SeqiBloc: Mining Multi-time Spanning Blockmodels in Dynamic Graphs*

Jeffrey Chan, Wei Liu, Christopher Leckie, James Bailey, Kotagiri Ramamohanarao
Department of Computing and Information Systems, University of Melbourne, Australia
{jeffrey.chan, wei.liu, caleckie, baileyj, kotagiri}@unimelb.edu.au

ABSTRACT

Blockmodelling is an important technique for decomposing graphs into sets of roles. Vertices playing the same role have similar patterns of interactions with vertices in other roles. These roles, along with the role to role interactions, can succinctly summarise the underlying structure of the studied graphs. As the underlying graphs evolve with time, it is important to study how their blockmodels evolve too. This will enable us to detect role changes across time, detect different patterns of interactions, for example, weekday and weekend behaviour, and allow us to study how the structure in the underlying dynamic graph evolves. To date, there has been limited research on studying dynamic blockmodels. They focus on smoothing role changes between adjacent time instances. However, this approach can overfit during stationary periods where the underlying structure does not change but there is random noise in the graph. Therefore, an approach to a) find blockmodels across spans of time and b) to find the stationary periods is needed. In this paper, we propose an information theoretic framework, SeqiBloc, combined with a change point detection approach to achieve a) and b). In addition, we propose new vertex equivalence definitions that include time, and show how they relate back to our information theoretic approach. We demonstrate their usefulness and superior accuracy over existing work on synthetic and real datasets.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data Mining; E.1 [**Data Structures**]: Graphs and Networks

General Terms

Algorithms

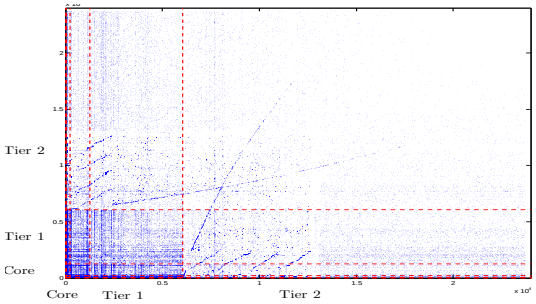
*This research was supported under Australian Research Council's Discovery Projects funding scheme (project number DP110102621).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
KDD'12, August 12–16, 2012, Beijing, China.
Copyright 2012 ACM 978-1-4503-1462-6 /12/08 ...$10.00.

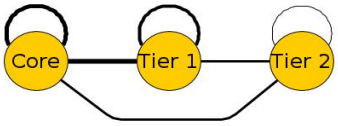
Keywords

Blockmodel, Dynamic Graphs, Minimum Description Length, Structural Equivalence

1. INTRODUCTION



(a) Adjacency matrix and the blockmodel decomposition. Red dotted lines delimit the boundaries of each role. The 3 roles are Core, Tier 1 and Tier 2.



(b) Image diagram of the 3 roles and their role to role relationships.

Figure 1: Blockmodel decomposition of a BGP connectivity graph over November 2011.

Blockmodelling has been studied for many years in the social sciences [16]. Blockmodelling is a powerful approach to decompose social networks and graphs into partitions and common roles. Vertices playing the same role, or are equivalent, if they have similar connections to other vertices. These roles, along with the role to role interactions, summarises the underlying structure of a graph succinctly. For example, consider Figure 1, which shows a blockmodel decomposition of the Internet routing graph in November 2011. Figure 1a shows the adjacency matrix of the graph, along with the partition into roles, delimited by the red dotted lines. We label the roles into 'Core', 'Tier 1' and 'Tier 2'. The decomposition shows the Core vertices are highly connected among themselves and to vertices in other roles. Tier 1 vertices are highly connected among themselves and con-

nected to some Tier 2 vertices. The overall structure is a core-periphery, which can be succinctly summarised by the image diagram in Figure 1b, which shows the roles as vertices, and the inter role interactions as edges (the amount of connectivity between roles is expressed by the thickness of the edges in the image diagram). As can be seen, blockmodels summarise the structure of a graph succinctly, allowing us to understand and characterise the underlying structure (e.g., it is a core-periphery), discover the important roles (e.g., the Core role is highly connected to all other roles and hence routes most of the traffic) and compare with other graphs.

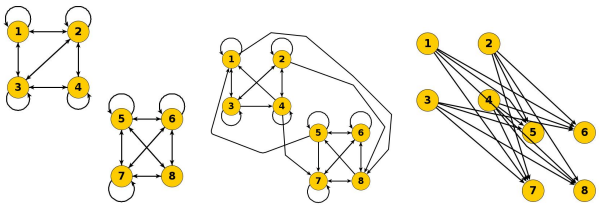
A community decomposition seeks to find roles that are densely connected within themselves and sparsely connected to vertices of other roles. Such a type of decomposition is not able to find the structure in Figure 1b. For example, the connections between the Core vertices to the Tier 1 vertices are more dense than among the Tier 1 vertices, so it is likely that the Core and Tier 1 vertices are incorrectly merged into one role. Blockmodelling, on the other hand, can find community decompositions, and hence, it can be considered as a generalisation of community finding for graphs.

The vertex equivalence used in the previous example is called *structural equivalence*; there are several others, but we concentrate on this type of equivalence in this paper as it is the most common and easiest to interpret. Equivalent vertices have similar in and out neighbours. For example, the vertices labelled 2 and 3 in Figure 2a are structurally equivalent because they have the same out neighbours (1,2,3,4) and the same in neighbours (1,2,3,4).

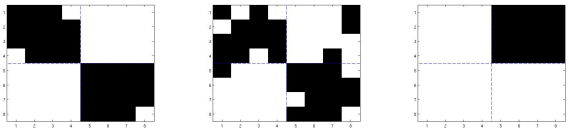
As the underlying graphs are not static, it is essential to fit and track blockmodels across time. For example, in the dynamic graph representing the email communications in Enron, it is known that during the crisis, communication patterns and intensity among different employee roles experienced large shifts [8]. Hence if we used a single blockmodel to describe the whole sequence we will not detect or see these changes.

There are several open challenges in fitting dynamic blockmodels. Unlike the case of static blockmodelling where there is a strong formalism between the vertex equivalences and what type of blockmodels will satisfy those equivalences [16], there are no equivalence definitions for dynamic blockmodels. These formalisms are important, because they provide the theoretical underpinnings between (intuitive) definitions of vertex equivalences (i.e., what defines a group of vertices playing the same role) and how they relate to structure seen in the adjacency matrix. In this paper, we introduce two formal definitions of evolving structural equivalence and provide lemmas on how this is reflected in the block structure of a blockmodel decomposition.

In addition, dynamic blockmodels have only been recently studied [17]. In these works, each snapshot in the dynamic graph is modelled by a blockmodel, with smoothing applied on the membership of the roles (for the rest of the paper, we will use roles and positions, which is the term from social sciences, interchangeably). However, this might not be the optimal model to represent a dynamic graph. Consider the following scenarios, using the Enron email network as an example. If we used a single blockmodel to represent the whole graph sequence, it will be the simplest blockmodel, but it will be unlikely to model the communication changes inside Enron. On the other hand, if we represented each snapshot



(a) T1 Snapshot. (b) T2 Snapshot. (c) T3 Snapshot.



(d) T1 Adj matrix. (e) T2 Adj matrix. (f) T3 Adj matrix.

Figure 2: A dynamic graph example of 3 snapshots. The top and bottom rows illustrate the snapshots and corresponding adjacency matrices respectively.

with its own blockmodel, this will accurately model the underlying dynamic graph, but at the expense of overfitting. Hence, it is desirable to find a balance between accuracy and generality. During periods where the edges within the graph undergo minor noisy fluctuations, it is better to use one blockmodel to represent these periods. On the other hand, when the underlying graph is going through a fundamental shift, then using multiple blockmodels to represent this period might be preferable. This shows the importance of having a) an approach to quantify and find a tradeoff between model complexity and model accuracy; b) a method to fit blockmodels over subsequences of snapshots and c) a way to find where best to segment a graph sequence.

In this paper, we present an information theoretic approach to address these challenges. Using the minimum description length principle [14] to quantify the tradeoff between model complexity and accuracy, we propose four different encoding schemes that are used as objectives for finding blockmodels over subsequences, for our two proposed definitions of evolving structural equivalence. In addition, we propose a new, fast, change point detection based approach and a new blockmodel comparison measure to segment the graph sequence. Using generated and three real datasets, we show that our approach can accurately determine good locations to segment the dynamic graph sequence and find interesting dynamic blockmodels. We call our framework SeqiBloc (**S**ubs**eq**uence **I**nformation Theoretic **Bloc**kmodelling).

We present the following contributions in this paper:

- We introduce two formal definitions of evolving structural equivalences, describe when they are applicable, and show their corresponding matrix form.
- We propose four different information theoretic encodings that quantify the tradeoff between model complexity and accuracy and conform to the new equivalence definitions.
- Using our encoding schemes, we propose a blockmodel comparison measure and a fast and accurate change point detection based segmenting approach to determine when new blockmodels need to be learnt.

2. RELATED WORK

In this section, we describe related work in static and dynamic blockmodelling and finding dynamic communities.

Blockmodelling was initially introduced in sociology to model social networks [16]. Later, stochastic blockmodels [2] were proposed to relax the presumption of exact vertex equivalences. Edges, the position of vertices and other variables are modelled as random variables. The assumptions made about the variables modelled, the dependencies between random variables and the parametric distributions of the probabilities lead to different formulations and probabilistic models. For example, Airolid et al. [2] introduced a model that allowed vertices to belong to multiple positions In [17], Xing et al. extended the mixed membership model of [2] to dynamic networks, where there is one blockmodel per snapshot. The positions of vertices are allowed to change with time and be smoothed between adjacent snapshots. This is similar to one of the dynamic equivalence definitions we propose. To improve the fitting complexity, Yang et al. [18] performed the fitting using Bayesian inference.

Statistical approaches can infer insightful models, but their fitting process can be computationally demanding. More importantly, the dynamic blockmodel techniques fit one blockmodel per snapshot, which is likely overfit.

There are a variety of approaches proposed for finding and tracking communities across time. Initially, static methods were applied to each individual snapshot and the amount of overlap between the communities of adjacent snapshots were used to categorise community events such as growth, decay and stability [11]. Subsequent approaches used the idea of smoothing from evolutionary clustering [4] to find communities that are good for the current snapshot but also close to the communities found from the previous snapshot. Work in this area includes [5], which extended static spectral community finding to this framework.

Dinh et al. [9] proposed an incremental algorithm to update existing communities when a new snapshot arrives. Similarly, [1] introduced efficient ways to maintain a set of communities in massive graph streams. The emphasis is on incrementally maintaining the best current set of communities, not on finding communities across a subsequence.

Algorithms for finding dynamic communities generally do not work with dynamic blockmodels, as finding communities is only one type of blockmodel and there are many other interesting types, like the core-periphery structure of the BGP example.

To the best of our knowledge, only [15] represents a graph sequence as a series of models. Graphscope [15] uses a minimum description length coding formulation to find evolving co-clusters in bi-partite graphs. The sequence of graphs is partitioned into segments where a single co-clustering holds. This is similar to our approach, and as a baseline we have adapted Graphscope to finding unipartite blockmodels. However, the original Graphscope can only find one type of dynamic equivalence and its greedy lookahead segmenting approach misses many segmenting points in synthetic and real datasets. In addition, the blockmodels it found for real datasets have an unreasonably high number of positions. In contrast, we propose four different encodings to find our proposed dynamic equivalences, and our change point detection approach can accurately segment the graph sequences and find blockmodels that have a reasonable number of positions.

Symbol	Description
G	A graph with vertex set $V$and edge set E.
$<G^{ts,te}>$	A continuous graph sequence of snapshots.
v_i	A vertex.
$e_{i,j}$	An edge from vertex v_i to v_j.
$\mathbf{A}_t$	Adjacency matrix of G^t.
n	Number of vertices.
$\mathcal{C}^L$	A set of vertex positions with index L.
C_i	A vertex position.
$\mathbf{A}^t(C_i, C_j)$	A block with rows from C_i and columns from C_j, from graph snapshot G^t.
$m_1(\mathbf{A}^t(C_i, C_j))$	The number of 1's in block $\mathbf{A}^t(C_i, C_j)$.
H(X)	Entropy of random variable X.

Table 1: Description of symbols used in this paper.

3. BACKGROUND

In this section, we formally introduce the ideas of blockmodelling and structural equivalence, their dynamic variants, and the notation we use in this paper (see Table 1).

A **graph** G consists of a set of vertices V, and a set of edges E, $E : V \times V$. In this paper, the graphs we study are uniquely labelled and the set of vertices do not change (we leave this to future work). A graph can also be modelled by its adjacency matrix $\mathbf{A}$. A **dynamic graph** is represented as a sequence of snapshots $< G^{1,T} >=< G^1, \ldots, G^t, \ldots, G^T >$, $1 < t < T$ (T can be ∞).

3.1 Blockmodelling

We start by describing the traditional, static definition of blockmodelling. Recall that a blockmodel partitions the vertices of a graph into a set of positions, based on notions of vertex equivalence. The positions in turn divide the adjacency matrix of the graph into a set of blocks. Each block defines the edge interactions between two positions.

Let the set of positions be denoted by $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$, e.g., C_1 in Figure 2d is $\{1, 2, 3, 4\}$. A block $\mathbf{A}(C_i, C_j)$ is a submatrix of the adjacency matrix $\mathbf{A}$, and represents the edges from the vertices of position C_i to the vertices of position C_j. An example block is $\mathbf{A}(C_1, C_1)$ from Figure 2d, which occupies the top left of the adjacency matrix.

A **blockmodel** is defined as $\mathbf{B}(\mathbf{A}, \mathcal{C})$, where the rows and columns of $\mathbf{A}$ are rearranged into blocks according to the set of positions $\mathcal{C}$, such that the vertices in the same position are (structurally) equivalent.

3.2 Evolving Blockmodels

As explained in Section 1, evolving structural equivalence has not been explored in the literature hence it is not clear what it means to have dynamic structural equivalence. Therefore, in this section, we propose and define two definitions of dynamic structural equivalence.

The most strict definition, **block preserving structural equivalence**, asserts that two vertices are equivalent over a span of time if they have the same neighbours over that span of time.

Definition 1. Two vertices v_i and v_j are **block preserving structurally equivalent** over the continuous time span of $[ts, te]$, where $ts \leq te$, if for t and t', where $ts \leq t \leq te, ts \leq t' \leq te$, and for $v_k, v_l \in V$:

1. $e^t_{i,k} \in E^t$ iff $e^{t'}_{j,k} \in E^{t'}$ and $e^t_{l,i} \in E^t$ iff $e^{t'}_{l,j} \in E^{t'}$;

2. $e^t_{i,k} \notin E^t$ iff $e^{t'}_{j,k} \notin E^{t'}$ and $e^t_{l,i} \notin E^t$ iff $e^{t'}_{l,j} \notin E^{t'}$.

For example, in Figure 2, vertices 2 and 3 are block preserving equivalent over T1 and T2, but not T3, because their neighbourhood in T3 is different from T1 and T2. Next, we relate this definition of equivalence with the density of the blocks of the corresponding blockmodel.

LEMMA 1. *Let a block consisting of all 1s be denoted by* $\mathbf{1}$*, and all 0s be denoted by* $\mathbf{0}$*. Let* V *be partitioned into block preserving structurally equivalent positions,* $\mathcal{C}$*, over the span* $[ts, te]$*. Then for all* t, t'*,* $ts \leq t, t' \leq te$*, and* $\forall C_x, C_y \in \mathcal{C}$

1. $\mathbf{A}^t(C_x, C_y) = \mathbf{1}$ *or* $\mathbf{0}$*;*
2. $\mathbf{A}^t(C_x, C_y) = \mathbf{A}^{t'}(C_x, C_y)$.

The inverse direction holds true also.

PROOF. Consider the vertices v_i and $v_j \in C_x$ and $v_k \in C_y$, $\forall C_x, C_y \in \mathcal{C}$. Since v_i and v_j are equivalent, then they must either both have an edge, or both have no edge to v_k over all t. This means condition 1, and since it is over all t, condition 2 of the lemma are both true. To show the inverse direction, consider a block $\mathbf{A}^t(C_x, C_y)$. If it is equal to $\mathbf{1}$, then there is an edge from all vertices $v_i \in C_x$ to all vertices $v_j \in C_y$. As the blocks are equal across all t, then the edge $e_{i,j}$ must exist over all t, satisfying first part of condition 1 of Definition 1. Similar reasoning can be used to show the second part of condition 1 and condition 2 of Definition 1 ($\mathbf{A}^t(C_x, C_y) = 0$ case) holds also. □

The second definition, **position preserving structural equivalence**, asserts that two vertices are equivalent over a span of time if for every time instance in the span, the two vertices are structurally equivalent. Note that the set of neighbours of equivalent vertices can be different at different time instances, but must be the same at the same time instance. For example, vertices 2 and 3 are position preserving equivalent over T1 to T3 in Figure 2. This equivalence is useful for finding dynamic equivalences that might not maintain the same connectivity throughout a span of time.

For example, we could represent the replying between a group of experts and newbies in a Q&A forum as a dynamic graph with one snapshot per hour. During the day, the experts are active, but they go to sleep at night. Using a position preserving structural equivalence definition, the experts are considered equivalent throughout a 24 hour period because they are structurally equivalent for each snapshot. They are still regarded as experts when they go to sleep. However, applying a block preserving definition, their behaviour means they are equivalent in two subsequences - one for the day, when they are highly active, and one for night, when they are inactive and likely to have the same replying behaviour as newbies (and be equivalent to them). This equivalence asserts that the experts are only considered experts when they are actively answering questions.

Definition 2. Two vertices v_i and v_j are **position preserving structurally equivalent** over the continuous time span of $[ts, te]$, where $ts \leq te$, if $\forall t, ts \leq t \leq te$, and for $v_k, v_l \in V$:

1. $e^t_{i,k} \in E^t$ iff $e^t_{j,k} \in E^t$ and $e^t_{l,i} \in E^t$ iff $e^t_{l,j} \in E^t$;
2. $e^t_{i,k} \notin E^t$ iff $e^t_{j,k} \notin E^t$ and $e^t_{l,i} \notin E^t$ iff $e^t_{l,j} \notin E^t$.

LEMMA 2. *Let* V *be partitioned into position preserving structurally equivalent positions,* $\mathcal{C}$*, over the span* $[ts, te]$*. Then for all* t*,* $ts \leq t \leq te$ *and* $\forall C_x, C_y \in \mathcal{C}$*,*

1. $\mathbf{A}^t(C_x, C_y) = \mathbf{1}$ *or* $\mathbf{0}$.

The inverse direction holds true also.

PROOF. The proof is similar to Lemma 1 and due to space limitations, we omit it. □

4. INFORMATION THEORETIC BLOCKMODELLING

In this section, we describe our information-theoretic framework to evaluate how well blockmodels conform to our proposed equivalence definitions.

Given data and a model, the idea is to evaluate a model by the amount of compression it can achieve on the data against how complex the model is. The more compression that is achieved, the lower the cost to encode the model, and the more accurate the model is. This is balanced against the complexity of the model (the more complexity, the higher the encoding costs). Using the minimum description length principle, we can combine the two ideas as C(graph, position) = C(position) + C(graph | position). A blockmodel (described by its set of positions) is the best tradeoff between accuracy and complexity if it minimises this expression.

In this work, we assume an unweighted, directed graph representation. Note that we can extend our representation to weighted graphs; we describe this in Section 7. Next, we describe the different encoding schemes.

4.1 Individual Encodings

As a baseline and for our change point detection approach, we first present two encodings that encode each snapshot of a graph sequence with a blockmodel.

4.1.1 Individual Snapshot Encoding

Let $\mathcal{C}^t$ denotes the set of positions (blockmodel) for snapshot G^t. Then the total encoding cost is:

$$C_{\text{Ind}}(< G^{1,T} >, \{\mathcal{C}^1, \ldots \mathcal{C}^T\}) = C(n) + \sum_{t=1}^{T} C(\mathcal{C}^t) + C(G^t|\mathcal{C}^t)$$

where $C(n)$ is the cost to send the number of vertices (needed to decode the sizes of the positions), $C(\mathcal{C}^t)$ is the cost to encode the positions, and $C(G^t|\mathcal{C}^t)$ is the cost to encode the snapshot G^t using $\mathcal{C}^t$.

Next, we describe the position description cost $C(\mathcal{C}^t)$. Let the random variable $\Psi(\mathcal{C})$ represent the distribution of the position memberships of $\mathcal{C}$; i.e., $p(\Psi(\mathcal{C}) = i) = \frac{|C_i|}{n}$, where $C_i \in \mathcal{C}$. Let H(X) denote the entropy of random variable X. From information theory [7], we know we can design lossless codes that uses $H(\Psi(\mathcal{C}^t))$ bits, on average, to encode the position of a vertex, and $n * H(\Psi(\mathcal{C}^t))$ to encode the positions of all vertices. In order to be able to recover the memberships fully, we also need the number of positions, which can be encoded as an integer [7] with cost $C(|\mathcal{C}^t|)$. Then the cost to send the position information is : $C(\mathcal{C}^t) = C(|\mathcal{C}^t|) + n *$ H($\Psi(\mathcal{C}^t)$).

Next, we explain how to compute $C(G^t|\mathcal{C}^t)$. Given a blockmodel, the snapshot can be described as a sequence of its blocks[1].

[1]In row or column order - we use row order

$$C(G^t|\mathcal{C}^t) = \sum_{C_i \in \mathcal{C}^t} \sum_{C_j \in \mathcal{C}^t} C(\mathbf{A}^t(C_i, C_j))$$

We can treat each timesliced block as a vector, and encode it using single symbol encoding. Let $m_1(\mathbf{A}^t(C_i, C_j))$ denote the number of 1's in the block $\mathbf{A}^t(C_i, C_j)$. Let the random variable $\Phi(\mathbf{A}^t(C_i, C_j))$ represent the distribution of 0's and 1's in the block $\mathbf{A}^t(C_i, C_j)$; i.e, $p(\Phi(\mathbf{A}^t(C_i, C_j)) = 1) = \frac{m_1(\mathbf{A}^t(C_i, C_j))}{|C_i|*|C_j|}$. Then the cost to send one block is:

$$C(\mathbf{A}^t(C_i, C_j)) = C(m_1(\mathbf{A}^t(C_i, C_j))) + |C_i| * |C_j| * H(\Phi(\mathbf{A}^t(C_i, C_j)))$$

Again, we need the number of edges (1's) in the block, sent with cost $C(m_1(\mathbf{A}^t(C_i, C_j)))$, in order to fully reconstruct the edges in the block.

4.1.2 Smoothed Individual Encoding

This model is based on ideas from evolutionary clustering, to smooth out position changes between two consecutive snapshots. Similar to the individual snapshot encoding, we can write the overall encoding cost as:

$$C_{\texttt{sInd}}(< G^{1,T} >, \{\mathcal{C}^1, \ldots \mathcal{C}^T\}) = C(n) + C(\mathcal{C}^1) + \sum_{t=2}^{T} C(\mathcal{C}^{t-1}|\mathcal{C}^t) + C(G^t|\mathcal{C}^t)$$

The only term that is different from the individual snapshot encoding is $C(\mathcal{C}^{t-1}|\mathcal{C}^t)$, which is the encoding cost of describing the changes in position between $\mathcal{C}^t$ and $\mathcal{C}^{t-1}$. This can be encoded using a membership difference vector of length n, where we can encode each symbol with an average cost of $\mathrm{H}(\Psi(\mathcal{C}^t) \mid \Psi(\mathcal{C}^{t-1}))$ bits:

$$C(\mathcal{C}^{t-1}|\mathcal{C}^t) = C(|\mathcal{C}^t|) + n * H(\Psi(\mathcal{C}^t)|\Psi(\mathcal{C}^{t-1}))$$

4.2 Subsequence Encodings

These encodings are used to describe blockmodels that span more than one snapshot and measure how close they are to being block preserving and position preserving equivalent. The graph sequence is represented as a series of continuous subsequences, where a single blockmodel holds for each subsequence.

4.2.1 Block Preserving Encoding

This encoding scheme is designed to find block preserving equivalent blockmodels. Let the subsequence be delimited by the timing indices $t_1, t_2, \ldots, t_L$, where $t_1 < t_2, \ldots, < t_L$ (with $t_1 = 1$ and $t_L = T$) and L is the number of subsequences up to time index T. Then the graph sequence $<G^{1,T}>$ is divided into a series of subsequences, $< G^{t_1,t_2} > \cdot < G^{t_2,t_3} > \ldots < G^{t_{L-1},t_L} >$. Let $\mathcal{C}^s$ represent the set of positions for the subsequence $< G^{t_s,t_{s+1}} >$, $1 \leq s \leq T-1$. The the overall encoding cost is:

$$C(< G^{1,T} >, \{\mathcal{C}^1, \ldots \mathcal{C}^L\}) = C(n) + \sum_{s=1}^{L} C(t_{s+1} - t_s) + C(\mathcal{C}^s) + C(< G^{t_s,t_{s+1}} > |\mathcal{C}^s)$$

Most of the cost terms are the same as the individual encodings, apart from $C(t_{s+1} - t_s)$, which is the cost to encode the length of the subsequence $<G^{t_s,t_{s+1}}>$, and the block encoding cost $C(< G^{t_s,t_{s+1}} > |\mathcal{C}^s)$. Let $\mathbf{A}^{t_s,t_e}(C_i, C_j)$ denote the multi-time slices of the adjacencies of the snapshots G^{t_s} to G^{t_e}, between the positions C_i and C_j. Then the graph description cost is:

$$C(< G^{t_s,t_{s+1}} > |\mathcal{C}^s) = \sum_{C_i \in \mathcal{C}^s} \sum_{C_j \in \mathcal{C}^s} C(\mathbf{A}^{t_s,t_{s+1}}(C_i, C_j))$$

Each multi-time sliced block is encoded as one string. An useful way to think of this is that we unroll each block of each snapshot into a string. The strings are concatenated and then encoded as a single string. The cost is:

$$C(\mathbf{A}^{t_s,t_{s+1}}(C_i, C_j)) = C(m_1(\mathbf{A}^{t_s,t_{s+1}}(C_i, C_j))) + |C_i| * |C_j| * H(\Phi(\mathbf{A}^{t_s,t_{s+1}}(C_i, C_j)))$$

When $C(< G^{t_s,t_{s+1}} > |\mathcal{C}^s)$ is minimal ($H(\Phi(\mathbf{A}^{t_s,t_{s+1}}(C_i, C_j))) = 0$, $\forall C_i, C_j \in \mathcal{C}^s$), then the multi-sliced blocks must be all 0s or all 1s. This means the two conditions of Lemma 1 are satisfied, which means the decomposition is block preserving equivalent. Therefore this encoding is a measure of block preserving equivalence.

4.2.2 Position Preserving Encoding

This encoding implements the position preserving equivalence. The overall encoding cost formulation is similar to the block preserving encoding, except for how the multi-time block slices are encoded:

$$C(\mathbf{A}^{t_s,t_{s+1}}(C_i, C_j)) = \sum_{t=t_s}^{t_{s+1}} \left(|C_i| * |C_j| * H(\mathbf{A}^t(C_i, C_j)) + C(m_1(\mathbf{A}^t(C_i, C_j))) \right)$$

The blocks are encoded per snapshot. Again, $C(\mathbf{A}^{t_s,t_{s+1}}(C_i, C_j))$ is minimal when $H(\mathbf{A}^t(C_i, C_j)) = 0$, which occurs when the position preserving condition of Lemma 2 is true, hence this encoding is a measure of position preserving equivalence.

5. FINDING OPTIMAL BLOCKMODELS

In this section, we describe our common approach to optimising the encodings and finding optimal blockmodels. We use a similar approach to [15] to optimise our encodings over subsequences, but we introduce a change point detection approach to segmenting the graph sequence, which we will show is more accurate and faster than the one lookahead approach of Graphscope.

Given a subsequence segment, the greedy algorithm approach used to optimise the positions will iteratively try to merge positions, split them and move vertices between positions to minimise the encoding cost.

5.1 Determining the Optimal Segments

Graphscope determines a change point by comparing the encoding costs of extending the existing, single model against the costs of two models, the existing model up to the change point and a new model for describing the rest of the sequence (the lookahead subsequence). Graphscope uses a lookahead of 1 snapshot, and as our results show, it can sometimes miss change points. Once a change point is missed, the objective generally favours extending the current subsequence,

Algorithm 1 Main procedure in SeqiBloc.

1: **Input: New snapshot G^t, Current subsequence $<G^{ts,te}>$, prev. blockmodel $\mathcal{C}^{t-1}$**
2: **Output:** $\mathcal{C}^{ts}$
3: // Compute partitioning on single snapshot
4: $\mathcal{C}^t$ = updatePos(G^t)
5: deltaCost = $M(\mathcal{C}^t, \mathcal{C}^{t-1})$
6: **if** isChangePoint(deltaCost)) **then**
7: $\quad\mathcal{C}^{ts}$ = updatePos($<G^{ts,t}>$)
8: **end if**
9: updateTimeSeries(deltaCost)

because the lookahead is too short. Hence, a greater lookahead is needed, but there is no principled and inexpensive way to determine the appropriate lookahead, and the optimal lookhead is likely to change across time.

Instead, we propose a simple change point detection approach (see Algorithm 1 for an overview). First, note that a new blockmodel should only be induced when there is a change point and the existing one is no longer accurate for describing the new snapshots. This means that the optimal blockmodel over the new snapshots varies significantly from the existing blockmodel. Conversely, when there is no change point and only (uniformly) random noise, then the likelihood of dense or sparse sub-blocks appearing by chance is low and therefore the blockmodel over the new snapshots is likely to be similar to the existing blockmodel. Therefore, if we found the best blockmodels for each snapshot (using one of the individual encodings), and monitored the differences between consecutive blockmodels (see next section), then the actual change points are likely to occur when there are significant spikes in the difference, which can be detected using traditional change point detection methods.

5.1.1 Blockmodel Difference Measures

In this subsection, we describe the two measures we used to determine the change points in the graph streams.

To detect the change points of position preserving blockmodels, we monitor for differences in the positions of two time-adjacent blockmodels. Existing measures from external clustering validation can be used to measure this position difference. We found that the Variation of Information (VI) measure [13] performed well for this task. We denote this change point approach with the VI measure as *pos-cp*.

To detect change points in block preserving blockmodels, we know from Lemma 1 that we need to monitor for changes in the positions and the block densities. There are no measures that satisfy our requirements; the closest are spatially aware cluster comparison techniques [6], but these assume the entities are points embedded in a Euclidean space, which is generally non-trivial to map from graphs.

Hence, we propose a measure, called BMDD that compares the differences in densities across the blocks of the two blockmodels. We cannot directly compare blocks, because the positions and blocks between two models might not align. Instead, we compute the expected difference in densities across the blocks of the two models. Recall that $\Psi(\mathcal{C})$ is the random variable of the position memberships and event space of $\mathcal{C}$. Let $d()$ denote a distance between densities (we will elaborate shortly) and the probability $p(\Psi(\mathcal{C}^t) = i, \Psi(\mathcal{C}^t) = j, \Psi(\mathcal{C}^{t-1}) = x, \Psi(\mathcal{C}^{t-1}) = y)$ be simplified to $p(i, j, x, y)$. Then BMDD is defined as

$$
\begin{aligned}
&BMDD(\mathcal{C}^t, \mathcal{C}^{t-1}) \\
&= \sum_{r1,c1,r2,c2} p(r1, c1, r2, c2) d(\mathbf{A}^t(C_{r1}, C_{c1}), \mathbf{A}^{t-1}(C_{r2}, C_{c2})) \\
&= \sum_{r1,c1,r2,c2} p(r1, r2) p(c1, c2) d(\mathbf{A}^t(C_{r1}, C_{c1}), \mathbf{A}^{t-1}(C_{r2}, C_{c2}))
\end{aligned}
$$

where $p(r1, r2) = \frac{|C_{r1} \cap C_{r2}|}{n}$ and $p(c1, c2) = \frac{|C_{c1} \cap C_{c2}|}{n}$. BMDD effectively weighs the difference between block densities based on the amount of overlap the two blocks have.

For the density distance dd, we used the absolute difference of their densities $d(\mathbf{A}^t(C_{r1}, C_{c1}), \mathbf{A}^{t-1}(C_{r2}, C_{c2})) = |\mathbf{A}^t(C_{r1}, C_{c1}) - \mathbf{A}^{t-1}(C_{r2}, C_{c2})|$, which we found to work well. We denote this change point approach with the BMDD measure as *pos-cp*.

5.1.2 Change Point Detection Approaches

The change point we seek is a large, significant spike. There are many different change point measures we can use (our framework can accommodate any that can detect spikes), but we found the basic control chart measure [3] worked well. The control chat measure tracks the mean and standard deviation of our blockmodel difference measures, and flags a change point when the standard deviation is significantly ($\Delta * \sigma$, where Δ is the alarm threshold) above the mean. Because we do not make any assumptions on the underlying graph stream, we cannot use techniques like generalised likelihood tests to quantify significance. Hence, the alarm threshold is the only parameter in our approach. We found $\Delta = 0.5$ works well for many datasets, and this can be used as the default value.

6. EVALUATION

In this section, we will show that our change point detection based algorithms, *pos-cp* and *pos-cp*, are more accurate and efficient than the corresponding Graphscope algorithms at detecting change points and learning new blockmodels. In addition, we demonstrate the difference between the block and position preserving formulations. We use a combination of synthetic data and three real datasets.

We evaluate against two baseline Graphscope algorithms, *block-gs* and *pos-gs*. *block-gs* and *pos-gs* use the one-lookahead approach of Graphscope to optimise the block preserving and position preserving encodings respectively.

6.1 Synthetic Evaluation

To construct the synthetic graphs, we use a static stochastic blockmodel algorithm [2] to generate initial static blockmodels of 250 vertices, 10 uniformly sized positions, 30% of blocks having 75% density and rest having 5% density. We then introduced change points at 10 uniformly random locations, where we change the underlying blockmodel by moving vertices between positions, flipping the densities of some blocks (e.g., 0.75 to 0.25) or adding or deleting a position. Each subsequence (of 100 snapshots) is then generated from a static blockmodel, with some uniformly distributed noise (flipping of edges) added to the snapshots between the change points. For each parameter setting, we generated three blockmodels, and from each, three subsequences, and ran each of the algorithms three times, for a total of 27 runs per algorithm for each dataset[2] setting.

[2]Available at `http://eng.unimelb.edu.au/jeffreyc/data`

Algor.	Precision	Recall	VI / sshot
pos-cp	1.000(0.000)	0.733(0.267)	4.101(1.931)
pos-cp	0.626(0.000)	1.000(0.000)	3.990(1.992)
block-gs	0.500(0.000)	0.933(0.047)	4.956(0.936)
pos-gs	0.000(0.000)	0.000(0.000)	5.035(0.843)

(a) Results for datasets with 5% noise, and an average of 1 position addition/deletion and 5% position membership changes at the change points.

Algor.	Precision	Recall	VI / sshot
pos-cp	0.222(0.416)	0.033(0.067)	5.550(0.357)
pos-cp	0.717(0.081	0.989(0.031)	5.529(0.442)
block-gs	0.500(0.000)	0.900(0.000)	5.677(0.242)
pos-gs	0.000(0.000)	0.000(0.000)	5.689(0.148)

(b) Results for datasets with 5% noise, and no position additions/deletions and 5% block density flips at the change points.

Algor.	Precision	Recall	VI / sshot
pos-cp	1.000(0.000)	1.000(0.000)	5.663(0.227)
pos-cp	0.627(0.023)	1.000(0.000)	5.739(0.257)
block-gs	0.500(0.000)	0.900(0.082)	5.794(0.184)
pos-gs	0.000(0.000)	0.000(0.000)	5.792(0.190)

(c) Results for datasets with 5% noise, and no position additions/deletions and 5% position membership changes at the change points.

Table 2: Segmenting results for generated datasets. Results are reported as average (std. deviation).

6.1.1 Measures

To evaluate the ability of the algorithms to detect the introduced change points, we use the precision and recall measures. Let I^{true} and I^{act} denote the set of true and detected change points respectively. Then precision $= \frac{|I^{true} \cap I^{act}|}{|I^{act}|}$ and recall $= \frac{|I^{true} \cap I^{act}|}{|I^{true}|}$. We measure how similar the detected and generated blockmodels are by comparing their sets of positions across the snapshots and averaging the results. We use the average variation of information (VI) per snapshot as our comparison measure. Lower VI means more similar sets of positions. Although the mean of the total objective values for the change point algorithms are 20–30% lower than the corresponding Graphscope variants, we do not report these values because we found they vary greatly between the datasets (large variance) and hence are inconclusive for our analysis.

6.1.2 Results

In this subsection, we evaluate the segmenting accuracy of SeqiBloc. We first evaluate its ability to locate the introduced change points, varying the amount of noise and the size of the blockmodel shift at the change points.

We first changed the amount of noise introduced from 1% of edges to 10% of edges. We found that the algorithms were mostly invariant to the amount of noise introduced, hence we only show one of the results (5% of edges changed) in Table 2a. The results indicate that when *pos-cp* detects a change, it is always correct in these tests (100% precision), but it can sometimes miss some change points (73% recall). On the other hand, *pos-cp* is able to detect all the introduced changed points (100% recall), but is sometimes oversensitive and makes a false detection (62.6% precision). *block-gs* has less precision and recall than both our algorithms, indicating it is overly sensitive due to its greedy nature. *pos-gs* cannot detect any change points at all. The variation of information values indicate that both *pos-cp* and *pos-cp* obtained blockmodels that are significantly closer to the true blockmodels than the Graphscope variants.

Dataset	Vert. #	Seq. Length	Resolution
Reality Mining	97	290	day
Enron	141	209	week
BGP	30k	13	2 days

Table 3: Real datasets statistics.

Next, we show the results of experiments where we only change the block densities (Table 2b) and the position memberships (Table 2c). The first test should favour *pos-cp*, as *pos-cp* and its position-preserving formulation generally cannot detect change points with block densities changes only. The results in Table 2b confirm this. The second test should favour *pos-cp*, as it only involves position change and our measure BMDD monitors for a change in position and density. Again, the results confirm this. Note that for both tests, *pos-gs* could not detect any change points, and when comparing the block-preserving algorithms, *block-gs* is always less accurate than the equivalent *pos-cp*.

The synthetic results show that the change point algorithms are more accurate than their Graphscope counterparts.

6.2 Real Data Evaluation

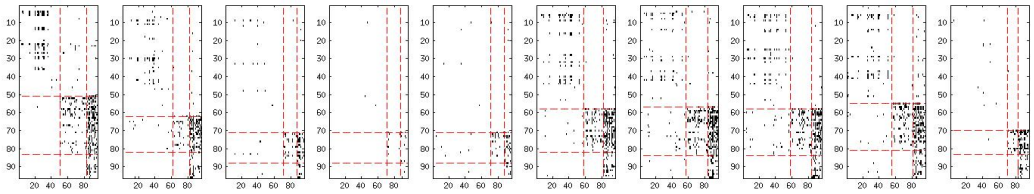
In this subsection, we demonstrate the blockmodels and segmenting produced by the different formulations for the MIT reality mining proximity graphs [10], the Enron email graph [8] and the BGP Internet routing network [12]. Their statistics are described in Table 3.

The Reality Mining graph measures the proximity of users in a laboratory environment over a period of a few months. Each vertex in the graph represents a user, and an undirected edge represents proximity. The Enron email graph tracks the email communications among Enron employees[3]. Each vertex in the Enron graph represents a person, and a directed edge represents sending at least one email from the person represented by the source vertex to the person representing the target vertex. Finally, the BGP Internet routing graph represents the connectivity among organisations in the Internet. A vertex represents an organisation, and an undirected edge represents connectivity between them. The time spanned by each snapshot of the Reality Mining, Enron and BGP graphs are one day, one week and two days respectively. The datasets span a variety of data types, with the MIT and Enron data being unstable and noisy in general and the BGP data being larger and relatively stable.

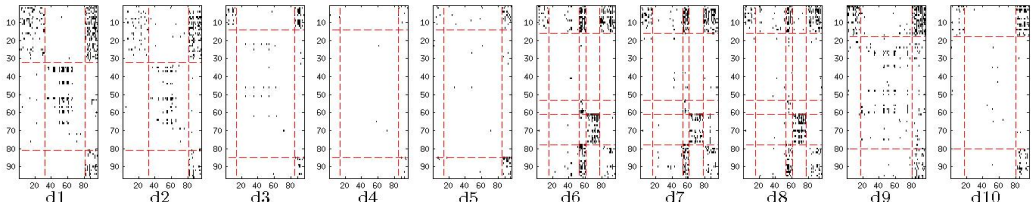
6.2.1 Reality Mining

To provide an illustration of the blockmodels found, we first show a 10 day graph sequence of the proximity network, with each snapshot spanning a day. We show the results from the *pos-cp*, *pos-cp* and *block-gs* algorithms (Figures 3a to 3c). Each figure shows the adjacency matrix of each snapshot, with the red dotted lines as delimiters of the positions/blocks. Time runs top to bottom, left to right. We used a basic matching algorithm to align the blocks and vertices as best we can, but vertices of different snapshots that

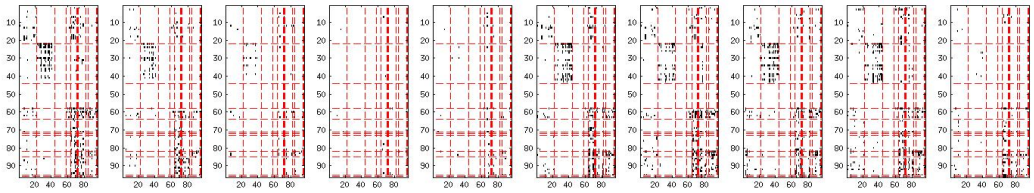
[3]We use the version of dataset that only tracks the internal communications within Enron.



(a) Blockmodels obtained using *pos-cp*.



(b) Blockmodels obtained using *pos-cp*.



(c) Blockmodels obtained using *block-gs*.

Figure 3: Adjacency matrices of a 10 day graph sequence extracted from the Enron graph. Red dotted lines delimit blockmodel decompositions of each snapshot.

have the same labels in the figures might not correspond to the same actual vertex.

As the Figure 3c shows, *block-gs* fragments very easily. Once the number of positions increases, the greedy algorithm is unable to recover and all subsequent parts of the graph sequence are lumped into one blockmodel.

Now consider the blockmodels obtained from *pos-cp* and *pos-cp* (Figures 3a and 3b respectively). They clearly reflect the weekday (5 days of high levels of proximity) and weekends (almost no proxmity detected). Because the position preserving equivalence is easier to satisfy, *pos-cp* is generally able to find blockmodels of longer lengths (e.g. d6–d9 in Figure 3b vs. individual blockmodels over the same period in Figure 3a). In contrast, the block preserving equivalence is less likely to fragment (compare the subsequence d1–d8). Both equivalences are able to produce blockmodels that appear to be visually reasonable.

To analyse the amount of fragmentation and the length of segments obtained, we analysed the number of segments, their average length and the number of positions found over time. The results are in Table 4a and Figure 4a, which shows the number of positions found across time.

The results in Table 4a and Figure 4a confirm that *block-gs* cannot segment the proximity graph sequence and only managed two fragmented segments. *pos-cp*, being more strict than *pos-cp*, had more segments, but the larger standard deviation of segment lengths show that they vary more in length than ones obtained from *pos-cp*. The reason for this is that *pos-cp* tends to be more sensitive to changes (recall the synthetic results), hence more likely to segment during

Algor.	Seg. #	Seg. len.	Run time	Obj. Val.
pos-cp	125	2.33 (4.51)	5.316s	212572
pos-cp	97	3.0 (3.09)	8.873s	188077
block-gs	2	146 (89)	52.384s	348253

(a) Reality Mining Results.

Algor.	Seg. #	Seg. len.	Run time	Obj. Val.
pos-cp	90	2.32 (4.96)	5.036s	155959
pos-cp	59	3.54 (4.75)	7.732s	137817
block-gs	1	209 (0.0)	704.178s	286972

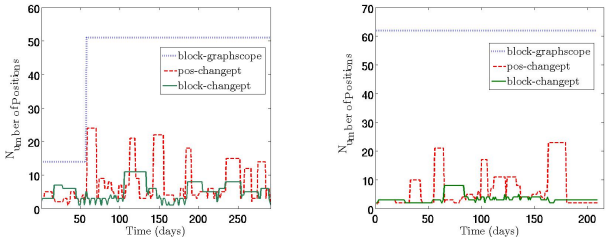
(b) Enron Results.

Table 4: Segmenting results for Reality Mining and Enron. Segment length results are reported as average (standard deviation).

fluctuating periods than *pos-cp*. In addition, both change point models used much less positions to describe the data than *block-gs*, even after we capped the maximum number of positions to 51. The lower objective value of *pos-cp* over *block-gs* suggest that the evolving blockmodels found by *pos-cp* are a better fit. The running time of *block-gs* is much slower than the other two algorithms, because the optimising algorithm's complexity scales at least quadratically with the number of positions.

6.2.2 Enron

We repeat the previous analysis to evaluate the segmenting quality on the Enron data. Consider Table 4b and Figure 4b. They show that *block-gs* is unable to segment the stream at all, producing one segment for the whole stream.



(a) Reality Mining. (b) Enron.

Figure 4: Plots of the number of partitions vs. time for the Reality Mining and Enron datasets.

This caused the single blockmodel to fragment into many positions and causing the run time to grow to 700 seconds. In contrast, both *pos-cp* and *pos-cp* were able to segment the sequence into a number of segments, with *pos-cp* producing more segments again. Again, it shows *block-gs* fragmenting and not being able to produce meaningful blockmodel decompositions, *pos-cp* having more positions in general while *pos-cp* being the most stable of the three, in terms of number of positions.

6.2.3 BGP

The BGP data is largely stable, and for the period we analysed over November 2011, there was no known outages. For both *pos-cp* and *pos-cp* we correctly found one segment, with the set of positions found illustrated in Figure 1. The running time was several hours, which is about 10-100 times more scalable than state of the art blockmodelling algorithms [17].

In summary, both *pos-cp* and *pos-cp* can mine blockmodels that are a balance between accuracy (one blockmodel per snapshot) and simplicity (one blockmodel for the whole sequence). We found weekday/weekend structure in the Reality Mining data, a stable hierarchical structure for the BGP graph, and showed that the change point formulations in SeqiBloc performed well in synthetic datasets and do not fragment like Graphscope does.

7. CONCLUSION

In conclusion, we have presented a novel framework, SeqiBloc, to decompose a dynamic graph into a series of multi-snapshot blockmodels that summarises the evolving structural patterns. In this framework, we have introduced two new definitions of dynamic structural equivalence and showed what this means in terms of adjacency matrix structure and blockmodelling. Based on these definitions, we have formulated four different information theoretic encodings that correspond to the new equivalence definitions and provide an intuitive tradeoff between the number of positions in a blockmodel, the time it spans and how well it fits the subsequence spanned. We then introduced a change point detection approach with a new blockmodel comparison measure, BMDD, to find the appropriate length of these blockmodels. Using synthetic and real datasets like the Reality Mining, Enron and BGP dynamic graphs, we showed our approach can find relevant segments and discover interesting and intuitive blockmodels across time.

For future work, it will be interesting to mine for frequent multi-snapshot blockmodels. For example, in the Reality Mining results, we saw there were clear weekday and weekend structural patterns. If we can group the similar weekend and weekday behaviours (in terms of blockmodels), then we can find normal patterns of interactions and detect outliers.

Another possible future direction is to extend the framework to weighted blockmodels. There are no standard definitions of weighted equivalences and blockmodels, hence there is a need to find intuitive definitions of weighted equivalences.

8. REFERENCES

[1] C. C. Aggarwal, Y. Zhao, and P. S. Yu. On Clustering Graph Streams. In *Proceedings of SDM*, 2010.

[2] E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing. Mixed membership stochastic blockmodels. *J. of Machine Learning Research*, 9, June 2008.

[3] B. Brodsky and B. Darkhovsky. *Nonparametric Methods in Change Point Problems*. Springer, 1993.

[4] D. Chakrabarti, R. Kumar, and A. Tomkins. Evolutionary clustering. In *Proceedings of KDD*, 2006.

[5] Y. Chi, X. Song, D. Zhou, K. Hino, and B. L. Tseng. Evolutionary spectral clustering by incorporating temporal smoothness. In *Proceedings of KDD*, 2007.

[6] M. Coen, H. Ansari, and N. Fillmore. Comparing clusterings in space. In *Proceedings of ICML*, 2010.

[7] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley-Interscience, 2006.

[8] J. Diesner, T. Frantz, and K. Carley. Communication Networks from the Enron Email Corpus It's Always About the People. Enron is no Different. *Comp. & Math. Org. Theory*, 11(3), 2005.

[9] T. N. Dinh, I. Shin, N. K. Thai, M. T. Thai, and T. Znati. *A General Approach for Modules Identification in Evolving Networks*. Springer, 2010.

[10] N. Eagle and A. (Sandy) Pentland. Reality mining: sensing complex social systems. *Personal Ubiquitous Comput.*, 10, March 2006.

[11] D. Greene, D. Doyle, and P. Cunningham. Tracking the Evolution of Communities in Dynamic Social Networks. In *Proceedings of ASONAM*, 2010.

[12] Y. Hyun, B. Huffaker, D. Andersen, E. Aben, M. Luckie, kc claffy, and C. Shannon. The IPv4 Routed /24 AS Links Dataset - November 2011.

[13] M. Meila. Comparing clusterings by the variation of information. In *Proceedings of the 16th Annual Conference on Learning Theory and 7th Kernel Workshop*, page 173. Springer Verlag, 2003.

[14] J. Rissanen. Modeling by shortest data description. *Automatica*, 14, 1978.

[15] J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *Proceedings of KDD*, 2007.

[16] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge Uni. Pr., 1994.

[17] E. P. Xing, W. Fu, and L. Song. A State-Space Mixed Membership Blockmodel for Dynamic Network Tomography. *Annals of Applied Statistics*, 4(2), 2010.

[18] T. Yang, Y. Chi, S. Zhu, Y. Gong, and R. Jin. Detecting communities and their evolutions in dynamic social networks - a Bayesian approach. *Machine Learning*, 82, 2011.