uncoarsening a graph, this issue was simply sidestepped.

- Any visualization of the graphs/network structure was completed using the standard Fruchterman-

Reingold force-directed algorithm, as implemented by the Python NetworkX package. Briefly, the algorithm revolves around associating between each pair of vertices a "spring" which pulls them in accordance to their connection (edge) weight together against a repulsive force experienced mutually between all vertices [41].

- Unlike the previously described trial averaging method, the structure for the sparsification trials was simply varying the ε parameter while fixing a value of (p, q) and conducting a single trial per parameter set. This was specifically done as, unlike accuracy metrics, which vary significantly on the basis of random variations in the formation of the SBM graph, the time necessary to execute an algorithm remains relatively constant on the basis of the graph size. While there are variations associated with random initializations of some algorithms, such as k-means, these were not in the sparsification section, rendering this a non-issue.

4.1 SBM Clustering Results

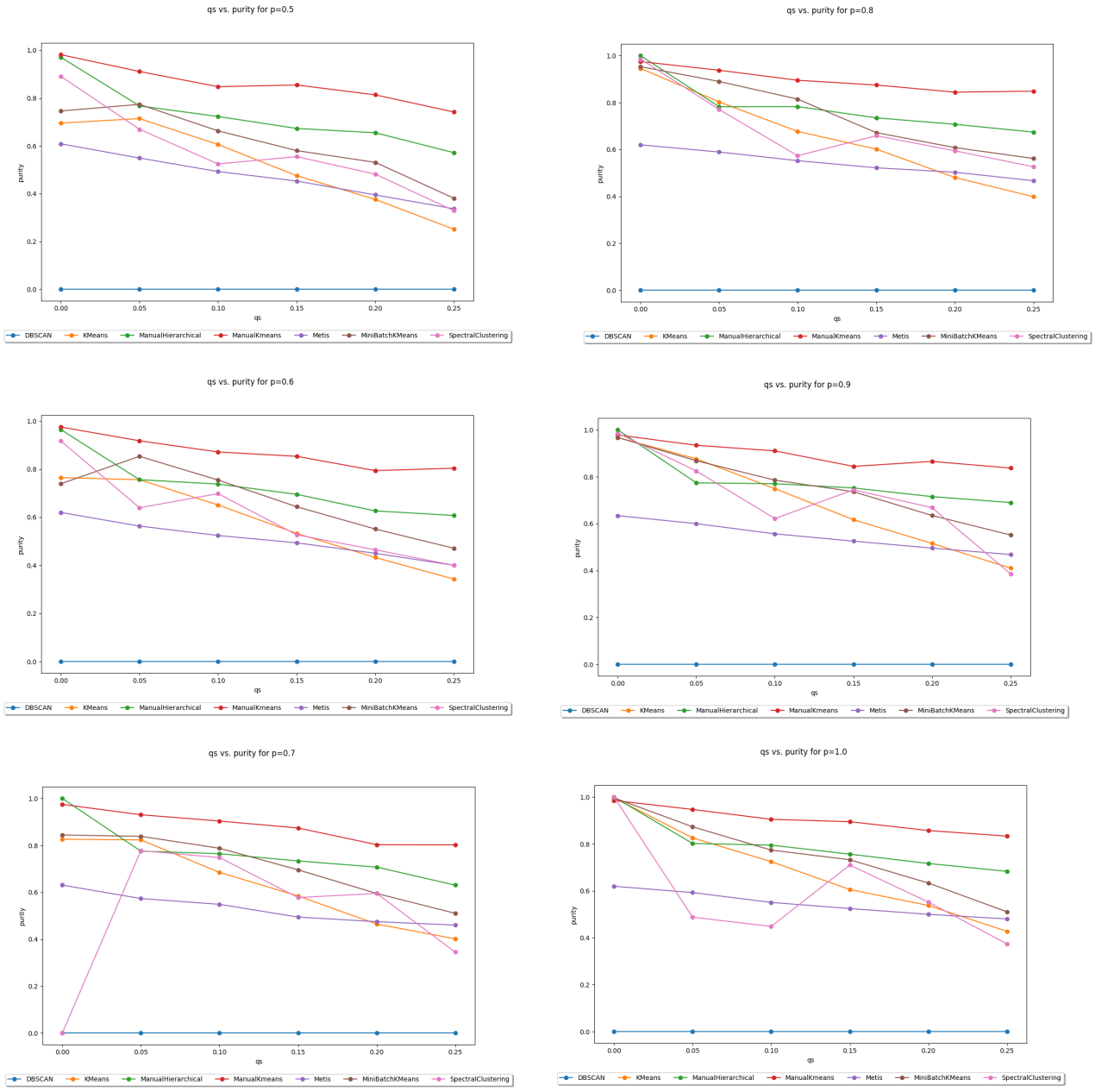

Figure 13: Results of running the clustering algorithms for the purity metric. The p values go from $p = 0.5$ to $p = 1.0$ in increments of 0.1, increasing down the columns. As a reminder, purity varies from 0.0 to 1.0, with 0.0 corresponding to there being many vertices of *different* classes being clustered together and 1.0 where only vertices of the same classes were clustered together. Thus, a perfect clustering would have a purity of 1.0.

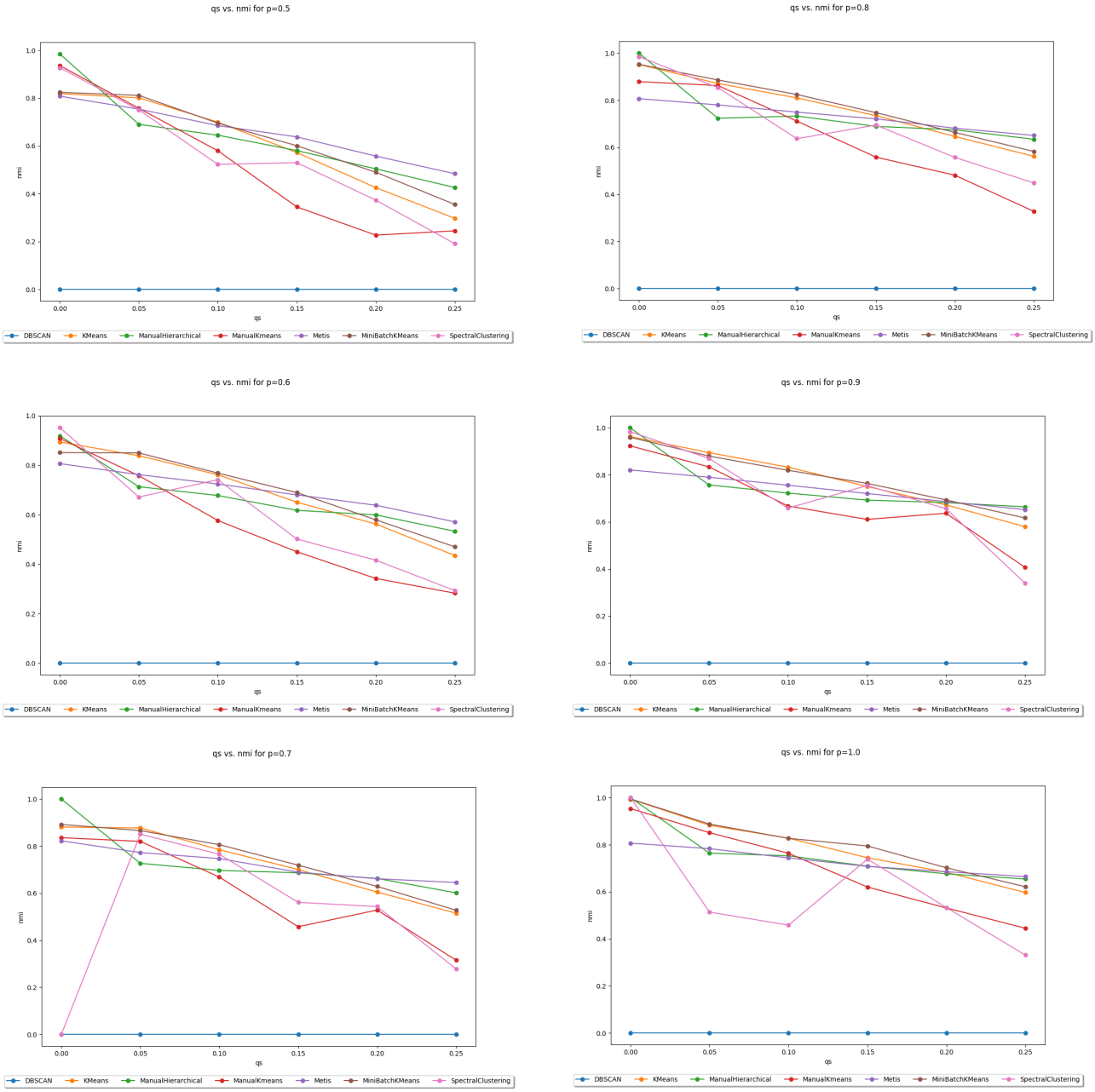

Figure 14: Results of running the clustering algorithms for the NMI metric. The p values go from $p = 0.5$ to $p = 1.0$ in increments of 0.1, increasing down the columns. As a reminder, NMI varies from 0.0 to 1.0, with 0.0 corresponding to where the clustering of a vertex gives *no* indication of what its true class is and 1.0 where the cluster provides perfect information thereof. Thus, a perfect clustering would have an NMI of 1.0.

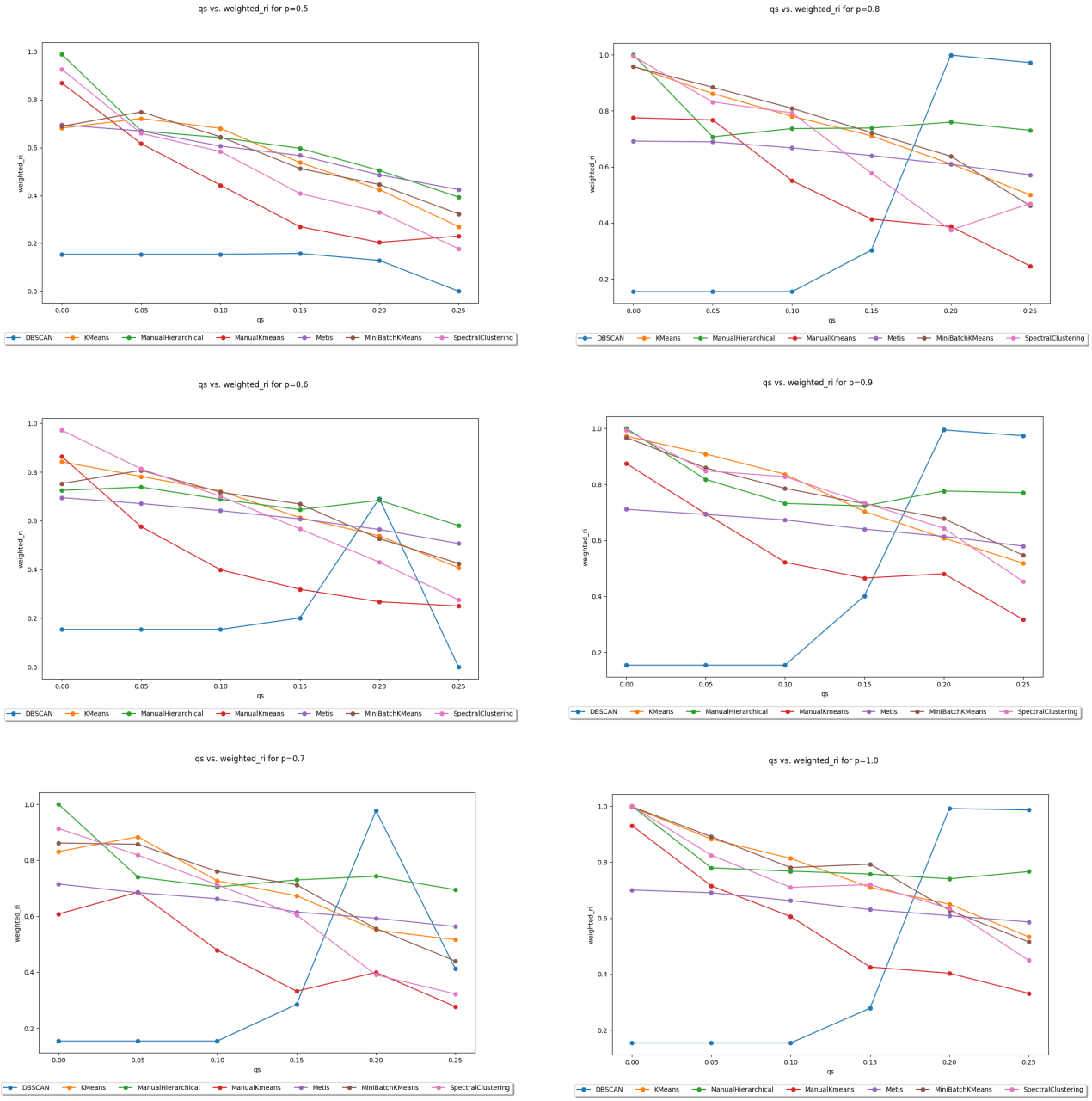

Figure 15: Results of running the clustering algorithms for the F-score metric with $\beta = 0.5$. The p values go from $p = 0.5$ to $p = 1.0$ in increments of 0.1, increasing down the columns. As a reminder, the F-score varies from 0.0 to 1.0, with 1.0 corresponding to where all *pairwise* predictions of clusters were correct. That is, all pairs of vertices (i, j) were predicted to be in the same cluster when indeed they were in the same class, or they were predicted to be in different clusters similarly when that was the case.

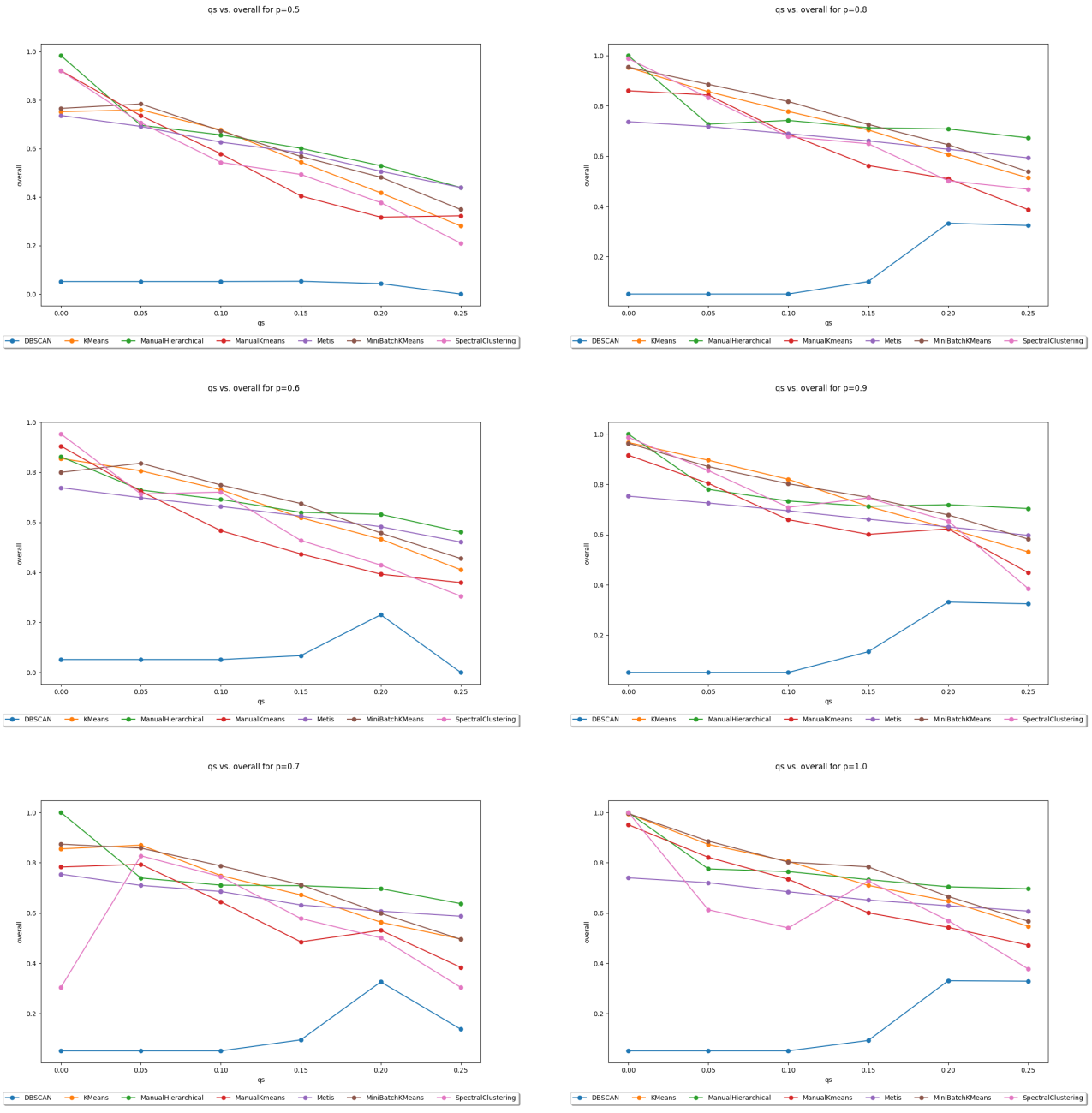

Figure 16: Results of running the clustering algorithms for the overall score metric, as defined by Eqn. (17), where we specifically elected to use $w_{purity} = 1.0, w_{NMI} = 3.0, w_F = 2.0$. The p values go from $p = 0.5$ to $p = 1.0$ in increments of 0.1, increasing down the columns.

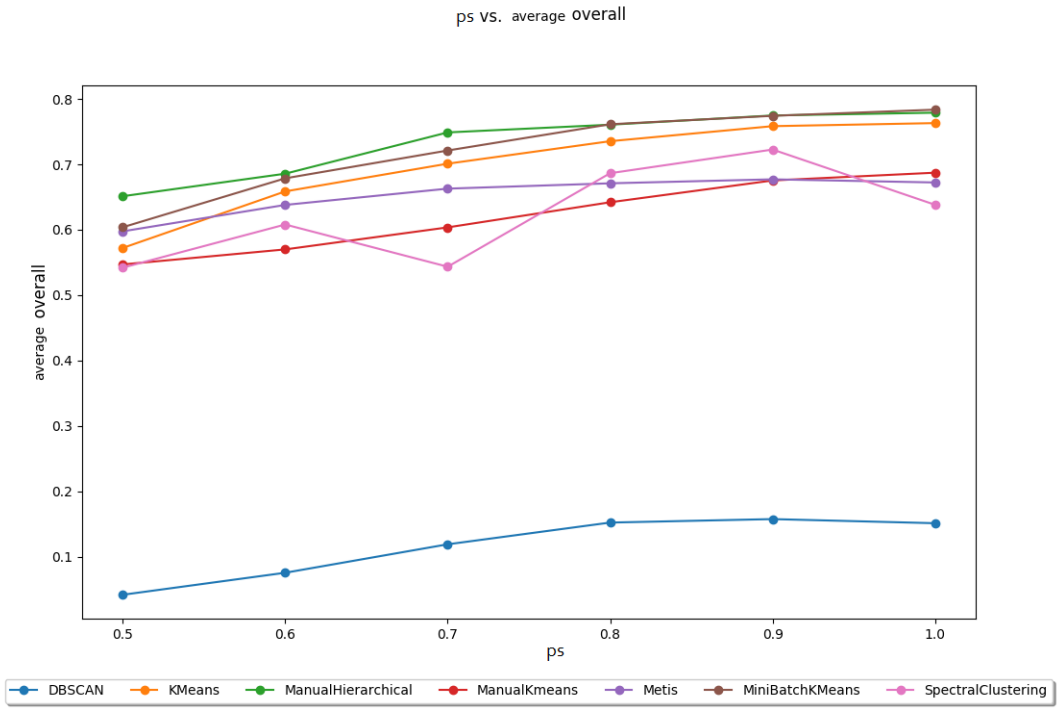

Figure 17: For each of the p trials in Figure 16, we averaged the overall scores across the different values of q, plotted here.

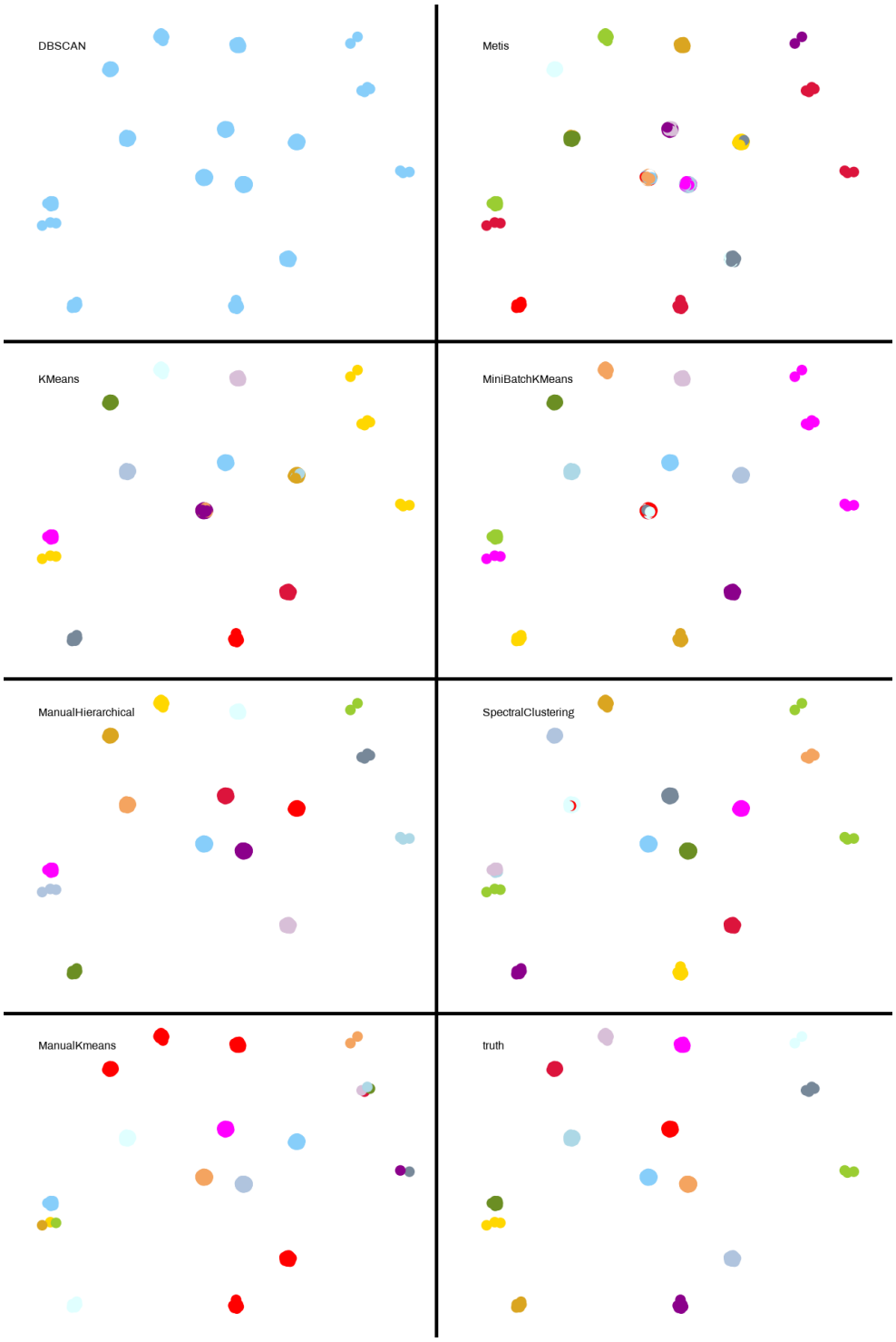

Figure 18: Clustering results for $p = 0.80, q = 0.0$, where the performed clustering is annotated in the top-left corner and the ground truth provided in the bottom-right. Clearly, while the colors within a particular clustering are consistent, it means nothing to compare these colors across clusterings as labelling one group A and another B is wholly arbitrary and just as valid vice versa. Of note is that most algorithms performed quite well here, namely in the circumstance where the clusters are well-separated (i.e. $q = 0$).

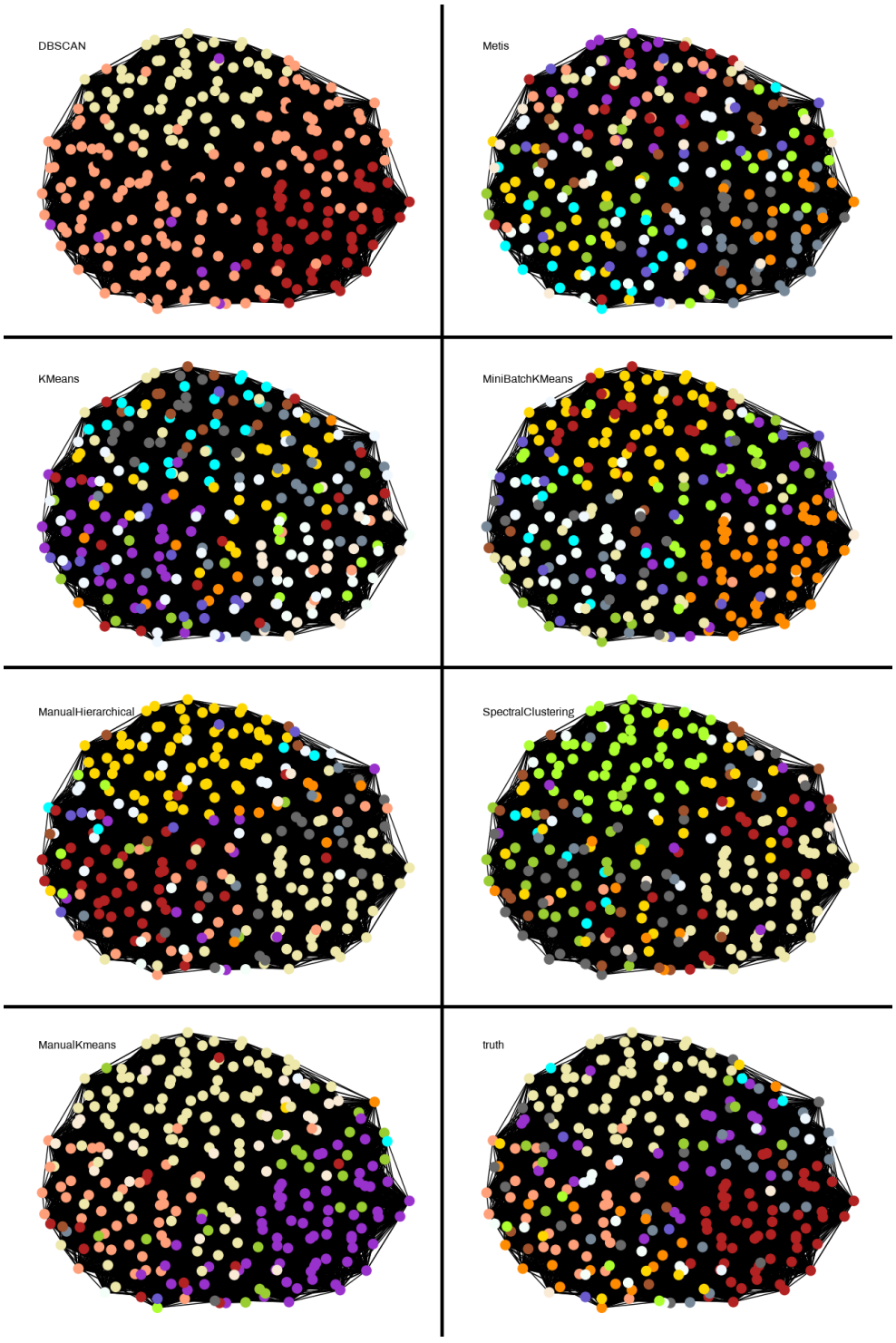

Figure 19: Results for $p = 0.90, q = 0.15$. Unfortunately, interpreting these results is more difficult than the previous due to the excess of edges present and lack of a natural method of laying out the vertices on a 2D plane. While not completely obvious, there are many black vertices in the illustration of the DBSCAN result, which correspond to the outliers as defined and explained about the algorithm (i.e. vertices not assigned to a particular class).

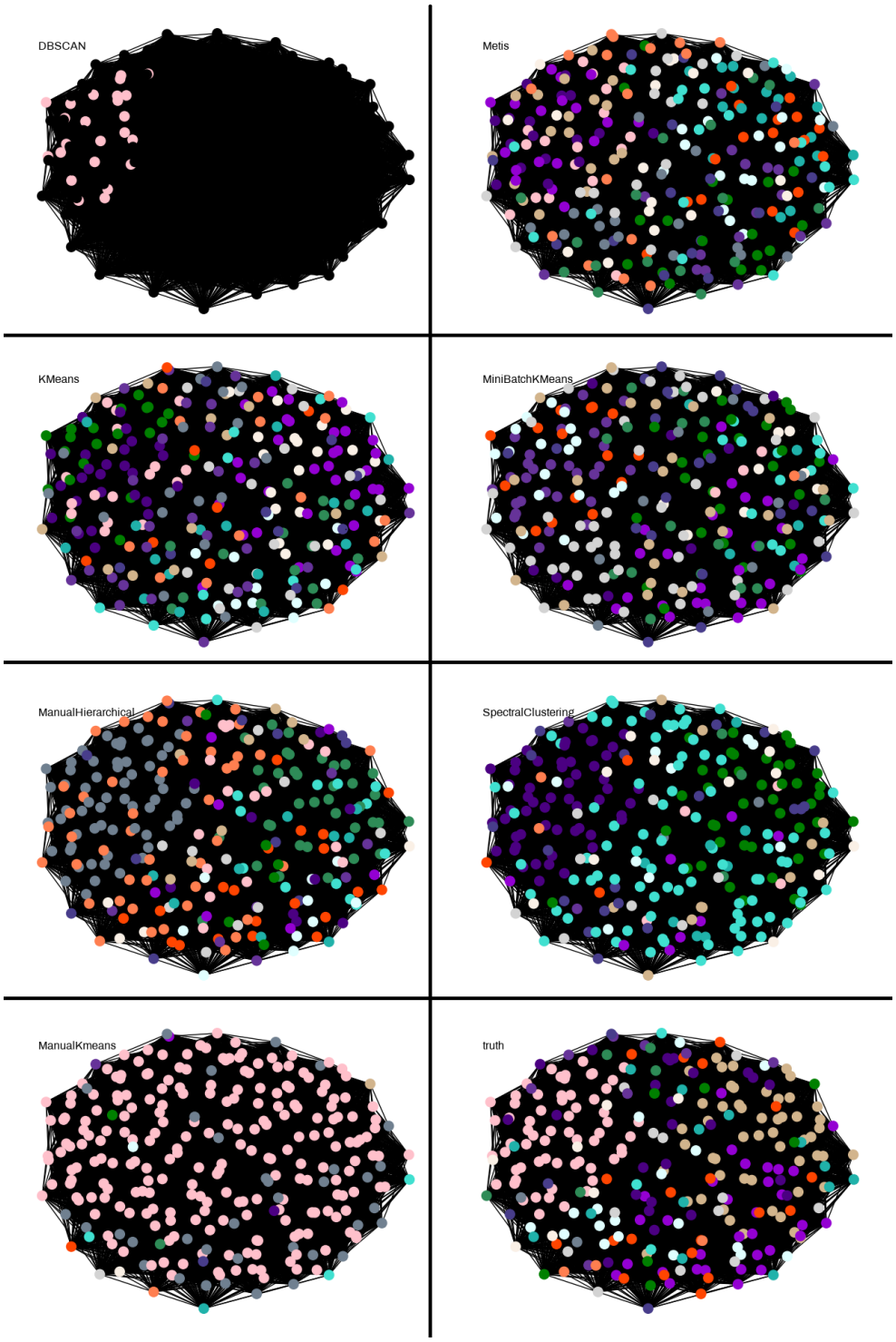

Figure 20: Results for $p = 0.60, q = 0.20$. This marks a very similar case to that illustrated in Figure 19. Additionally, the black outlier vertices referenced in Figure 19 are much more evident in this illustration.

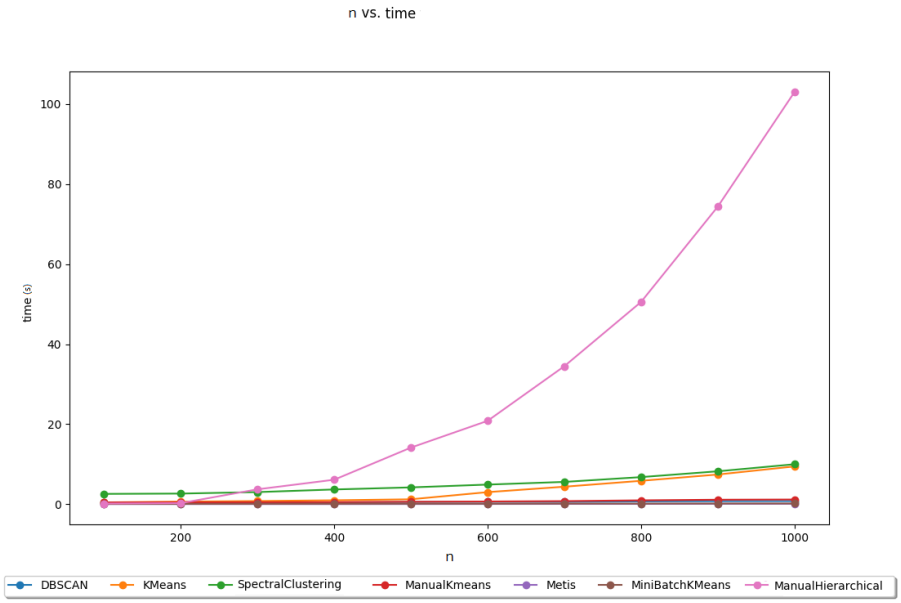

Figure 21: Times for $p = 0.90, q = 0.15$ for various values of n. Specific trends are more fully discussed later, but there is a clear trend of more time consumed the larger the graph, though the computational complexity clearly differs across the algorithms. Note that the METIS line is directly underneath the MiniBatchKMeans line, making it quite difficult to see. Suffice it to say that METIS had a near-zero runtime.

4.2 SBM Sparsification Results

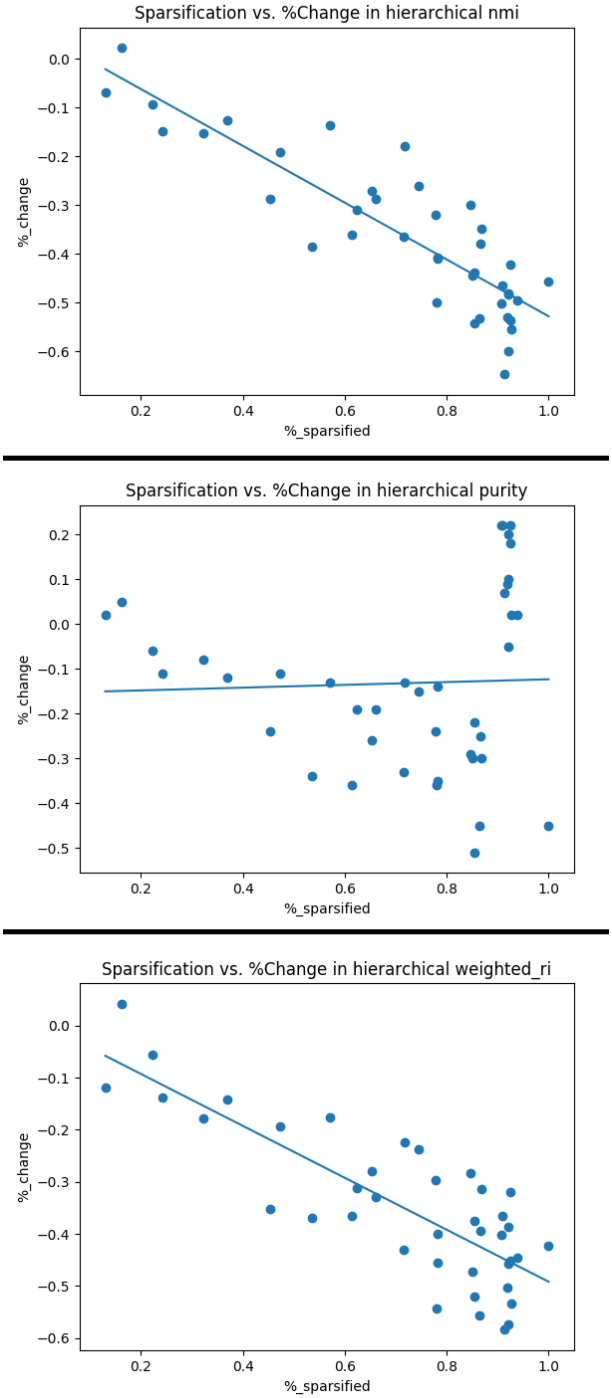

Figure 22: For the fixed values of $p = 0.75, q = 0.10$, we performed sparsified hierarchical spectral clustering experiments, varying the % sparsified by adjusting ϵ from Section 2.9. $\%_{change}$ refers to the change compared to a trial run without sparsification, meaning a value < 0 refers to a drop.

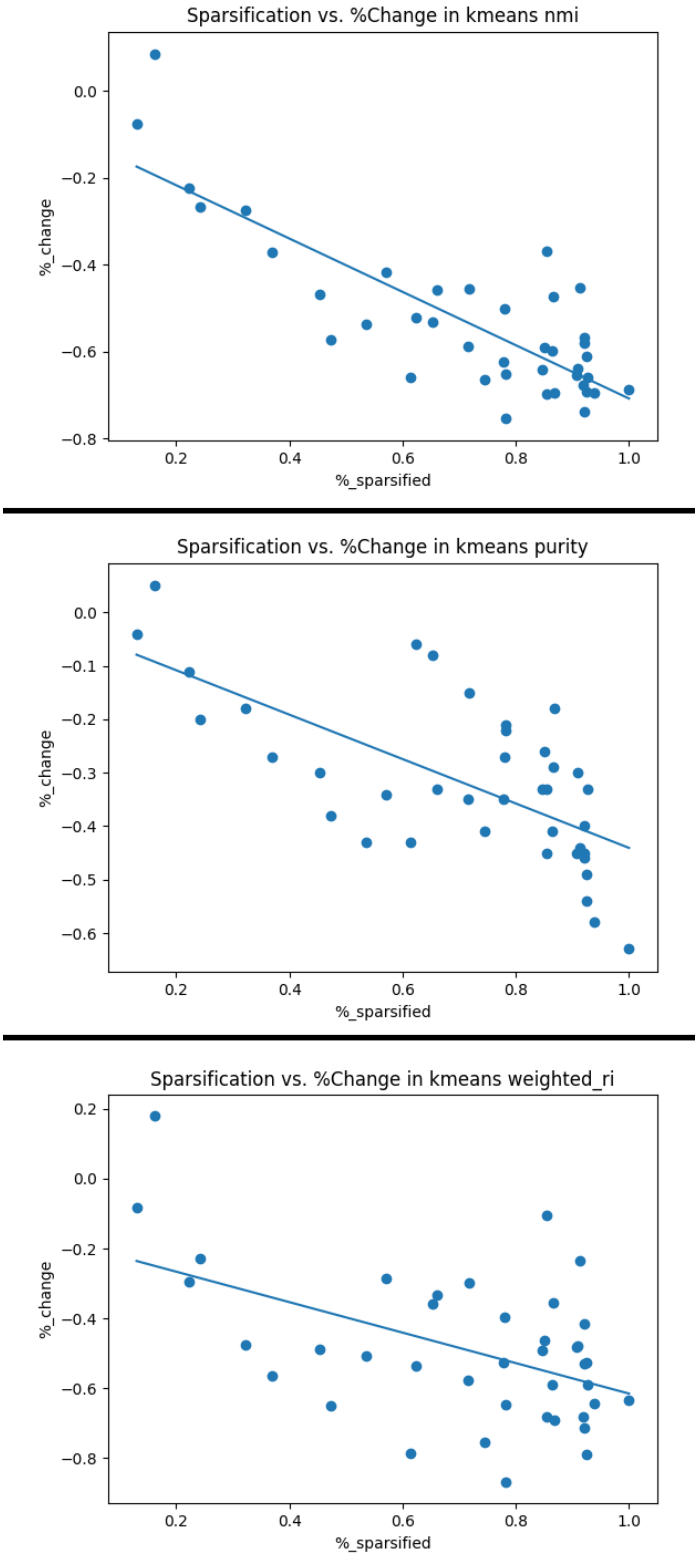

Figure 23: For the fixed values of $p = 0.75, q = 0.10$, we performed sparsified k-means spectral clustering experiments, varying the % sparsified. A full explanation of the experiment can be found in the description of Figure 22.

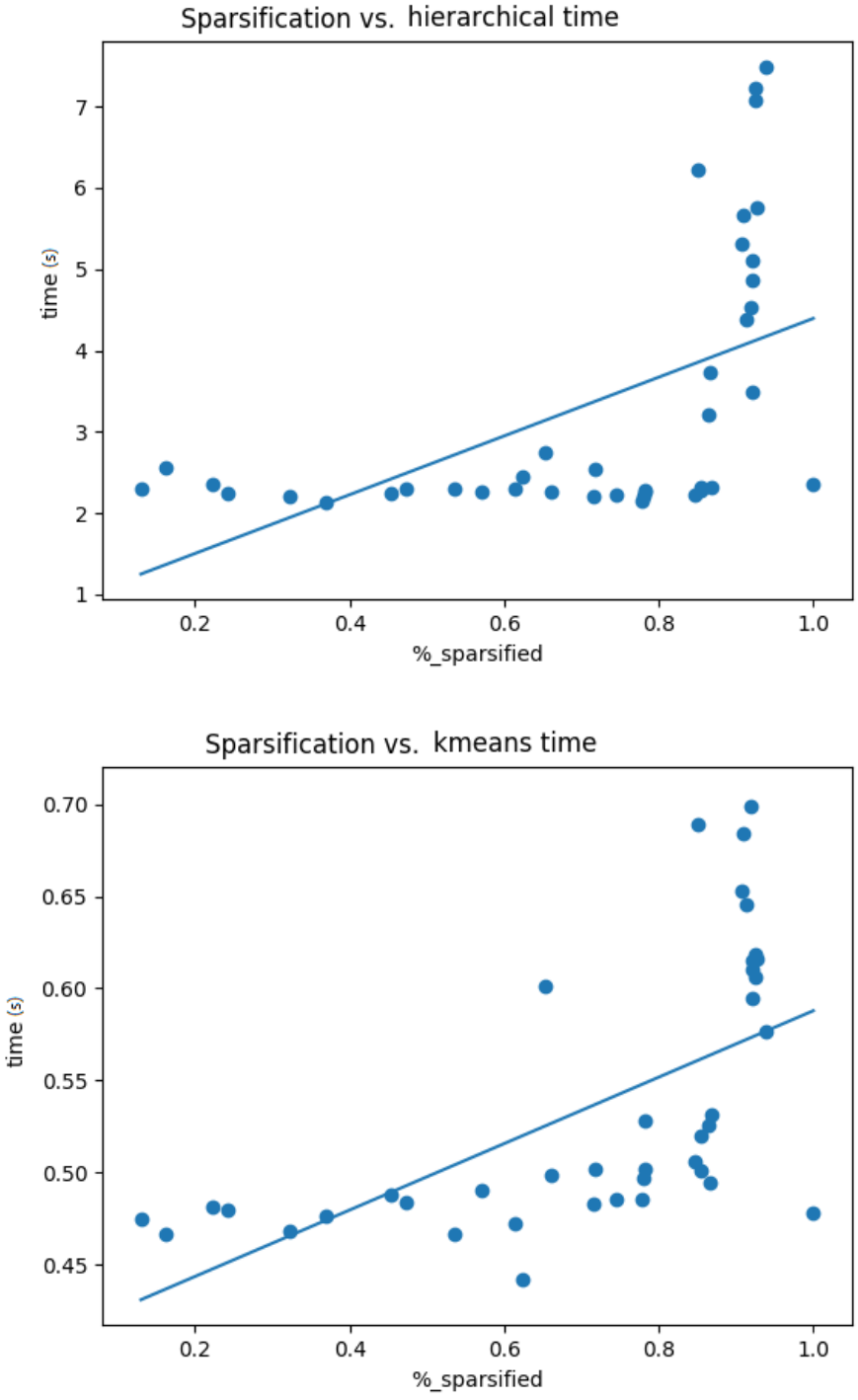

Figure 24: The time required for clustering with hierarchical (top) and k-means spectral clustering (bottom) with respect to the % sparsified. Surprisingly, there was no significant decrease in clustering time required even with increasing sparseness in the graph, more fully elaborated in the corresponding part of Section 5.

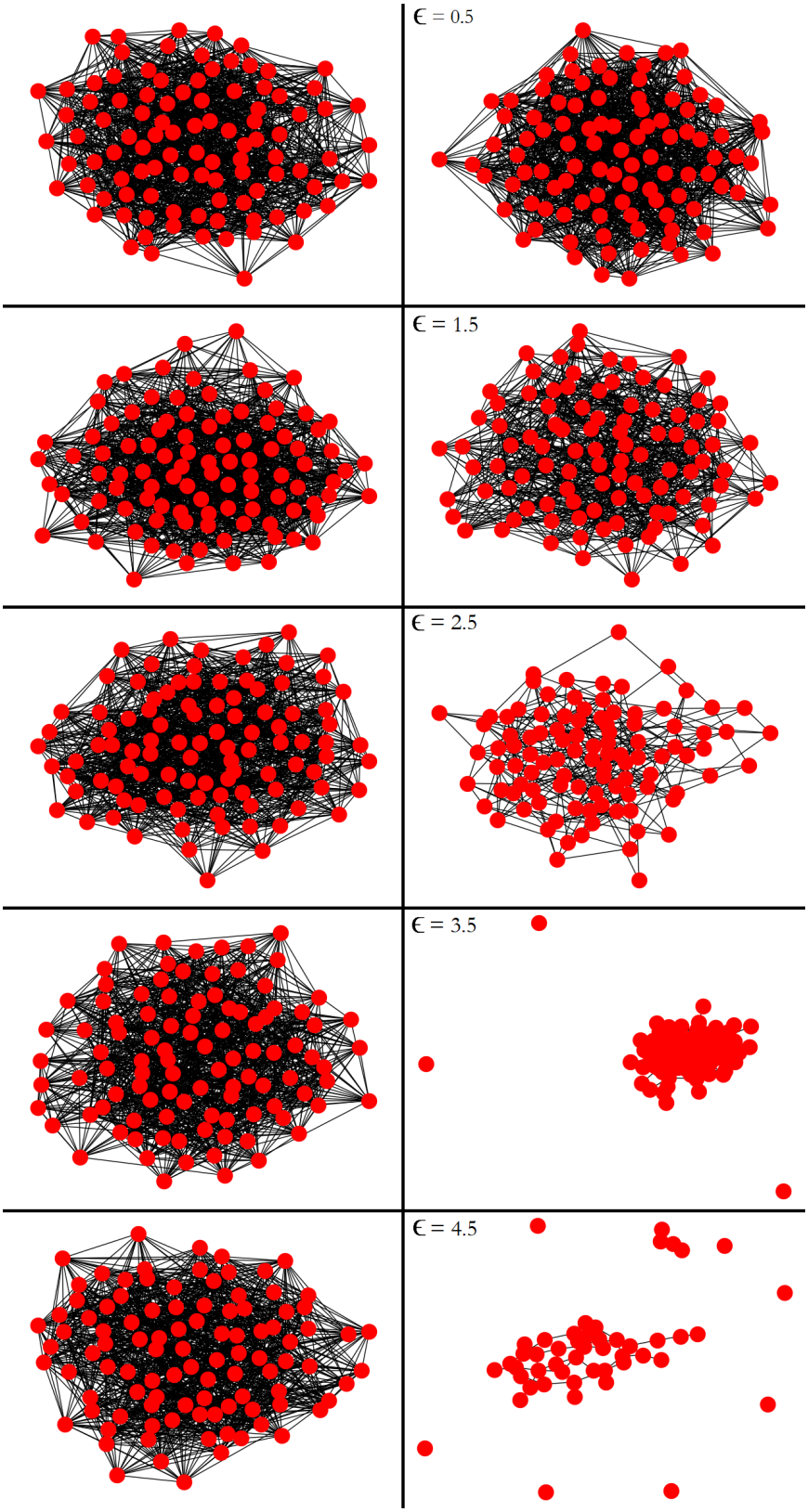

Figure 25: For the original SBM with parameters $p = 0.75, q = 0.10$ on the left-hand side, these show the corresponding sparsified graphs, with parameters $\epsilon = 0.5, 1.5, 2.5, 3.5, 4.5$ respectively following the graphs on the right-hand side. No clustering is depicted in this illustration, as it seeks to solely convey the extent to which edges were removed.

4.3 SBM Coarsening Results

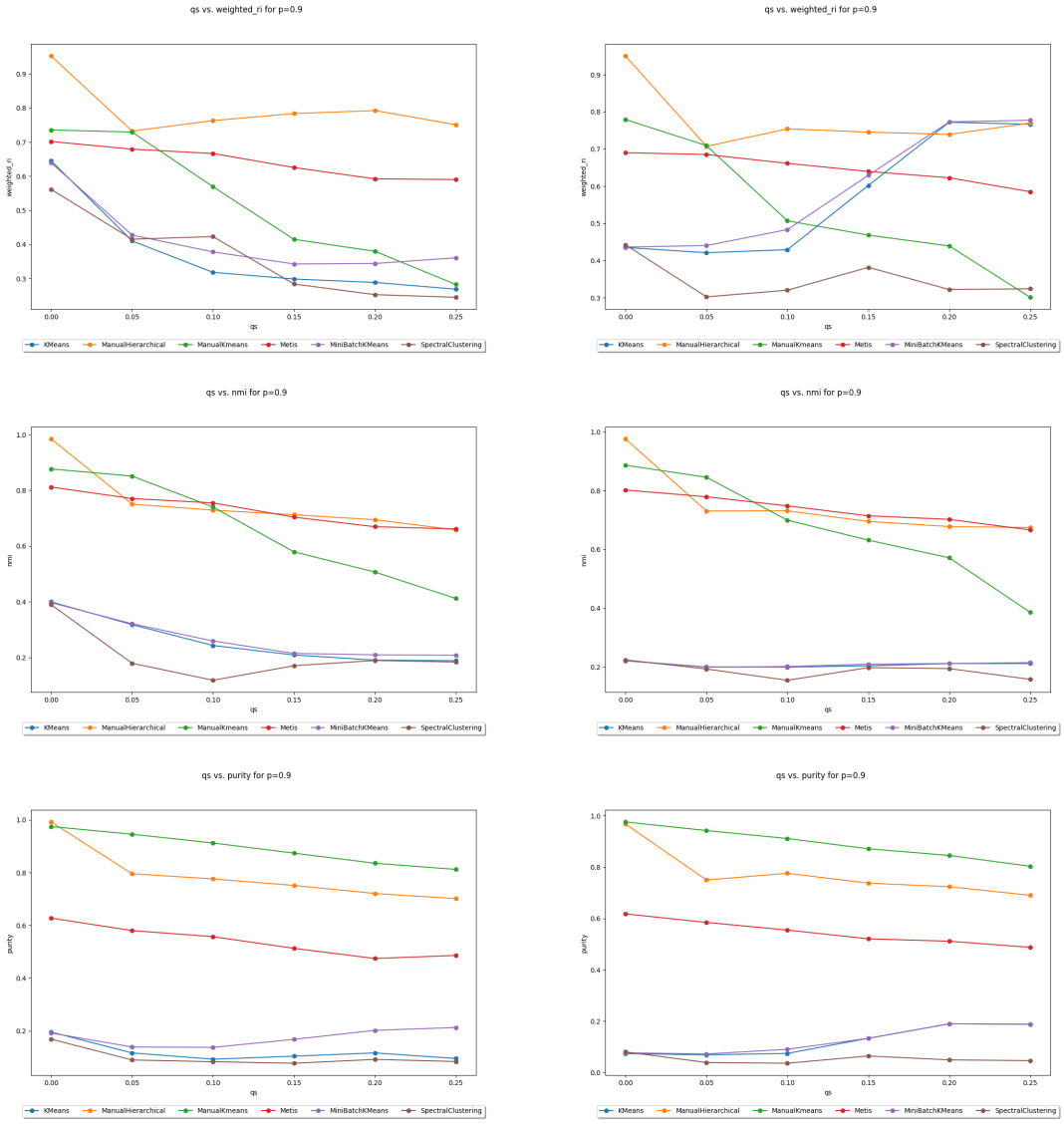

Figure 26: Similar to the standard offline clustering algorithms, this shows (for the fixed choice of $p = 0.9$) the variation of the different accuracy metrics over corresponding values of q. The left column is for **one** coarsening iteration; the right is for **two** iterations. METIS, while included for the sake of completeness, can be ignored for the purposes of discussion.

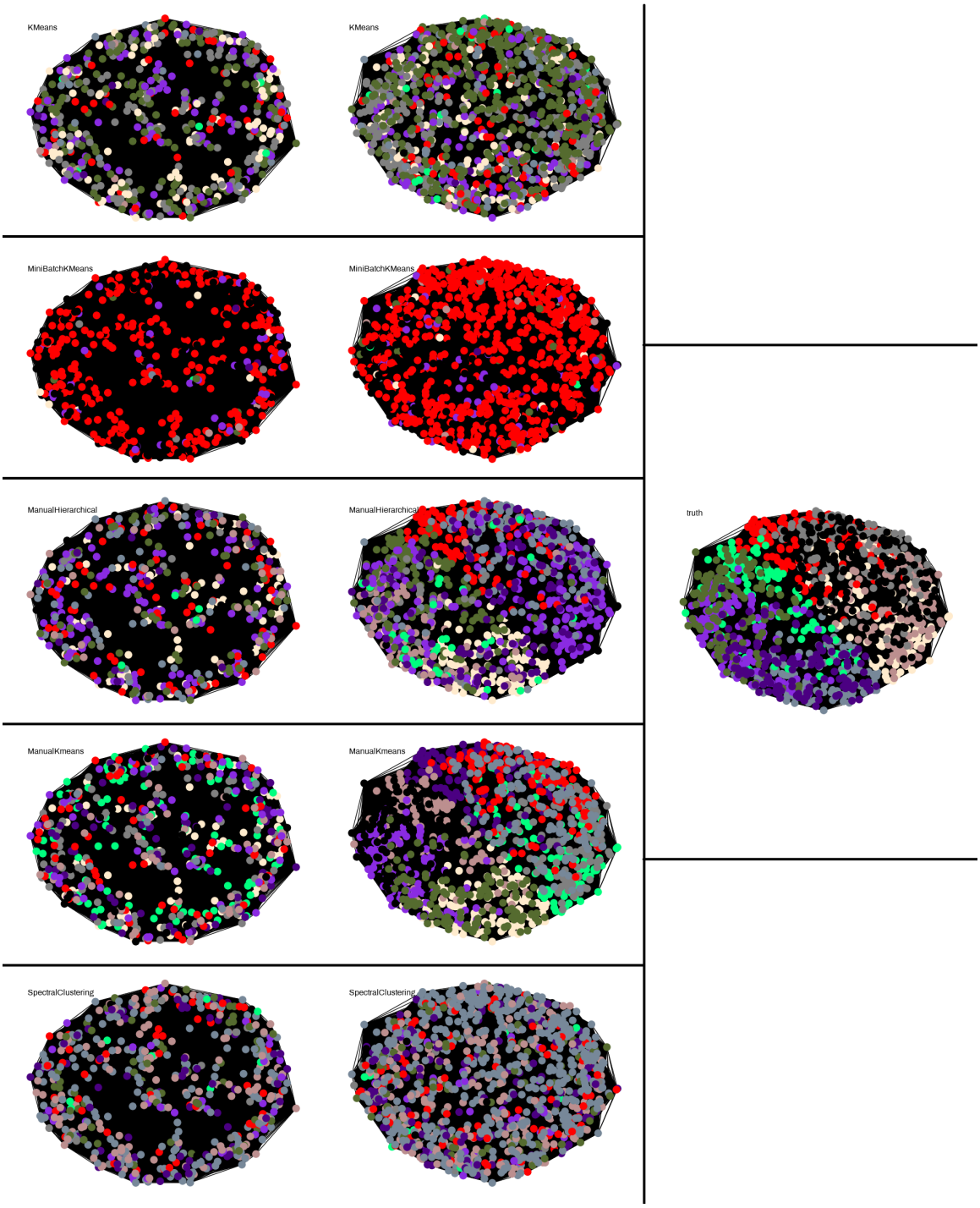

Figure 27: For the original underlying truth graph depicted on the far right, each of the left column of images depicts the clustering results on the coarsened graph. These specifically were the result of running *one* iteration of coarsening, i.e. after one maximal matching was edge contracted. The middle column is each of the clusterings expanded to the original graph.

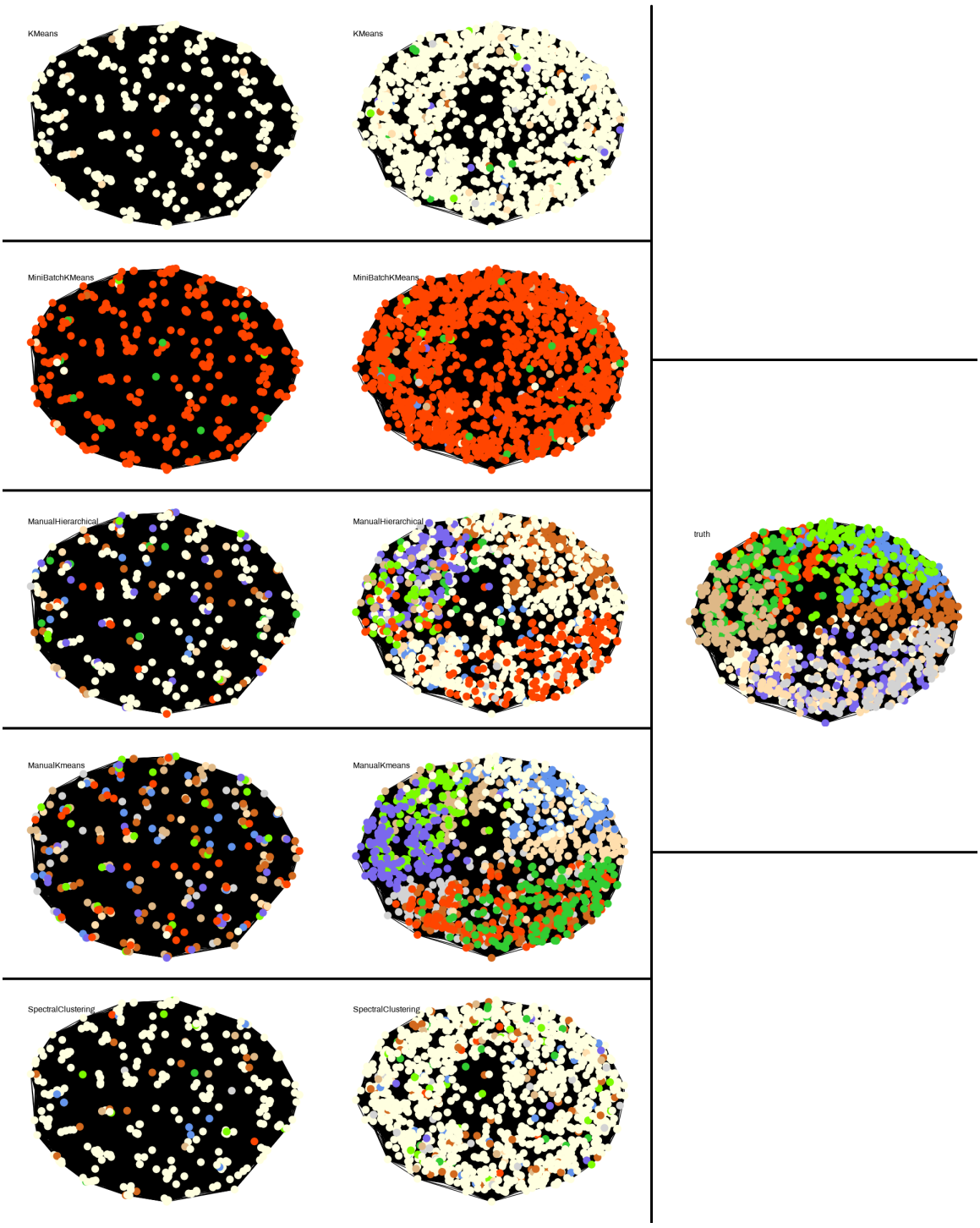

Figure 28: For the original underlying truth graph depicted on the far right, each of the left column of images depicts the clustering results on the coarsened graph. These specifically were the result of running *two* iteration of coarsening, i.e. after two maximal matching was edge contracted. The middle column is each of the clusterings expanded to the original graph.

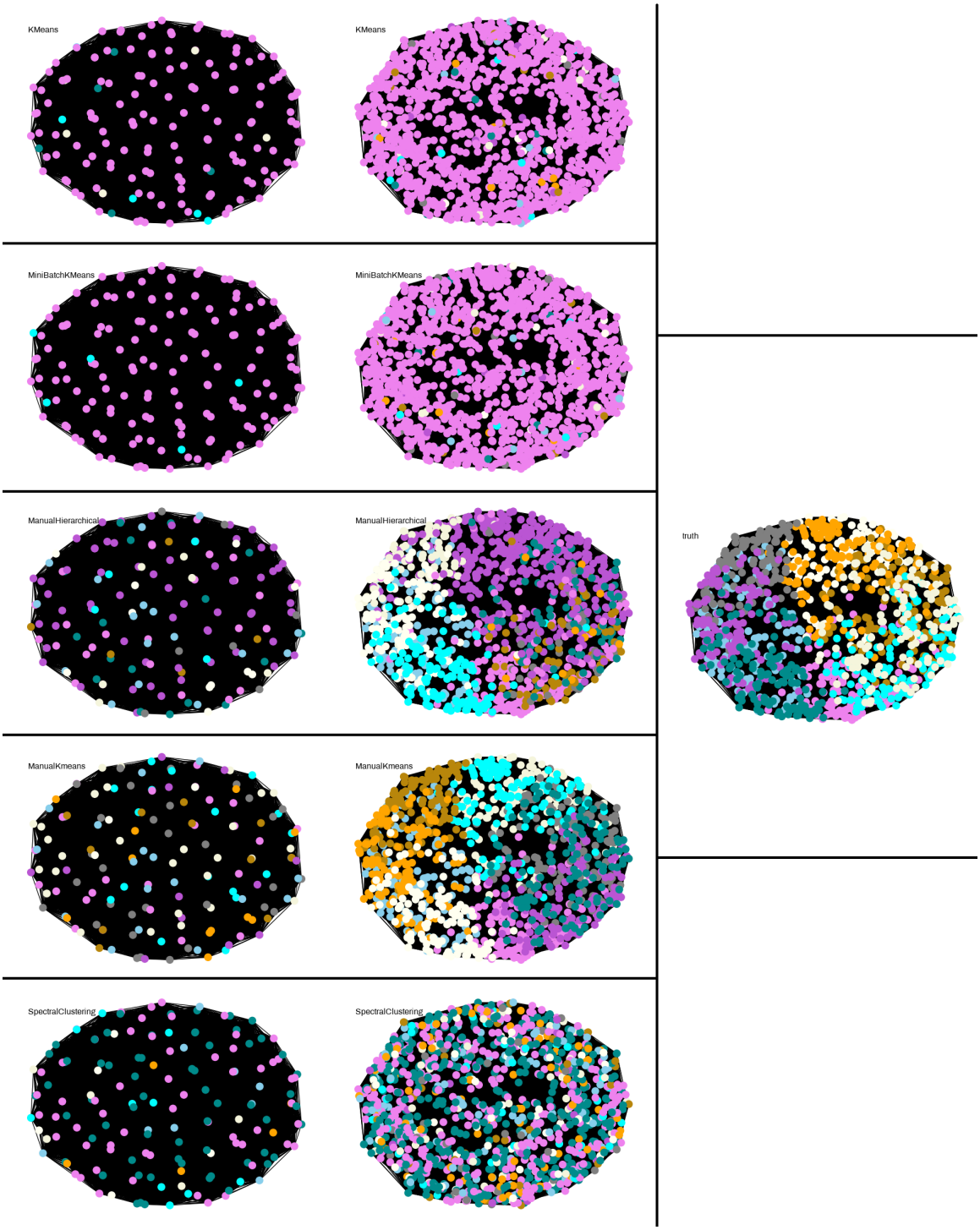

Figure 29: For the original underlying truth graph depicted on the far right, each of the left column of images depicts the clustering results on the coarsened graph. These specifically were the result of running *three* iteration of coarsening, i.e. after three maximal matching was edge contracted. The middle column is each of the clusterings expanded to the original graph.

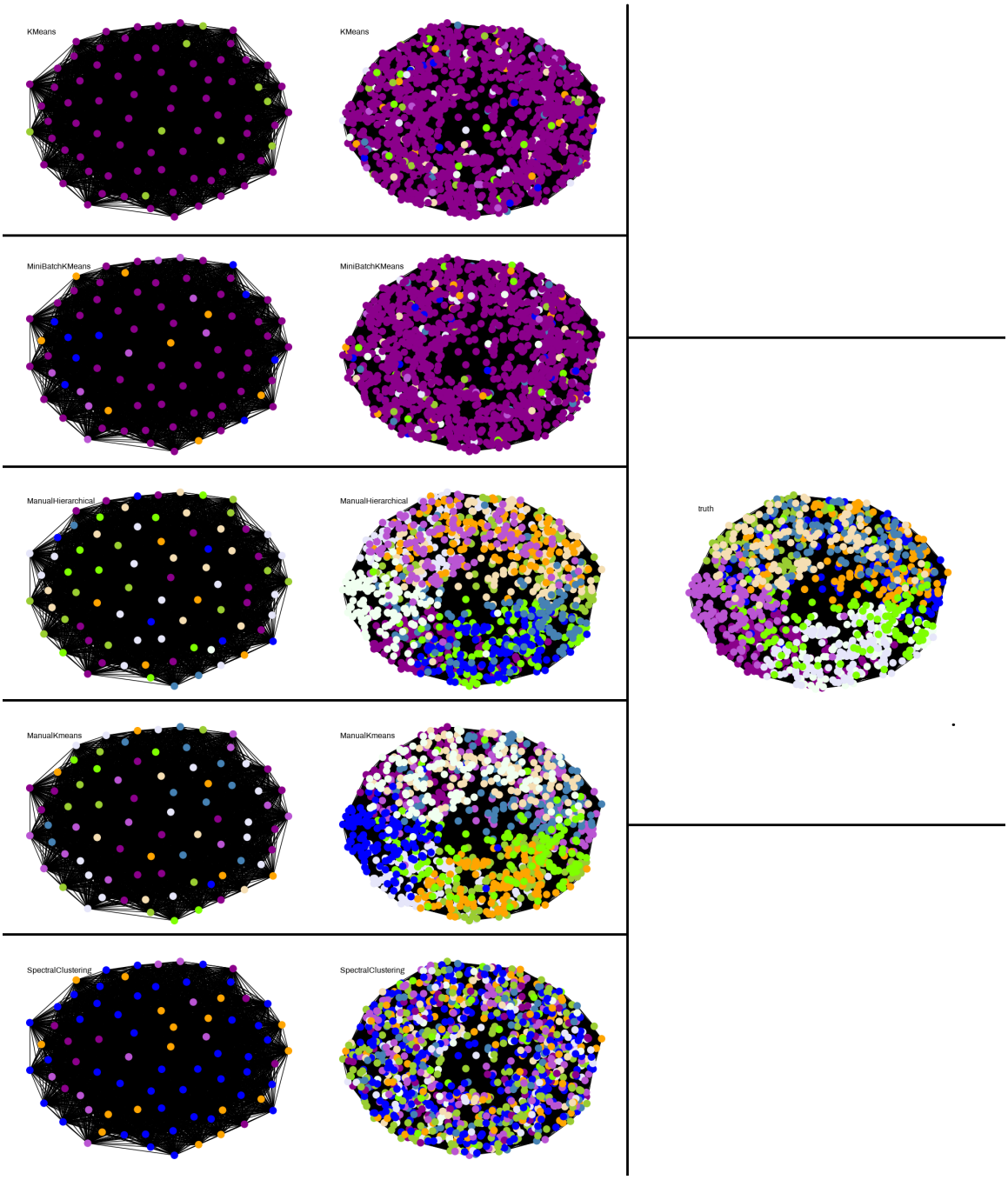

Figure 30: For the original underlying truth graph depicted on the far right, each of the left column of images depicts the clustering results on the coarsened graph. These specifically were the result of running *four* iteration of coarsening, i.e. after four maximal matching was edge contracted. The middle column is each of the clusterings expanded to the original graph.

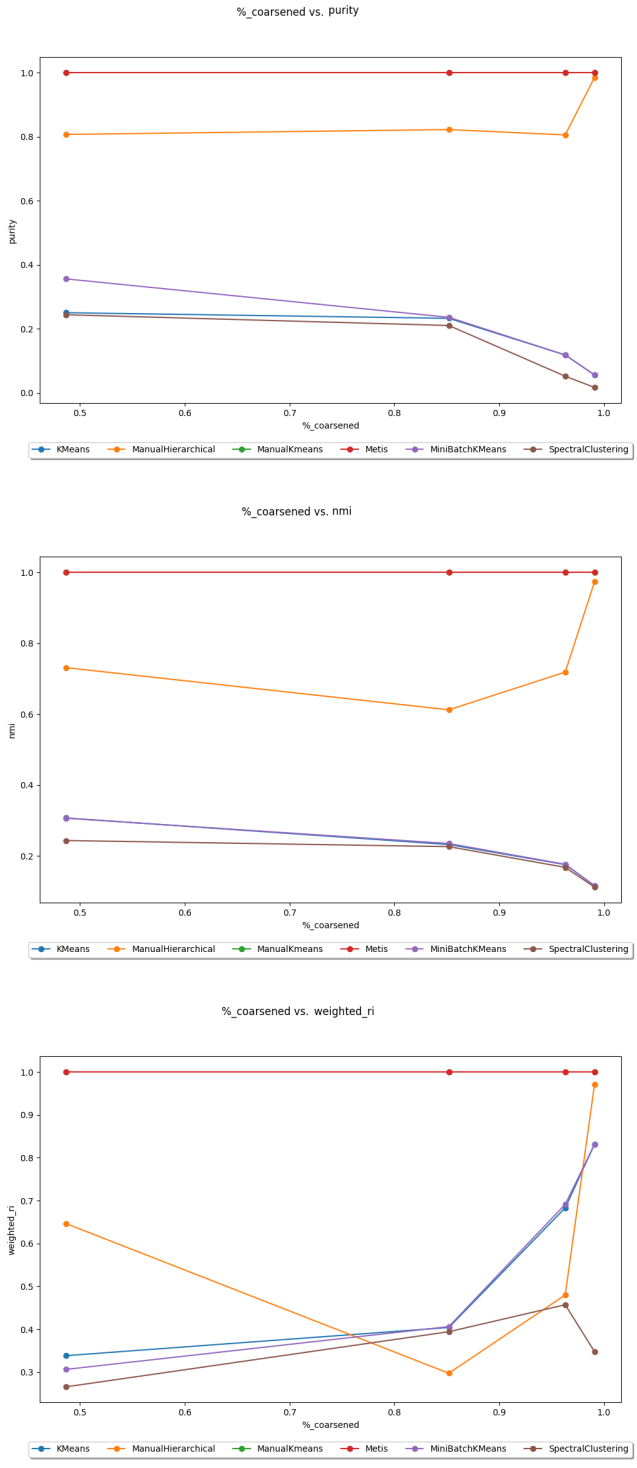

Figure 31: The performances of the various clustering algorithms for the clustering metrics, i.e. purity, NMI, and F-score, against % coarsened. % coarsened refers simply to reduction in edges, namely in the same way % sparsified was previously defined.

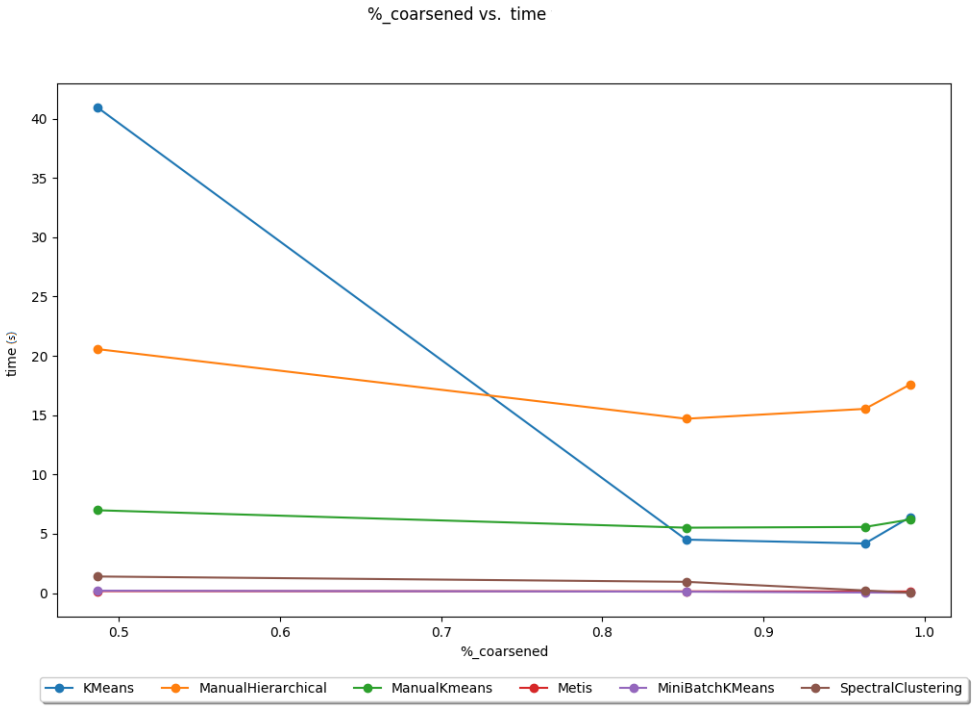

Figure 32: The times required for completion of the trials for the various clustering algorithms against % coarsened. % coarsened refers simply to reduction in edges, namely in the same way % sparsified was previously defined. Once again, METIS had near instantaneous performance and is obscured by the MiniBatchKMeans line.

4.4 Full Dataset Results

METIS was run with the desired output to be 10000 clusters on a subset of the data. *All* the vertices in that subgraph *were* clustered, as desired. However, there were approximately 110,000 addresses considered in this subgraph, making it difficult to visualize the entirety of the raw clustering results. From these clusters, however, some of the addresses could be associated with names, through information made public at [4]. As a result, as previously explained, we were able to taint a subset of the clusters. We present these tainted clusters, i.e. the clusters with real-world identifiers associated in Section 7, specifically in Table 4.

5 Discussion

As previously mentioned, the discussion of the results presented in the previous section are all below.

5.1 SBM Clustering Results

From the results, we see many trends of importance in ascertaining the choice of clustering algorithm. We structure the findings slightly differently than the results were presented, namely first discussing general observations followed by, more importantly, the strengths and weaknesses of each algorithm rather than organizing the discussion by accuracy metric.

5.1.1 General Observations

To begin the discussion, it is of note to study the overall trends of algorithmic performances with respect to p and q alike. That is to say, as expected, all algorithms performed worse by all metrics with increasing q. This is natural to assume, as, as $q \to p$, there is no way to differentiate from the perspective of a single vertex whether another is connected as a result of it being in the cluster or **not** being in the cluster. While the trend *exists* in all the metrics, the F-score seems to be the most sensitive to it, though the reason why such is the case is not entirely clear. Future work may concentrate on this particular point if the F-score is deemed to be the main metric of interest.

In addition, these dropoffs were surprising linear in nature. That is, most algorithms had a linear decrease in performance with increasing q. Natural intuition would tend to there being some threshold value of q (called q_T) such that for any $q > q_T$, accuracies would be very near 0.0 due to the inability to distinguish in-cluster and out-of-cluster realizations. While this is likely the case, as was proven in Theorem 2.2, the experiments conducted herein likely terminated prior to reaching said point.

This trend of decreasing performance as p, q converge is similarly present in the case of p but is less clearly visible. That is to say, for a given fixed value of q, observing the accuracy metric across various values of p tended to show superior performance for high values of p, once again expected by the previous explanation.

5.1.2 K-means

To begin the conversation for clustering algorithms, we start with the standard k-means baseline. Note that any reference to the accuracy of the k-means algorithm in the below discussion also extends

to the partner mini-batch k-means, as the underlying algorithm and its trends are directly tied to those of the original. The main direct observations from the graphs are that k-means performed surprisingly well in accordance to all metrics, that is purity, NMI, and the F-score, specifically for low values of q but had drastic dropoff in performance with increasing q. It further similarly had excellent performance with respect to time, with the mini-batch k-means performing nearly best of all the tested algorithms in the trials conducted.

With these observations in mind, we extend the conversation to understand why it is we see these characteristics. By noting its unanimously positive performance across metrics, it seems natural to believe the clusters were well-formed, i.e. not too compartmentalized, and aligned quite well with the underlying class separation. We now specifically seek to understand why k-means performed extremely well for low values of q or, in other words, for well separable data. This is likely because the k-means algorithm is effectively geometric in nature, as discussed in formulating the k-means results in the form of Voronoi diagrams (i.e. Figure 7). By "geometric," we mean there is a natural method for taking the results of k-means and visualizing the *method* by which the final clusters for vertices were assigned. This is an important yet subtle point to understand: there is a *large* difference between visualizing the final *clusters* assigned to vertices and visualizing the *method* to determine them. The former can be done by simply embedding the vertices into a 2D space and coloring them according to the cluster, as was done in our results. In contrast, the latter visualizes *why* particular clusters were assigned, i.e. Voronoi regions for k-means. No comparable visualization methods exist in spectral methods, since they often captures non-planar clusters, inherently making the separators impossible to visualize.

In fact, taking a step aside, the graphing algorithm itself, namely the force-based algorithm used to visualize/plot the graphs in NetworkX, serves as a spatial clustering algorithm. That is, in having mutual repulsion between vertices and attractors between those that were connected, this algorithm clusters nodes naturally by their connections with one another. Thus, cases where the underlying classes can be well separated in this visualization mark circumstances where an algorithm well-suited for spatial partitioning would perform quite well. This notion of a well-suited "spatial partitioning" algorithm is precisely that of a "geometric" algorithm presented above, meaning k-means seems naturally suited to these circumstances. Hence, overall, k-means is quite well-suited for accurately extracting clusters where q is quite low, which can be empirically approximated by checking the validity of the visualization associated with the graph. In line with that, with higher q values,

the associations become far harder to extract in a plane, substantially diminishing k-means' utility in said circumstances. This additionally explains the greater sensitivity displayed by k-means and mini-batch k-means with respect to q over all the metrics.

As previously mentioned, with the extensive use k-means finds in academia and industry alike, it has been greatly engineered and optimized in terms of performance time. This, of course, comes bundled with the fact that its complementary mini-batch algorithm is even more performant, as it only uses batches of the data to perform updates for the associated means, much in the way mini-batch gradient descent trades off time efficiency for accuracy. In addition to these notes, the general suite of `scikit-learn` algorithms all perform better than those developed manually. This should come as no surprise considering both the large community involved in the development of `scikit-learn`, its extensive industry use, and lack of thorough optimization considered in procuring the manual implementations of spectral clustering.

5.1.3 K-means Spectral Clustering

The k-means and hierarchical spectral clustering methods had surprisingly different behaviors in comparison with one another, for which reason we discuss the two separately. In particular, noting general observations, k-means spectral clustering performed significantly better with respect to the purity than with the other two metrics and was in fact the best performing algorithm with respect to purity. For both of NMI and the F-score, k-means spectral clustering's performance drastically fell off with q though it remained quite high over all such values with respect to purity. In addition, the k-means spectral clustering was quite efficient, largely because it only involved a single transformation before applying the k-means algorithm. It is additionally of note that both of the manually implemented algorithms were highly performant with respect to purity as compared to the performance of algorithms that came pre-developed in the `scikit-learn` package.

Having brought up these observations, it becomes necessary to understand why k-means spectral clustering had such a disparity between these various metrics. Seeing as purity and NMI are closely tied in how they compute accuracy with the main difference being that the latter enforces a "penalty" to assigning each node its own cluster while the former has no such enforcement, it seems natural to assume this disparity is the result of k-means spectral clustering tending to over-compartmentalize the clustering. Specifically, purity would reward a clustering that assigns vertex to its own cluster, as each cluster would have only vertices of a single class trivially, whereas NMI would punish this

case, as the entropy of such a clustering is quite high. In fact, to confirm this hypothesis, simply refer to Figure 19, specifically the *ManualKmeans* clustering. From this, we see that the clustering is, in fact, largely distributed amongst the purple, orange, and tan classes, with the rest of the colors, such as aqua, grey, and white, only being associated with a single vertex.

One conjecture of the reason behind said overcompartmentalization arises from the nature of k-means spectral clustering being a one-shot algorithm, which causes it to miss key substructure that exists as a result of the disparate cluster sizes. Specifically, k-means hiearchical clustering is *very* sensitive to small clusters. Namely, in the case of small classes, a single "dirty edge," i.e. presence of an edge when one should not exist or vice versa, would greatly throw off the corresponding eigenvector relative to the impact a similar dirty edge would have on an eigenvector of a larger class. As a result, the corresponding eigenvalue would likely not be sufficiently close to 0 to be in the first k eigenvectors, meaning it would completely ruin the prospect of clustering. This is in sharp contrast with the hierarchical complement, as we discuss in the section that follows. This dramatic sensitivity to noisy in small clusters explains the *very* sharp drop in performance with respect to NMI and the F-score with increasing q. After all, q effectively serves to replicate the real-world notion of "noisy" predictors. This suggests that k-means hierarchial clustering would best be suited for cases with uniformly sized classes and low noise.

Clustering was very fast for spectral k-means, making it appealing on the practical front. This, as previously mentioned, is likely because the eigenvector computation, which serves as the bottleneck in most spectral algorithms, only had to be performed once prior to switching back to the heavily optimized k-means algorithm. Having said that, therefore, it follows that k-means spectral clustering is quite ill-suited for the purposes of deanonymization due to its tendency to overcompartmentalize. Thus, only in circumstances where clusters are expected to be of similar size or where purity is the priority and NMI is explicitly *not* should k-means spectral clustering be considered.

5.1.4 Hierarchical Spectral Clustering

In contrast to k-means spectral clustering, hierarchical spectral clustering performed exceptionally well per the accuracy metrics, nearly beating out all other algorithms in each trial. In addition, hierarchical spectral clustering had little in the way of sensitivity to q. That is to say, while it did have a drop in performance across the metrics, the drop was completely within reason of the clustering task becoming more challenging. This is a *highly* desirable property to have in our final

use case, specifically as the q is both an unknown value and will likely vary across different regions in the graph. The former is clearly desirable, as it means this approach is not particularly sensitive to noisy heuristics, putting to rest any concern of the overall graph collapsing to supernodes as was previously a concern. The latter is a subtle point that we wish to elaborate upon more below, owing to the unique approach hierarchical clustering employs.

In particular, towards the latter point, we return to how the k-means spectral clustering tended to miss key substructure resulting from having to extract the clusters in a single run. Hierarchical clustering, in contrast, has ample opportunity to identify substructure, as each iteration looks to the existing clusters to determine which is "most separable." To reiterate, each iteration of spectral clustering looks at all the current subgraphs that have been produced from bisections, finds each of their splitting eigenvalues, i.e. 2nd smallest eigenvalues, and chooses that which is the smallest. In turn, it only bisects those clusters that naturally look at though they should be split.

Thus, to contrast k-means spectral clustering, with respect to the sensitivity to variations in a small cluster, such noise would only soil a single "leaf" in the tree of clustering bisections. That is to say, rather than completely changing the embedding space for the *entire* graph, as is the case for k-means spectral clustering, this noise would only affect the vertices associated with that particular class, as the 2nd eigenvalue associated with that subgraph would likely not be sufficiently small for it to be deemed as corresponding to a set of easily separable vertices.

Additionally, this step-wise nature of hierarchical clustering very naturally lends itself to the deanonymization context considered here. That is to say, if we consider a particular subgraph of the overall graph, it likely resembles another SBM with different parameters than those of the original graph. In turn, it seems natural to perform the subgraph clustering separately than that for the original graph to achieve a more accurate clustering, as is done by heirarchical clustering. Additionally, due to the little degradation of hierarchical spectral clustering over q, it follows that this procedure extends well to all the subgraphs clustered in the various stages of the algorithm.

Unfortunately, as a result of this separation of clustering through the series of subgraphs, hiearchical spectral clustering is *significantly* slower than all other clustering algorithms, which effectively renders it unusable for the final application. This follows from the repeated eigendecompositions that have to be performed on the subgraphs, each of which takes time proportional to $O(n^3)$, though the n is reduced in considering only subgraphs. The strengths of the hierarchical spectral clustering are, therefore, its ability to accurately perform clustering, as a consequence of handling the clustering

on each subgraph at separate points and additionally not being sensitive to stray "noisy" edges. Its main weakness is definitely its time efficiency, specifically its lack thereof. Future implementations should seek to optimize its implementation, as it seems to hold major promise in extracting accurate clusters. Thus, hierarchical spectral clustering should be applied in contexts where the graph at hand has a modest number of nodes or where time is not critical.

5.1.5 `scikit` Spectral Clustering

There is little to glean from the `scikit` implementation of spectral clustering other than observing that its performance parallels that of the k-means spectral clustering algorithm. In other words, it seems quite likely that `scikit`'s implementation of spectral clustering is k-means in nature, though there may be additional optimizations employed causing its performance to be slightly worse than that achieved with the manual implementation. Thus, all the observations and associated explanations provided in the section describing k-means spectral clustering hold here with no additional insights that can be gleaned.

5.1.6 DBSCAN

Unfortunately, it turned out that DBSCAN rarely produced any non-trivial results for the clustering trials. That is to say, most of its results simply returned all the vertices being in a single cluster, which resulted in poor performance across *all* metrics, despite its time efficiency being one of the best of the algorithms tested. This is largely due to the sensitivity in its classification results to its hyperparameters, which we fixed through all the trials (specifically to $\epsilon = 10.0$ and $numPts = 5$). This decision to fix its parameters was made as it simulates the expected final use case. That is to say, it seems unlikely the algorithm would be run multiple times on the final dataset while tweaking the paramater to obtain sensible results. It, instead, makes more sense to achieve desirable results using some fixed parameter set on a test set and then retain those for the final data clustering.

In addition to returning only large superclusters in the results, DBSCAN rarely extracted more than three clusters, even when its results were non-trivial. That is to say, it seems highly unlikely that the cases of widely varying class sizes would be handled appropriately by DBSCAN, making it ill-suited for the purposes of deanonymization. In addition, many of those vertices that were not in the largest cluster ended up being classified as "outliers," i.e. in accordance to the algorithm, where certain vertices can be not assigned to a particular cluster.

One point of note is that DBSCAN seemingly has an increase in F-score towards the tail end of q values. That is to say, with increasing q, it seemingly seems to perform better. In looking at the associated clustering plots, however, we see that this is not the result of it performing more closely to the underlying classes but rather an artifact of how the F-score is defined and how outliers were simply not considered in the intersections. This resulted in the TP, FN terms (in the F-score definition), for example, being quite small for the high q trials. In other words, as the DBSCAN only "classified" a small percentage of the overall graph, the F-score was simply calculated on this subset, which artifically inflated its value.

That being said, DBSCAN is an algorithm whose main use cases are where rapid iteration is possible. That is to say, it can be considered in situations where the algorithm can be run multiple times while tweaking the parameters and verifying the outputs' validity. This is helped by the exceptional speed with which DBSCAN completes the clustering task.

5.1.7 METIS

METIS had many odd artifacts of performance across the metrics. Noteworthy was that METIS had *very* little sensitivity to either of p, q. Specifically, while it tended to perform slightly worse with increasing q, its drop was nearly always the least of all the algorithms. This, as emphasized in the analysis of hierarchical spectral clustering, is a very desirable property, though largely for the first reason, namely that of not knowing to what extent the various heuristics will introduce noise into the final graph. In line with that, however, METIS tended to have relatively poor performance across all metrics for low values of q but relatively high performance for high q, i.e. with respect to the other algorithms. In other words, METIS had mediocre accuracy scores for low q values, though other algorithms' performances substantially dropped with increasing q while METIS remained around the same level. An additional point of note is that METIS performed quite poorly on the purity metric but quite well per both of NMI and the F-score.

With respect to the first note, the lack of sensitivity likely follows from the coarsened graph being largely consistent across q. That is to say, while an increasing q means that vertices that should *not* be coarsened into a single vertex have a higher probability of doing so, this is still offset by the fact there will always be more edges between vertices of the same class, meaning, over repeated trials, intraclass vertices would tend to be coarsened together more often than the interclass vertices. Thus, METIS inherently is less sensitive to q than the other algorithms by producing a smaller, coarsened

graph that is largely independent of the associated graph's microstructure.

The disparity of purity in contrast to the other metrics likely also has to do with this ignorance of microstructure. Producing a highly NMI and F-score performant clustering while lacking in purity suggests that, while a majority of vertices were pairwise classified correctly and the classifications similarly non-compartmentalized, the errors concerned the sizes of the clusters. In particular, this suggests there is some disparity between the underlying class size distribution and that predicted by the clustering. This naturally explains the dip in purity, since, as previously explained, purity is computed by effectively assigning each cluster with its best "truth class" and, assuming these, finding the overlap in vertex cluster IDs with their class IDs. Hence, if we consider an example where the underlying truth had one large class and one small class whereas the clustering produced two classes of even size, *both* clusters would be associated with the larger of the two classes in computing the purity, as it would have a larger overlap with both of the two classes as compared to the smaller simply by virtue of being larger. In this case, even a small number of pairwise errors could cause one cluster to be misclassified as a different true class, in turn substantially degrading the purity score. This artifact is in fact visible in Figure 19, which demonstrates that METIS seems to procure clusters of relatively uniform size.

METIS, however, is *significantly* faster than all other algorithms, pushing it as the ideal choice of use. Of course, this time efficiency likely arises from three sources. The first is one purely a matter of implementation, namely METIS is implemented in `C++` as compared to these other algorithms, which were developed in Python, though a majority tend to rely on `C++` API hooks rather than pure Python. The second is that METIS is written for parallel execution, that is, taking advantage of all cores present on the machine at hand. Unfortunately, Python has a global thread lock, making this less of a possibility for implementing such clustering efficiency improvements. The final point is that of the algorithm itself, which defeats the main $O(n^3)$ bottleneck of eigendecomposition by significantly decreasing the graph size. Thus, with these things considered, METIS was the clear choice in considering the time complexity.

Thus, the main contexts in which METIS is well applied is where purity of the final clusters is not of concern or where the distribution of cluster sizes across the data is expected to largely uniform. In addition, METIS should applied where time efficiency is of great concern, as its execution is near instantaneous.

5.1.8 Clustering Algorithm Summary

We now summarize the findings of this section in the form of a table, namely where the various checkmarks denote the extent to which each clustering algorithm had a strength respect to a particular metric. Note that the lack of checkmarks indicates poor performance.

Table 3: Clustering Algorithm Strengths Summary

	K-means	**K-means Spectral**	**Hierarchical Spectral**	**DBSCAN**	**METIS**
Purity	✓	✓✓	✓✓		
NMI	✓		✓✓		✓✓
F-score	✓		✓✓		✓✓
Time	✓✓	✓		✓✓	✓✓✓

5.2 SBM Sparsification Results

We now turn to discussing the effectivenesses of the preprocessing techniques considered. As expected by definition of the spectral clustering parameter, we had greater sparsification happening as $\epsilon \to \infty$. In line with that, only the two spectral clustering algorithms were of relevance for this particular experiment, as this sparsification method was specifically chosen with the intention of procuring a graph whose Laplacian matrix was as similar to that of the original as possible, which is largely of relevance for spectral clustering. In addition, the spectral clustering of `scikit-learn` was not tested, as the previous section revealed that it consistently performed worse than either of the manually implemented alternatives by any of the metrics aside from time.

Of note is the extent to which sparsification resulted in a degradation of clustering performance. Specifically, even a relatively small percent of sparsification, i.e. 10% of the overall graph edges, resulted in a drastic drop of most accuracy metrics (roughly by 5-10%). The main noteable behavior seems to be a spike in performance of the purity at the point of 90% sparsification. This, however, simply seems to suggest that, given the extremely poor performance of the other two metrics, that this was simply the result of extreme compartmentalization, as was described in the case of k-means spectral clustering. This, however, is completely reasonable, as the case of higher % sparsification effectively corresponds to the case of having isolated vertices, in which clustering algorithms have little information suggesting vertices should in fact be clustered together, thus favoring the compartmentalized result. The trend of diminishing performance with further sparsification further seems to be quite linear.

Surprisingly, however, the desired effect of time reduction was not too significant. It turns out

that sparse matrices are not any easier to eigendecompose than their dense counterparts. Although there are more efficient means of calculating just the first few eigenvectors, rather than the entire decomposition, that arise in the case of sparse matrices, the original graph matrices were sufficiently sparse as to allow for that to be applied. In other words, sparsification offers little of service for our purpose, as the graph of 200 million vertices and 10 billion edges is roughly only .000025% filled with non-zero values. Thus, it is highly recommended that sparsification *not* be employed as a preprocessing step in this context, as there was no additional time saved in exchange for the loss in accuracy.

5.3 SBM Coarsening Results

As mentioned in the preceding exposition, discussion of METIS was moot for this particular setting. As it already includes coarsening, additional coarsening with worse reconstruction layered on top is impractical for future purposes, meaning there is little practical value to be gleaned from that analysis. Instead, we choose to investigate other trends. We structure this discussion of the effectiveness of graph coarsening as follows. We first discuss the trends visible in Figure 26, namely the trends of accuracy metrics with increasing degrees of coarsening across fixed values of p with varying q. We then discuss the results of the clustering produced by the various algorithms at hand, specifically in Figures 27, 28, 29, and 30. We follow this with an overall discussion of the trends in accuracies by the different clustering algorithms with respect to percentage of the graph coarsened. Note that, while the coarsening processes can equivalently be classified by the percent of vertices removed from the original graph, we chose to define this by the percent of *edges* removed from the original graph to make the comparison of coarsening to sparsification more straightforward. Thus, this particular discussion focuses on the results presented in Figure 31. We conclude this discussion with a discussion on the impact of coarsening on the time required to complete clustering, presented in Figure 32.

5.3.1 SBM Coarsening Results: Effectiveness Over q

To start, we consider those trends of note visible in Figure 26. Of note is that all the non-manual clustering algorithms' performances were *significantly* diminished in effectiveness as contrasted to the algorithms implemented manually. This likely has to do with the lesser sensitivity of the eigenvectors to the coarsening of the graph as compared to the *significant* sensitivity of the graph spatial

structure to it. This hypothesis is more clearly evident in the clustering results diagrams, which we discuss subsequently. Said hypothesis, however, explains why k-means was drastically worse in the coarsening trials as compared to its typical performance. This idea, however, fails to explain the significant drop in the performance of `scikit` spectral clustering, for which a more thorough understanding of the internal `scikit` source code would be required.

Additionally, simply considering the two manual spectral clustering techniques, the trend with respect to q seems *less* steep than in comparison to the non-sparsified graph. That is to say, while the performance certainly tends to decrease with $q \to p$, it does not drop nearly as rapidly as in Figures 13, 14, and 15. This likely has to do with the previously suggested idea behind METIS's lack of sensitivity to q, namely that of coarsening producing a graph that is largely independent of the microstructures. In other words, since this coarsening remains relatively constant assuming $p \gg q$, the resulting accuracy remains largely unchanged for any choice of (p, q) in this regime. This reasoning, however, does not extend to k-means, as its results fully hinge on having the entirety of the spatial structure of the graph being clustered.

Hence, the main takeaway is that coarsening is relatively well suited for spectral means of clustering, with it reducing sensitivity to q. However, it is ill-suited for spatially sensitive algorithms, such as k-means.

5.3.2 SBM Coarsening Results: Clustering Algorithm Comparison

We now directly discuss the results produced by the various clustering algorithms when subjected to coarsening, as visualized in Figures 27, 28, 29, and 30. Note that, while the previous section had results compiled with fixed p and *varying* q, these trials were *all* conducted with **fixed** values of $p = .90, q = .20$. In addition, the sizes chosen for the clusters were uniform, so as to understand the behavior of the algorithms in a new light, i.e. the sizes were taken to be *all* 125. That is to say, the classes were of the form: `[125,125,125,125,125,125,125,125,125,125]`.

From these clustering results, we see that even in some of the *most* coarsened graphs, the two manual spectral clustering methods worked incredibly well. This is particularly visible in 29 and 30, where the task of clustering is completely non-obvious in the coarsened graph, yet the final results are remarkably close to the ground truth. This additionally suggests the lack of sensitivity of the eigenvectors and eigenvalues of graph Laplacians when coarsened. Said hypothesis should be tested in future studies on graph coarsening with respect to clustering.

Additionally, during the vanilla clustering algorithm discussion, it was suggested that k-means spectral clustering would perform quite well in the context where clusters were of uniform size. Indeed, we see that, as a graph with a uniform distribution of class sizes would similarly result in a uniform size distribution across its coarsened graph, k-means spectral clustering had phenomenal performance in these trials, confirming our hypothesis.

Our hypothesis regarding the k-means algorithms' performances, as was previously seen per its accuracy metrics, was similarly confirmed, specifically that of its sensitivity to spatial properties resulted in its poor coarsening performance. Clearly, per the visualizations in the left columns of these figures, the coarsened graphs could not be well-separated due to the excess of edges associated with each vertex. In turn, k-means similarly could not parse through the excess of edges associated with the vertices, producing a near trivial clustering in all cases.

Thus, from these figures we can confirm our suspicions of the degraded performance of k-means and its great contrast to that of the spectral clustering algorithms.

5.3.3 SBM Coarsening Results: Accuracy Metric Trends

With respect to the accuracy metrics, we see that most algorithms tended to perform worse with increasing coarsening *except* for the F-score. Curiously, the pairwise predictive accuracy appears to *increase* with greater coarsening. These trends, however, exclude both of the manually implemented spectral clustering techniques, where k-means spectral clustering perfectly reconstructed clusters per all metrics considered and hierarchical spectral cluster's performance decreased per the metrics. To address k-means spectral clustering, its outstanding performance largely has to do with the construction of this experiment. That is to say, as this experiment began with a uniform size distribution of classes, it was perfectly suited to k-means spectral clustering's strengths. Hierarchical spectral clustering, additionally, had the expected behavior of decreasing performance with additional coarsening, due to the predicted reason of a loss in information with the loss of graph microstructure.

Having said that, we now turn to the performance of all algorithms apart from hierarchical spectral clustering, namely those of the k-means algorithms. Clearly, the two k-means algorithms produced results that effectively put all the vertices into a single cluster, meaning this cluster contained *all* the true underlying classes and was, therefore, unpure. Similarly, knowing that a vertex was in this cluster revealed *nothing* about which class it is associated with in reality, making its

NMI similarly low. Seemingly unexpectedly, the F-score is quite high. The reason behind this is, as all vertices were in the same cluster, any pair of vertices that were in the same class in reality would seemingly be "appropriately" classified. Any pair of vertices that were in separate clusters would, by the same reason, be mostly incorrectly classified. *However*, the latter was less heavily weighed by our choosing $\beta = .5$, thus explaining this seeming boost in accuracy.

5.3.4 SBM Coarsening Results: Time Impact

Surprisingly, however, the coarsening did not provide ample speedup to warrant its integration. From the measured time performance, most of the algorithms were quite static across the number of iterations of coarsening. The supposed boost in speed would arise from the reduction in vertex count rather than that of edges, as evidenced by the lack of speedup from graph sparsification. Specifically, as eigendecomposition is $O(n^3)$, speed is greatly dependent upon the vertex count, meaning any significant reduction should greatly improve performance.

However, it appears as though the process of obtaining a maximal matching or that of projecting clustering results itself takes up any time saved by this reduction, resulting in a relatively flat curve. It seems likely that METIS achieves its remarkable speed through use of parallel computation of matchings or simply a more optimized implementation of the projection algorithm. From this, however, it seems that no manual coarsening can replace that performed in METIS, due to the loss in accuracy without time saved.

Overall, the performance of the hierarchical spectral clustering algorithm was surprisingly *not* significantly diminished per applying coarsening. Future studies, therefore, may wish to improve the efficiency of such coarsening algorithms to use them in conjunction with *hierarchical* spectral clustering to amend its time deficiencies.

5.4 Full Dataset Results

Unfortunately, as we only ran the trial on a subset of the overall dataset, the addresses identified had *no* overlap with the ground truth data. That is, no wallets with known owners from the catered "A Fistful of Bitcoins" dataset were present in the subset of data we clustered, which prevented us from assigning a numerical accuracy score. That being said, however, in the process of associating each cluster with a real-world entity name, we iterated through all the addresses in a given cluster, determined whether it had a publicly visible entity name attached, and, if so, assigned the cluster

to that name. In addition, we tracked to see if a cluster was being assigned multiple names, as this would directly indicate a misprediction by the clustering algorithm. This rudimentary check of clustering accuracy revealed no errors, meaning no two addresses in a single cluster were publicly associated with different entities. While a nice first check, future studies should seek to make this confirmation of the clustering result more thorough.

6 Conclusion

Thus, from this investigation, we have found many points of interest for the deanonymization of Bitcoin and cryptocurrency transactions in general. Specifically, these points tend to deal with the general application of graph clustering algorithms towards the context of a catered heuristics graph associated with the likelihood two wallets corresponding to the same real-world entity. Each of the algorithms considered in this investigation had various strengths and weakness when tested against the corresponding, representative SBM (stochastic block model).

To summarize, k-means performed well against all metrics in the context of having low values of q, namely in geometrically separable graphs, since it is closely tied to planar clustering. Its main applicability, therefore, comes in the form of graphs where heuristics are very carefully catered, i.e. with little noise, and where time efficiency is of great importance. K-means spectral clutering, in contrast, was very suited for achieving high purity without respect to NMI and F-score, largely due to its tendency to overcompartmentalize the vertices. It tends to perform very time efficiently and well where the underlying classes are uniformly sized. In contrast, hierarchical spectral clustering was capable of extracting much of the substructure embedded in the graph, in turn performing clustering that measured well against all accuracy metrics. This, however, was performed in the face of great time issues, making it not applicable for large graphs. DBSCAN performed quite poorly across the board due to our fixing its parameters in accordance to its expected final use case, which resulted in its often returning trivial clusterings or classifying vertices as outliers. METIS achieved high accuracy in terms of both NMI and F-score but not purity, indicating its tendency to mispredict cluster sizes as being more uniform than they were in reality. The algorithm, however, runs *very* fast, making it quite applicable to the large-scale contexts considered here.

In addition to these standard clustering algorithms, we attempted to perform sparsification and coarsening preprocessing to see their relative impact on accuracy and time. Sparsification, however, caused significant reductions in all accuracy metrics, across both k-means and hierarchical spectral clustering, without resulting in any significant reduction in time. We determined that no major improvements in clustering efficiency resulted from the removal of edges, as the eigendecomposition is largely independent of the presence of non-zero elements in the corresponding matrix. Coarsening, similarly, resulted in little reduction in time required for clustering with an associated reduction in most accuracy metrics, though this was more so caused by the inefficiency of the coarsening and reconstruction processes.

From here, we applied METIS to a partially curated heuristics graph to obtain the final deanonymization result. We specifically clustered all the contained addresses in approximately 3.45 seconds, though the procurement of data took longer. Additionally, some of the clusters were associated with names, per those publicly available on the Blockchain Explorer site. Unfortunately, no entity identified overlapped with the known set of address correspondences, which made affirming accuracy difficult. That being said, no cluster was found that contained multiple addresses with IDs of the two being *different*. That is to say, no cluster had conflicting associations with real-world entities.

For future follow-on endeavors, there are some leads on what to investigate. The first revolves around the introduction of an automated weight updating algorithm on the heuristics. This would simply be based on some form of gradient descent updating on the weight. Much in the way SGD (stochastic gradient descent) has currently become the de-facto means by which weights are updated in neural networks, this can be directly extended to the context of updating the weights associated with the heuristics based on its predictions on the set of vertices associated with the ground truth data. Initialization could be set uniformly or according to some appropriate random distribution.

A second line of interest revolves around the limitations of METIS. Specifically, as was found in many of the accuracy metrics, METIS is quite poorly performant for non-uniformly sized classes. It, therefore, seems quite natural to potentially augment METIS with a means to handle this circumstance. In line with that, the hierarchical spectral clustering performed remarkably well by all metrics yet could not be applied due to its being inefficient, clearly suggesting the need to either look to parallelizing the algorithm or simply implement it in `C++` to see whether it could be applied to the generalized dataset and procure results of greater substance than were derived herein. Similarly, if the coarsening could be made more efficient, potentially by simply extracting the relevant code as it is implemented in METIS, it could be combined with hierarchical spectral clustering to procure a highly accurate and efficient method to be applied to the final dataset.

Finally, the lack of ground truth was a major limitation in this study. Namely, by catering a more comprehensive dataset of known data points, the accuracy of this algorithm can actually be confirmed going forward. Additionally, future work may simply seek to find side channel information with the addresses identified to confirm the validity of the METIS results on this dataset.

References

[1] Aggarwal, Charu C., et al. "A Framework for Clustering Evolving Data Streams." Proceedings 2003 VLDB Conference, 2003, pp. 8192., doi:10.1016/b978-012722442-8/50016-1.

[2] Arthur, David, and Sergei Vassilvitskii. "k-Means++: The Advantages of Careful Seeding." *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2007.

[3] Batson, Joshua, et al. "Spectral Sparsification of Graphs." Communications of the ACM, vol. 56, no. 8, 2013, p. 87., doi:10.1145/2492007.2492029.

[4] "Bitcoin Block Explorer." Blockchain, blockchain.info/.

[5] "Bitcoin Developer Guide." *Developer Guide - Bitcoin*, bitcoin.org/en/developer-guide#term-locktime.

[6] Chevalier, Cedric, and Ilya Safro. "Comparison of Coarsening Schemes for Multilevel Graph Partitioning." Lecture Notes in Computer Science Learning and Intelligent Optimization, 2009, pp. 191205., doi:10.1007/978-3-642-11169-3_14.

[7] Chi, Yun, et al. "Evolutionary Spectral Clustering by Incorporating Temporal Smoothness." Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '07, 2007, doi:10.1145/1281192.1281212.

[8] Christin, Nicolas. "Traveling the Silk Road: A Measurement of a Large Anonymous Online Marketplace." 2012, doi:10.21236/ada579383.

[9] "Clustering." Scikit-Learn, scikit-learn.org/stable/modules/clustering.html.

[10] "CoinJoin: Bitcoin Privacy for the Real World." Bitcoin Talk, 22 Aug. 2013, bitcointalk.org/index.php?topic=279249.0.

[11] Daszykowski, M., and B. Walczak. "Density-Based Clustering Methods." Comprehensive Chemometrics, 2009, pp. 635654., doi:10.1016/b978-044452701-1.00067-3.

[12] "DBSCAN." *Wikipedia*, Wikimedia Foundation, 19 Apr. 2018, en.wikipedia.org/wiki/DBSCAN.

[13] Dhanjal, Charanpal, et al. "Efficient Eigen-Updating for Spectral Graph Clustering." Neurocomputing, vol. 131, 2014, pp. 440452., doi:10.1016/j.neucom.2013.11.015.

[14] Ermilov, Dmitry, et al. "Automatic Bitcoin Address Clustering." 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), 2017, doi:10.1109/icmla.2017.0-118.

[15] "Evaluation of Clustering." Stanford NLP, 2008, nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-clustering-1.html.

[16] Greenberg, Andy. "Follow The Bitcoins: How We Got Busted Buying Drugs On Silk Road's Black Market." Forbes, Forbes Magazine, 7 Sept. 2013, www.forbes.com/sites/andygreenberg/2013/09/05/follow-the-bitcoins-how-we-got-busted-buying-drugs-on-silk-roads-black-market/#79bb83c8adf7.

[17] Guattery, Stephen, and Gary L. Miller. "On the Quality of Spectral Separators." *SIAM Journal on Matrix Analysis and Applications*, vol. 19, no. 3, 1998, pp. 701719., doi:10.1137/s0895479896312262.

[18] "Heuristics." *BlockSci 0.4.5 Documentation*, citp.github.io/BlockSci/heuristics/heuristics.html.

[19] Kalodner, Harry. "BlockSci: Design and Applications of a Blockchain Analysis Platform." ArXiv Preprint, 2017, doi:1709.02489.

[20] Karypis, George, and Vipin Kumar. "METIS–Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0." 1995.

[21] Katz, Lily. "Goldman Says Cryptocurrencies May Succeed as Form of Real Money." Bloomberg.com, Bloomberg, 10 Jan. 2018, www.bloomberg.com/news/articles/2018-01-10/goldman-says-viability-of-crypto-is-highest-in-developing-world.

[22] Kothari, Pravesh. "Karger's Min Cut Algorithm." 2015, Princeton University, Princeton University.

[23] LaViers, Amy, et al. "Dynamic Spectral Clustering." Georgia Institute of Technology, 2010.

[24] Liberty, Edo. "Simple and Deterministic Matrix Sketching." Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '13, 2013, doi:10.1145/2487575.2487623.

[25] Luxburg, Ulrike Von. "A Tutorial on Spectral Clustering." Statistics and Computing, vol. 17, no. 4, 2007, pp. 395416., doi:10.1007/s11222-007-9033-z.

[26] Meiklejohn, Sarah, et al. "A Fistful of Bitcoins." Proceedings of the 2013 Conference on Internet Measurement Conference - IMC '13, 2013, doi:10.1145/2504730.2504747.

[27] Moser, Malte, et al. "An Inquiry into Money Laundering Tools in the Bitcoin Ecosystem." 2013 APWG ECrime Researchers Summit, 2013, doi:10.1109/ecrs.2013.6805780.

[28] Mossel, Elchanan, et al. "Reconstruction and Estimation in the Planted Partition Model." Probability Theory and Related Fields, vol. 162, no. 3-4, 2014, pp. 431461., doi:10.1007/s00440-014-0576-6.

[29] Nakamoto, Satoshi. "Bitcoin: A Peer-to-Peer Electronic Cash System." 2008.

[30] Ning, Huazhong, et al. "Incremental Spectral Clustering by Efficiently Updating the Eigen-System." Pattern Recognition, vol. 43, no. 1, 2010, pp. 113127., doi:10.1016/j.patcog.2009.06.001.

[31] "P2PKH Vs P2SH." *Paranoid Bitcoin*, 12 Sept. 2017, www.paranoidbitcoin.com/2017/09/12/a-tuesday-two-part-2-p2pkh-vs-p2sh/.

[32] Pensky, Marianna, and Teng Zhang. "Spectral Clustering in the Dynamic Stochastic Block Model." ArXiv Preprint, 2017, doi:1705.01204.

[33] Purohit, Manish, et al. "Fast Influence-Based Coarsening for Large Networks." Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '14, 2014, doi:10.1145/2623330.2623701.

[34] Riddhiman. "Voronoi Diagrams in Plotly and R." Modern Data, 1 Feb. 2016, moderndata.plot.ly/voronoi-diagrams-in-plotly-and-r/.

[35] Rocha, Israel. "Improvements on Spectral Bisection." ArXiv Preprint, 2017, doi:1703.00268.

[36] Schaeffer, Satu Elisa. "Graph Clustering." Computer Science Review , 2007, pp. 2764.

[37] "Self - Regulatory Organizations; NYSE Arca, Inc; Order Instituting Proceedings to Determine Whether to Approve or Disapprove a Proposed Rule Change to List and Trade the Shares of the ProShares Bitcoin ETF and the ProShares Short Bitcoin ETF under NYSE Arca Rule 8.200 - E, Commentary .02." Securities and Exchange Commission, 23 Mar. 2018.

[38] Shindler, Michael, et al. "Fast and Accurate k-Means For Large Datasets. Advances in Neural Information Processing Systems, 2011.

[39] Shirriff, Ken. "Ken Shirriff's Blog." Bitcoins the Hard Way: Using the Raw Bitcoin Protocol, www.righto.com/2014/02/bitcoins-hard-way-using-raw-bitcoin.html.

[40] Spielman, Daniel A., and Nikhil Srivastava. "Graph Sparsification by Effective Resistances." Proceedings of the Fourtieth Annual ACM Symposium on Theory of Computing - STOC 08, 2008, doi:10.1145/1374376.1374456.

[41] Tamassia, Roberto. Handbook of Graph Drawing and Visualization. CRC Press Taylor & Francis Group, 2016.

[42] Tao, Terrence. "A Review of Probability Theory." *Terry Tao*, terrytao.wordpress.com/2010/01/01/254a-notes-0-a-review-of-probability-theory/.

[43] Trautman, Lawrence J. "Virtual Currencies: Bitcoin & What Now after Liberty Reserve and Silk Road?" SSRN Electronic Journal, 2014, doi:10.2139/ssrn.2393537.

[44] Wander, Matthaus. "Bitcoin Block Data." Wikimedia Commons, 22 June 2013, commons.wikimedia.org/wiki/File:Bitcoin_Block_Data.png.

[45] Whang, Joyce Jiyoung, et al. "Scalable and Memory-Efficient Clustering of Large-Scale Social Networks." 2012 IEEE 12th International Conference on Data Mining, 2012, doi:10.1109/icdm.2012.148.

[46] Yoo, Shinjae, et al. Streaming Spectral Clustering. 2016 IEEE 32nd International Conference on Data Engineering (ICDE), 2016, doi:10.1109/icde.2016.7498277.

[47] Zetzsche, Dirk A., et al. "The ICO Gold Rush: It's a Scam, It's a Bubble, It's a Super Challenge for Regulators." SSRN Electronic Journal, 2017, doi:10.2139/ssrn.3072298.

7 Appendix: Partial Deanonymization Results

Table 4: Partial BTC Deanonymization Results

gkelly	1BWWVLobmdTcpH9pVknpwxr9BisikiNkv2, 1LRsxr4gQgbnQQuCMH2VmHgxdGV5uGfZZe 14uagJsMwzUrgs93cZJK35nJjbPaLsiGUn, 1J9c5x1zeLYVrx3oFTjTetkJ9nn9wTUpfk 1BpgjuTMSSUN5BqEGt1ryfzugG38hTEYF, 1B2xAJERd9v4CxcPqymiePvKVJj65VM5uy 17oqKpowhafnherH8cnmTAu9UG78rtcA9M, 17apfd9RmxWy7bs4mYEFzar1fh278vvkQf 15Z5YJaaNSxeynvr6uW6jQZLwq3n1Hu6RX, 1Ju2SFnWJQonBpa5A3ghtRCctBrC6LudJ5 15vDKunFajcVVezdH1amMiBnWa6FoYugDE, 1Ad9zPhKH1PvEndVJPHNSrugzwV1fjCZcj
pyrrhos	1M9GDDeDpgp77MEHidi1fKbvXDqLmEennA, 1QAfJitvUCkK2ARthKEMFjL1SSVNuNbazG 1DHfipooqt94juJs4gTTfawenMZnwArBvN, 13xRB552qwSmfB1QGAK5thRQYFiysySGdB 1FHGJyTNE5RBQqG29HDJjLTAXvqdRQuebE, 1GTLpXfbux4JBv27mstfAsMLqZGYczoWVd 16hc37FM6ooorqooL1fRhQFMghyQHW46tC, 1gL39TnoUuXrJWhVftdr2qEfLLc9FY8J7 1LMdVCSyQssJWErsYTzN37FexemNMrBXkw, 135UTNs91PX9a7tTHuyBPDGcVusMnz3FW9 13sTBhwcp2nepH4HuqenWEzyrQzRdq1tnX, 1AGyEtx6Ne6B7UFx3VRb5A8doruEiMF9hK
Odin	15BCHKoSBwW8Q6NSjvcoYfspGVDBuy3WZu, 13zJLMai5Hh9H5zRw7f62u9GJXpqi8c9Eg 16htjyVERemFWRsNxRWknzoL2saVnHumSX, 1NwAL1TocngQd49Z1XdBmtJoi4KVyyzZ6u 1FR33MCUYkVDTrfMQWZ1xuoDRcP3S7caJu, 1519kAZELETBFXUM8xieVvDx5KjtuaZg4t 15AVxQbwpQSU3AwPj31XNwKsCxzjWJMvPC, 16GsibEzeoXYhRVfpaQudxvgzSJ6JVLE56 1Loestum3FqCeQi83b5Q7K88hqa5KDgy7P, 13hKkxkFqqNPpp35GHAnsXwuQArdH8XSE6 13ZfLD1YemWqfq3hMtiV7Uo7Ks8TSw3Yvr
bolapara	1Hy7DpnP1Tf2zg3ppN8RVxLVm5RTnXPVwG, 1PRn1aABPokKng8sStgFiv43VdKnE6XCuZ 19YLTPVFrDtyW9vMj3Af6QFWjHQomidjwp, 18uxWvbCpwiQZPwsxf58efzMoKqi9UYaoo 1KDAqgJQYABKPrsXYTkjWCAtLUS3bcrkEo, 18iW8sKWFt8nHCapHUu83Dt98aPcx6nBjD 12N1zD7RDk9FodVJWGu6mm4JxmoPdwZZFw, 1L4XXFKAWyMV67CCFKVe92i8PEucNrmxRf 186YLrAn5niLLyx3tjwXCcSvEs4T4mqMka, 17HJCxTTwHraHUHVBEiWoc6gBT8MyPNMvh 1MggGuuoTwNRPU1ZvREGWnDQ7jupR8Jqg
Ddraig	1FXLkrpshZ6GnsmErzD9cEyL5pq55wrmeP, 1AoxBFr4SGQL1M66nQspBhUgXicGpJn1pC 12UubZDtvyCEcSmuPm6V5ehkA9wNijsU5o, 1EgnoJodmjtM8EzQkniGpq2mnjH1HncNHG 1EEXxXdDxNBFcGjMdfcqXKLybrDLEck64m, 15yLC6Ji7zHarKwYG9iBUdCUaouBZ3ERmx 1NYs8fC7fU74RifHkWRwAcpmKJhgz2XRUq, 1H1UYS849kh9Z7XyduPcoRMNvTc4tNrerE 1MD6boqjpYczFzAqZKd3Thxhr6efqdL1cZ, 17pQ1H9oNJJnhhiJvp6Q1ThexbZL3yDMnx 1JBZEubRaJHPJ58q39Mn1xcYS2djRyYvEF, 1FJBB9UrtBJERXXFNmnpNu7q22LxDQW8Ci 1K62tR9Z4L88N5EbBR1ezZqxbpYTia88jt
nethoncho	13dUBRzsd96yMdGxSGBjogJ9HqTnegmoY8, 1HQNxDqJNgQpDo9nmxUhGrQtMBCW3uGAoA 13FtQ5UzsumrWSYDooJQNgfrQPRo77eGx7, 1G3iUvugzF8AAve2qPdjgvmXoufur1Ymc5 1AYyKF6ThhTQvMu286ttJARRRPnsShzNvF, 1BCepBALaCMsc3pKSdvXfb3HeN89AcTLsA 1NTa6HFQRTY6Vx6xKSmznJzFrYTW2Fqmr7, 1JT3SLMy5JEy467NfToLY5VxsqDU5CdEHS 1CdASvCQzcspY9CAFeFgwCui4yNiidZgrg, 1F9jY6MvN23jEdfKWLE2iTeUru867ddX7k 1Gg5hoWBiAVyoh4eoWBiV8BEqzNEDNPrLb, 14ywFUR97WUgei2grUAtjWhdEgykbEHG9C
Decksperiment	14T3bituMorqVE7F9kQg59havhakLjCvLf, 1ENqBJ6jRHztzuJxuwysUXAHiBrcGhq6gR 1FtHaYkoyzraCW1aV4a3a3w4eEGwmV2DLZ, 1HbVAkUBzG8yP34mwpvsaC8U47BdchBvCB 1MyFUcDP8wdg8o4wMdkYpJJ6cMYBk59HQz, 1Ez1famUFpHXQqXsXVGnxziaqv28D3Ak16 14QFUbGisbaJgAUTzQSXSrREKMW79EQCdB, 13cQFrLzSgGdV5NJEyBCChCKWHVxLSMJnS 17gAATDgRJJESposqU84haaFketFHG4H6H, 17U4epYKqUmJaeRUThkxbjExFeRVznRnPz 13th1YGpc9efL2B96FXsH2wtrx7ivvdcU2
BitLex	1JvNGADgt4EuRsW8oNr4jGWv7oXnVUrvN7, 14jA9GjFSdrcf5FmHscaWB642Qiz1EUxzo 16XWwEefny7aCXvwm3dDvq42G4nMSyiCv, 1CHKyATFXDxcaNvBCwoLhJwUzfyuXi81oZ 1G7ZTmLhk5wHZChT586Re2PKy8vPofW7SN, 1G8ScB3cTsJTUcNN43yo1xJuUJvLuqajXA 1591zAdaUAs3Y8nP1vQRnq4B6ZkvthLwVg, 12ctdFqSqgdx5CmVEQRGApqFKQzriXDo1y 17X2F2mVcDSf7BpD52jE1so1BwnyoeXyS7, 1YSugRnbzEDDKXksQWTTR77Gx9Rfxsicb 191M2v6y6Bj9Y9DDvsPCWm8bTJor1cWW6M, 1NkeJB646RkkWxuDSAshjHa8uijKJhgoia
theymos	149L5qbzNq4b9H8J87Htkuct5q5CJ8HE8F, 13PkfaZ2FZH6pqCWoJRKCkRd9uanBFKmqx 1Efdncvkckw4ucwudWayS5pHYXTXaR8ALV, 1KEPe91iz352EGNSHbUou49vRqpcR5bmpm 1126xucGhZRJKsmXwuW2zp2aN9pTGEXjFQ, 1GKDUuWBto7Eji5NZ956UizN6ThW6iCd5x 19kYanHx2ZHNAq6Sp8Z7k1sMkFknvTdTaf, 1LswQWg8i3qppi89qcNmakq5gRsanmHwZS 1NXYoJ5xU91Jp83XfVMHwwTUyZFK64BoAD, 1HRYbkiqQ3EEJQwUojz5ShMrd7MNSw4z5d 13nanxDF2PaEGwUKbjW6AuDJQHeTM9nwgr, 1Gb7oKgKZwZhgHopsbXvg1sRzG3umedoZL
dougztr	18pqnPkLYA71V38NxNo29bzGZiHuev46mv, 1C8nEDn3H1PRw6vF3HSKyPjYwfxSJ8L5Yi 15xoAQr3tBT2DZ2QGGrwVVD45wMUx9So9Z, 1GnCPeh639PfVH1o5zKz9FdW5MsTD1Ah94 1ME7usUpFdrHu7pBsvs3e9ZdPdDgCUohvK, 1D3UqLGyEZvFJGnHvPm1hRVC94cpnuEnQr 1DDEqq7A1wVaAwV5twXTH4MvAQVpcJxGWd, 1H3LMcdzx6gWNxTaE3yHCKxiRVNLFNy75p 1JE92gSriZFwHQG4ntFs8uRGb9S9rBeFQB, 1AGSg7UNZamoaoSyjzLvY5Sj9zYuiPj7uU 1LG9BsXXQU2gC29oZdS8XJSi9UwZuMnNyC, 182mDZZyVHMWR957kNNPbnGPkp7XAaX7BK 12TD2L5dUGn5wv1UteBgLKN1jAXRoExyck
jesmurf	1JBqJxtHHFa1srpWKGjq2t41ggiuvPo6x7, 1ieLuJ1ekWHESd2bNjwqnBuyRXHDZZ7Jt 1MvRc5jvqXXZEKuiyVDuDBmsNurpjrvvH5, 18MGoukYpW1tzs8bsNU4XTZGnX1WKZuDFL 197v7EWf3dm71M6kUBBoy45hoRaSHeA6SZ, 1GTLXA2C2ksFbyabsgaAiYMFgzv2DfNSkS 1LGpwDU5djqsR1X14Tcass3y9fULTzxJq3, 1LqVg6yeGvKWCG6y6vX7ABcgTLoNaZgVPN 1DUN6U3ip6pJKxP1LvTYReVwW6bZuzcbz6, 1NSpB23henSKQpYikESM2hrVVMcaAW5xDB 1CP44YjfR9BSAMMuKr5Ay3mEEh3DpnxyEY, 1EtopsUH8kAEKgYx26MmHTpS6Znrv4uoRe
tyler	1NbMtQCuXg65ZpmV87qbJ6Z89G8rySDPEN, 1K75JqLJqJhub5kEyuufKPpGMgKk9HDhR6 1MPqR7VfCky33vsLVsWN5Pyxvofnf61HrV, 1KGrZx8EnQmrsmFZeMXEc5Jyjjn5cz56AT 1LsgMzS3gnCbqg9MTQW5y6Gv58R9FBuXJE, 1LQibbwv16WuMNUhtz1zbLmPViZ1CDvv5C 12zzs7whQYqgLeXKqREp5JdQnktTLz4Ch7, 1PNq5rvVzzyeAdjgjTsd61Rg4yNKndMoTP 1DGfqRVeoFHyZnTTtCrufNByrcdpgUuQ2K, 1EddQ1rUfFniXspmbpLiYdXatqeR6E1Sai 1EKX95ieAk233VefP42aQJ2DJWQgH5o23s
Tilka	1D5J9cEMCR3k5VoKMoNLJsV9gEV19bXNK9, 16G4GxyP45cjtihHUKqjeaWh8WjCjXNsYX 14uv1KMCbHQVUXE1F21fcKxV3KcygUY2BZ, 1L6ttpjnM84oNziCYLBDyrXXXdejniandu 17w1xrZgDmUyJGsr5mEtgz99KSSB1sJToG, 1KozTrBYwjotsx4Cbsz8VNEqAWmCpJhMYt 1AfbZLkYkYGu9hMM3imNb8oKYyNjbypdMK, 1JEJE6Gnx9VkMFqJHQhaNKz939guGZxz66 1JcBMoBxsyxtQABPGcP1p9mXJyQT2nxt69, 14Crcw9XPBgJXEkoyVVAanTTZfasfsaPmY
gebler	13UpRrcu6FoQBYfskyuWRjitfqUL2WJHz2, 123R5yKBomnpAK1AhAmUX6J49Mpy3r2EdQ 1NQVqre5i8G6RePpW17XZtu8upXGbcx4EQ, 1Ev2QBCRbkY7Nxk3bubFsf3AZDRVdNW2X6 1A2yH3nCatqNUA2uAeMkFoMtNh7AF6qfod, 1Bdkxmxad6Y4rW61U3CMx6pWwXkgpeqmxi 18v7a73B6J51HZspbSrDEankmK4m8WmxQ6, 1LFbz9B1eqCPQbxsRq2QmPAp69bYu8cS15 18h6HV7RPU4xFwiap2RdzsqDjcgsEmh3vy, 17eHSiv82cTMqyCPqimuZ318oPSXy6soff 19APmirL5smYUq6PtTELm4ByZ4tHQhy2MN
psokol	177VFGcBfShp6Jz5TYfiww3ahR1KS7Suht, 1PqziDknXMu8KhtRmNQtesKBu2ZjSvRZho 18qWa6zYyTrX3cUDL7DvurLrFvD7yFQU3D, 1ABj7KxfKETKSJhkP95ZryC8AKf7MVsh4 16dgfZ755HGJd7BM7amLWJPWgPVsLjqYZS, 1KmCQsmahQ3Bz6yJuJVQNAztceeNNeKSQe 1CvsvUzbMeWfUKtbCAjF6cbnjgQ8rWjZFb, 1LFhAtoAczdLRSYNWh3WW5oZz5GuXsopZ5 1PA1DeABaTnrz5rzrVzi5E5Nxvp3yK1isd, 1CKUjRaeWEVgLW1YTBtaEhBENW3r42ZEaL

cgt	1FHJhdQY5aEeQst8fCynGSwmBgsssPUMiJ, 1BgZLFJRuktKPwufS4tmQWv6HeWw9VqRvp 1fxD9xNBbu5kc6AxyELiSEz4vH5omiVNt, 18n6kAyJrFT74L3zx66HrUH5ZTkDf2VSHt 19P8sgMct6RgttoPq6Xgbe3vGwBjySvGvQ, 1LYH6KLtFHPaWkNd7chNnHRHhnV5QJ6Ea6 19QxaN39pmiKmyoa5t6CvbUAi8zjZpC4dr, 1KoJbtaAGutpZnvXivFxkteL91zr4KewE1 1Eb2ph7U7wf5NfwN9fS5Fy88gkwfwD59jJ, 158iU2mfNCtmewP8zoRX6HVUjdAb1Yq9Po 14xv8KNFNRHjC8ouN5naKmrTdRpS726UYQ, 1F7MxBzmUf3s235bsPnkYK5s5fmeBeD5AJ 1MCwBbhNGp5hRm5rC1Aims2YFRe2SXPYKt
rodin	16t2SJC7WrfAnQVPEw3MRgydUP79ujxcb4, 1AHGKhM3NNhRhsE8SzqCr6FR7Z69K58M9p 14SivM3UhkjriF4Jo5fV3puBH9Tm3GYGor, 19uK5ZN4T77Lo2XzhYvR4Xdwpdcv8d3Hnq 131bwEe6zWvEpnxGAqxDSqr9nsrrr7GDqH, 1KgakJ21G4msdL2AafyDYBauxVMmCSV2Zv 15vsj4sJbjuGrj78UtSSpuCbXnSku3TWAh, 1DgHxEJyZNVkhDQmUEfUXa5v6SE3DANQ5r 1JtQtqAEMm7tufGtve1LgGrdiYUJRgT1MQ, 1HxDgoD7zUmshQKRQ6Pc7JKuLWagDTYfmm
gst	1YubXSuEhtkhuZuoVqZDyUdFHSQQeqgzF, 1MrxV7NNoAXBgFAYa6KmPFALaG5ppZfQuY 1CDZAN1HpRLUaSr9AdUTiaNft7mHP6ZVPA, 1ALV6ye2zq8z3nnpKi5vbHEZVKnhgaH5F6 1L6Qj1otCcQoYiA8y8B7otDspUxY8jtCJu, 1J2vjRQzdK6W1iq7qkKHAihMyrwRRZzJTm 1L1kS4EunjowgJfgH8rtXqkTDhTqWcsKvn, 17ojUWnQpi52TmN8s7qEPUwa8dJcB2YrRn 17jtufQB79mxJXM5WfsvGrqvgAXyd51NkT, 1K6SvWFkPMhGApxiDMWAX6PPgag5FZ7zLQ
Redajx	1H2rwFfjAhLsAwEZHJcsDtGzPJobiDAYfm, 1N4RpSAehk5kudAM72f45BzHwyu4h9xyBS 1GopSkZdvRjCz51KvnYAKBpBfFWNEcKmrm, 1JFWwuo7jkxNcCS4VumjqehzUKyfZZS9Xi 1LUG1NFLVgXoVFosqnqDDc4EjijGmCZ8hF, 1Dq344XZJEfPPSPfNSVXf3aR3AyDCxVX1u 1L3oHQBY6Z34Fjz7dE11PXENCD7TJNSqrM, 12GP4d5tt9gvfmL18kvty6vaEixkK2MQPx 1HNM585W23ow4b1BfL62c7T6m7BHaTHqpq, 12vCFaGm6UDG3348pPEuiMN8QxEXeddUYg 1G8Um9oJv3Vkk6h9KgAdarA4PSX3y29eqJ, 1NVctpkwn9FL9J6FxSJ4RFJUVGaujL6b9b
Marlsfarp	199oir55jzfTGANV2EvkhJbDkT98s6GoL6, 1HFawzhwXPd6dhMNbziUpxF37pP6jUL5Bc 17w8Jj8vByoupDiRpzzScwJMzS6emRtNdE, 1Jt9EqWnkw6HU7XdVDWUdGkQ2con7MuWVZ 17z5hmJzq2DBeqUTFTJRRNy8XW5SFdxbP1, 1HKaEHY4NDiEV3hgqwtvQSpPAK1YYAFadn 1ET41GtmM24UyqS4zQ547mgrcEoqzc6SiS, 1LFs8K8SBnnzJ5mnRA36vXGugAusJ8aP8v 1HqjicmH5ik4owBrqD9yZs2hVje9jKVJPq, 1KBg7rRZZwzA4AtLCGky1rqG6sX9jTJcWi 12JvLo7BEmPgtHgJ1Sosc4wrug3cAYdHDT
laszlo	17SkEw2md5avVNyYgj6RiXuQKNwkXaxFyQ, 157fRrqAKrDyGHr1Bx3yDxeMv8Rh45aUet 1K1W6YDsVJCys8PNtuHSaG3koxg179WrTL, 19szYHSLRJJY9PqEdvAtKaQB8XbUCFKrms 176cvG1hj4NzhKgg3wBvNfGuuoLDYkhJ6J, 1Gx4uS1wVKobzYxZ6nST7LHRZRZp9RdKSq 149iLTM499LvvktTSbub77dJYvZ7YVGtXo, 1MLh2UVHgonJY4ZtsakoXtkcXDJ2EPU6RY 137PoU6ALX6CFiHPc6XcMrrAU1H9LXWByk, 1XPTgDRhN8RFnzniWCddobD9iKZatrvH4 1MJP3tAHuYyjYQXU7qukzgogdXcV6s1iew, 18kr7ZBLjVq922bQdUHbZF6dYPwMzLUZA1 19tBkza8ZZBxa5zduVurVkZnckHPgJw6XS
The Eternal Coin	16LgJCFHYU85ehaysYkapVXHkfyqqxrsEN, 1NsdHK6tu82ZzLkk8mcYyB2XJzprmL3cDH 1CZg46YMS5QeRBdgPmi7DqmoiLavG2FSES, 1BESg9LmjZ6tAhYjBhUPKRwZVjMLQ5XHKk 1GkoHpFm3mBYCjiMzGTjToTddnU1xWWDgm, 14jhANvgxsqsaaWRJcSAitzDjGTM6XaS4v 1FaKRrqmSrgYb8PusstgDsjk731wBAZy9z, 1E9jfb4ATLyPrFeb1tdRgxaUxj5ev1WoJj 12uxvi2mnKEzWF4G9aCFFQSWWkP5D3TVxA, 1E1epXmoU58cUkGx5AHkmjRiQjTDuhzPQ4
AndrewBuck	1B2rgLHeFCKza3MdhKwTwDfFWiRtXz52pF, 1psJ8QAh3QcpaLrYPSUvmJeRC7P34Sb1V 1Ev4fjXsCTA7r497dZw6DMh7rfbRBDjY8D, 1Nwx4WqdWqWSy2iQN7Tfo7ugzZv56JczZi 1F2suRXWVhpMCx2zXR4KMpeLvjFq3zrC5e, 15hmEKugALo5t6hi6Wg1HExeRins6iCJk5 1GC3Yf4YRZRjBr89kAmjJEjNrWTdJV1mqZ, 1AexxjvNVDhdGP2SPeK9n5R7npsHnyTT3s 13BKFjBgnUnLqnVJohXoDaFAUpkJ5rzveh, 1B9njNHXVpg3mvvCQD3nDVCuVJHK2G8wgs 151HZ8MuwGLHt3hn1pndjRrEfvWB6uaMnM
Xunie	1BRizKk41Lf8cT9WCkEXhwXSMSJiieNBjX, 1GfXKTJF1RcHbyKzVprs4ZeiXatnYQK3Hc 1K1emEabNRqKjU7XQq3Ve7ACBFZKDb4rAe, 1P5YdDzLgnPrfAPYZB926rSVBessmfajzE 13MaitzNCwFeKJ2nxoXGwuyxoNhFyu7Uh6, 15dc63KY1wLpbraxuyFMeUXYrKJticWGcY 1LwbDD7U5hBMtz1oJFw54psuqXczsiYVo, 1LMthpfJDK2FEEgeMwacXn9YVw8Qd6BwVx 1PyNKEyqtDFB4HBG8ueDVxdPZfgwC6gAqw, 1JwgVLbFtQdBLUSKBc6YGDhhXQcMTyDfgo 13jKH6KKyPDJu71gREMSvzu3gqsoKtZe4e, 19wLv1rzwX4GDepWezuM6cMpdEftRR5CAq 14UgTTDsFgFGW7KGwXsTgQ2nRRPmb8yte2
ducki2p	1JWxcPHDVj3HumRpyDsQh1QxzfCNpiMXVm, 1MnQriexjQxWqdVUDTywqqN5fuda78LcTw 1y5rLcwXwZCFSYs69X5jEwWCgzMrKzSof, 15m4ZEGE8WKcu8AACkjzyMDcdbTS4UGcrp 1GySvX5HehecvSSXWQH1Go3VwBWHmFpfWZ, 14ccB2QFEKZnd1gLUtgr7vWg6JrMeXaQXK 1Bh7YQB5L157bJQJv2W3f1TQZyprjLHkRZ, 19pnDxfJjJb6yVmDE9yysp8sQGrNbAhxGz 1B4YpZ2iNK3AnvkTVhS26u2cWyi46gCfj2, 1AyXwEFrHgYXAAQRpHCBegTShffBtKgBS4 1MJk7ZEi2jwP57RdTisKndTp5o9XcjGzQ6
bittard	1GPyWrD3EvhLLD6yZeyVwCDjCPzEFvcEfr, 16v39123LWEKoBDtiMCVii2CxuSRskKkjE 129k2jxhK7jkgkjwRaU2LGDjBbefBfC75i, 1QKL5L2Ebn1JpmpZd72QtB2kvFb8c7EfyW 1Q7PM6ebzDpAkSm53sWnnBjLr1GxvnWMEo, 167ZWTT8n6s4ya8cGjqNNQjDwDGY31vmHg 1EWjpXbaPx7pcZXBugZNZPj76bHEmqD4KC, 1Eb2A9NQP5Mpm9MgNnnFjns15M156cUCJ2 15d6t2KrcEHzQffhf3Kb8vccd9NKJSoTAu, 1JA2e5VFoojFn1ff523xMyc1FFeVGs4PtD
tsr	15jbM6FDwuD1h8nim8nXWvvy7ttdpQNk54, 17EGvD3Mr4xSKU4J1Q7XX4Mqa2gh6H7d8b 1M2UXT8H4EDEawCQ4ocjGyxJGmxGiTkvQz, 175Yq9P2aobM1GPzMgTQqSdPHVevEMbytC 1JkVRKJFVYmtbp8o27vG1nTKG72FnrUY56, 14EP6w8t6cBpRY8BWKGhj1ZaQNsdNzTL5e 15UKjzLirrrrFc3hQXs1Fv9aUhYhBvdhuB, 1BymWiGq8LgmMMf8abYetp6zRJEXv57Ceq 12fAaUq1XzntCFs4f55eP2vt26Z4ioc318, 1Ljx7m8EeN7GFfjZpL184ZLUQEhKKDLVVd 158dg9xTamAUjJVi73cWSXN7eYb8Ka2Q4p
neptop	1CyAj9Bfg2qLJWaVgE4RjeHqjK29GTwKmK, 12XExN24kdnQqUeWJ1L1xLxAitG2wUmvwi 15vZM3C7BpL8DJZ6AP3XATnkzLfiamYt25, 1KkoBgNcJzRbE1ubyu6khKqPkCFtHXapEZ 1Ay5EueexCNrezQzmtoTVUrxexYRvWyEi, 1E25UJEbifEejpYh117APmjYSXdLiJUCAZ 1235KbMjDfnEvqiKiANHTHZEzxQZKJz6vj, 1BpKLUaH4rwjEuzZTxMdr5DcfdJP8JBSwV 1MYLaRVJpTrxo8bv7sd5B4kKe4nphXDRMh, 1K9jyorfJyEqkToVeRRkxUCKkSH8MCH3zo 14w2ChKR3sDRMjv7BV1UpJ1ZwyY6bxvg9v
MrBison	1AUmaj9wpHWd9qW5Jz6J97g3MpQrCP8vC4, 1DDyxvDHNwneRFQx7NMiheKchFRFngdsoB 1PUxeHXsFonZ1vtHQvZPxxUhY76zyTkFKd, 14H9CHDvaunHKSXHBiGdX33hqaiyKzhafR 1JmFmc2EMnkRVbRT2dudwLNattWnWvxcmT, 12yXVPangW66VSg8fZcL6S6QismRJzJy1N 184q9zcmLoSeutdY4gAzWooWX2aQPeE3af, 1GxTUiq2QiJzAVcp45jQTkypCtjxZ7GtKd 142VYfhaYBJXsmfMf1LnBiNxb6KYwdj4PM, 1AbafMpJhVGoJi2ML1eJgBbcucY6CeC4mw 1EdABCHEjE8VoYoNGY2RvXzA8Aiegq1U3J
danknug	1PSwWcVrqLpLCaA8db2C8nEMJGkxfedrbg, 1E5jhuXafyPxi8q5keoZ8ibobMY5RhUqEG 1JWtdasorpxL7HAEzatFLBmraEo1m2f4Fr, 1G83h6vn54k6crA54Wu7tYNpQQJS1J72WA 14NxFv66S3mhtKDAzRk4fiyeWK68pacMkz, 1LLvB65iZeCysdN5QMRrJAvX1ZXrjZQxKU 14u7Wr5QEQwYXXSxpMjtwqbffoHGNg7PnP, 1L7N6erN5CNtPYFVAVLTq7sZww1HEXk8eQ 16bHNPLLwty8kD3TxcgKtbeDehvy2nJTkU, 1HQVuGoCwiue62yiC22Y3PrEAcUBZwb6Vt 1H9An2V3AjXCSMwWjZ82GA4E6tk7c9bs9g, 15NycAoSfkDdJqM2CpAJiS2bsLggUXLdjR 1Nguz4LFSRUvpqn9W4s6mgCbaqWep9s6Ej

Honor Pledge

I pledge my honour that this paper represents my own work in accordance with University regulations.
- Yash Patel