Discovering Latent Blockmodels in Sparse and Noisy Graphs using Non-Negative Matrix Factorisation [*]

Jeffrey Chan[†], Wei Liu[‡], Andrey Kan[§], Christopher Leckie, James Bailey,
Kotagiri Ramamohanarao
Department of Computing and Information Systems, University of Melbourne, Australia

ABSTRACT

Blockmodelling is an important technique in social network analysis for discovering the latent structure in graphs. A blockmodel partitions the set of vertices in a graph into groups, where there are either many edges or few edges between any two groups. For example, in the reply graph of a question and answer forum, blockmodelling can identify the group of experts by their many replies to questioners, and the group of questioners by their lack of replies among themselves but many replies from experts.

Non-negative matrix factorisation has been successfully applied to many problems, including blockmodelling. However, these existing approaches can fail to discover the true latent structure when the graphs have strong background noise or are sparse, which is typical of most real graphs. In this paper, we propose a new non-negative matrix factorisation approach that can discover blockmodels in sparse and noisy graphs. We use synthetic and real datasets to show that our approaches have much higher accuracy and comparable running times.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data Mining; E.1 [**Data Structures**]: Graphs and Networks; J.4 [**Computer Applications**]: Social and Behavioral Sciences

Keywords

Blockmodel, Graphs, Non-negative Matrix Factorisation

1. INTRODUCTION

Discovering latent community structures is an important aspect of understanding, predicting and modelling many types of graphs[1] [10][11]. A common and intuitive definition of a community is a group of vertices that are densely connected among themselves and sparsely connected to vertices of other communities. However, there are many other graphs that have a latent structure that is different from this definition. For example, in the reply graph of a question and answer (Q&A) forum, a group of experts can be identified by their many replies to questioners, and the group of questioners by their lack of replies among themselves but many replies from experts [3]. If we use a community definition to decompose such a Q&A forum, everyone will be incorrectly associated in the same community. This demonstrates that a more general approach to finding the inherent graph structure is needed.

Blockmodelling is a powerful approach to decomposing graphs, and has been well studied in the social sciences [13]. Vertices are in the same *position* (group) if they have similar patterns of interactions to vertices of other positions. The Q&A forum structure fits this definition. The inherent structure is revealed by the *image diagram* (e.g., Figure 1f), where each vertex represents a position and an edge represents the aggregate interaction between positions, with insignificant interactions not displayed. The positions and the image diagram (and matrix representation of it) clearly summarise the overall social structure of the Q&A behaviour, and together form a *blockmodel*. Blockmodels allow us to: (1) explore the membership of vertices and how they relate to each other, (2) understand and characterise the underlying structure (e.g., is it a community or core-periphery structure) and (3) discover the important roles.

Blockmodelling is an NP-Hard problem [5], hence blockmodelling algorithms only seek to find good local optima. Wang et al. [12] and Long et al. [9] have formulated the blockmodelling problem[2] as non-negative matrix tri-factorisation. The problem becomes one of finding the optimal positions and image matrices that simultaneously minimise the difference between the original graph and the graph approximated by the blockmodel. Similarly, Reichardt et al. [11] have shown the similarity between the blockmodelling problem and finding optimal spin configurations in material physics. While promising results were reported, there are three important, unresolved challenges.

The first challenge is that many real graphs are sparse. Consider the well studied karate club network [13], which represents the associations between members after a schism

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

[†]Corresponding author: jeffrey.chan@unimelb.edu.au

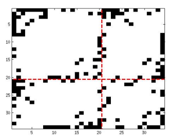
[‡]Currently at NICTA ATP Research Laboratory, Australia

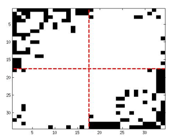
[§]Currently at Walter and Eliza Hall Institute, Australia

CIKM'13, Oct. 27–Nov. 1, 2013, San Francisco, CA, USA.
Copyright 2013 ACM 978-1-4503-2263-8/13/10 ...$15.00.
http://dx.doi.org/10.1145/2505515.2505595.

[1]We use the terms graphs and networks interchangeably.

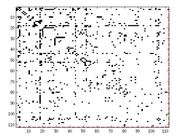
[2]The authors of these works use the term "community", but these works are effectively finding blockmodels.

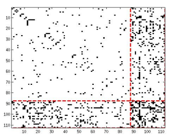

(a) Structure using [12].


(b) Structure accounting for sparsity.

$$\begin{pmatrix} 0.104 & 0.138 \\ 0.138 & 0.159 \end{pmatrix}$$

(e) Image matrix found of 1a.


(c) Structure using [9].


(d) Structure accounting for noise.

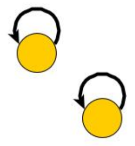

(f) Image diagram of 1b.

Figure 1: Rearranged adjacency matrices of the Karate club association network (Figures 1a-1b) and word adjacency graph (Figures 1c-1d).

in the club. The known structure is a two community structure (illustrated in Figure 1b, where an edge is represented by a black square and an absent edge by white space.). The blockmodel definition can represent a community structure. Hence the two non-negative factorisation algorithms in [12][9] should be able to find these communities, but over 100 runs we regularly obtain the incorrect blockmodel structure illustrated in Figure 1a. The reason this occurs is because the factorisation algorithms penalise edge and non-edge mismatches equally, where a non-edge is the absence of an edge between a vertex pair. In a sparse graph, the grouping of edges is more important than the grouping of non-edges, since performing the former is more difficult and therefore more likely that the group represents inherent structure and not structure by chance. Figure 1e shows the image matrix of the blockmodel discovered by [12]. This indicates that in the graph approximated by the blockmodel of [12], most of the vertex pairs will be predicted to have no edge between them, which is a good solution since many of the non-edges are matched, but the blockmodel of Figure 1b is in fact the more intuitive and valid one. Hence, an approach is needed that penalises the two types of mismatches differently.

The second challenge is that in some graphs the underlying structure is obfuscated by background noise and unexpected edges. For example, consider Figures 1c and 1d, which shows a word adjacency network of common adjectives and nouns in the Charles Dickens novel "David Copperfield" [10]. If we divide the nouns into one position and adjectives into another, the rearranged adjacency matrix looks almost uniformly random, suggesting for this dataset the adjacencies of the words follow structural patterns that are different from their classifications. A possible structure is illustrated in Figure 1d, which shows the adjectives and nouns that are adjacent to many of the other words that are grouped together. For this graph, the two factorisation algorithms preferred one position (although we initialise the algorithms to find two, see Figure 1c). As this example shows, it is important for blockmodelling algorithms to be able to handle unexpected structure and be insensitive to noise.

The third challenge is the scalability of the algorithms. Reichardt et al. [11] showed in their work that their approach can find blockmodels for sparse graphs. However, their simulated annealing approach is slow (it did not finish one run for a graph with 1000 vertices after five days). Therefore, it is important for any blockmodel algorithm to discover blockmodels in a reasonable amount of time.

In this paper, we introduce several objectives and optimisation algorithms to tackle these challenges (we call our framework *FactorBlock*). Our approach is able to discover the true blockmodel structure in a) sparse graphs; b) noisy graphs; and c) produces results of higher accuracy and comparable running times to the state of the art. Our approach is based on placing more importance in approximating edges well when the graph is sparse. We evaluate existing and proposed algorithms using both real and sparse graphs, and show that our proposed approaches are more accurate in both synthetic and real datasets as well as having comparable running times to the fastest existing algorithms.

2. RELATED WORK

Non-Negative Matrix Factorisation: Seung et al. [8] were the first to popularise non-negative matrix factorisation in machine learning. Subsequent work introduced coordinate descent and projected gradient descent [2] to improve the optimisation obtained.

Long et al. [9] and Wang et al. [12] introduced the idea of non-negative matrix tri-factorisation for finding blockmodels for graphs, and produced different multiplicative optimisation approaches. Zhang et al. [14] introduced a coordinate descent algorithm to find overlapping position blockmodels. These algorithms perform well when the dataset structure is clear, but are not able to handle sparse and noisy graphs well, as we show in Section 6.

Blockmodelling: In [3], Chan et al. proposed an information theoretic approach of finding blockmodels in evolving graphs. Airold et al. [1] introduced a mixed membership probabilistic model, where vertices can belong to multiple positions. Karrer et al. [7] proposed a generative probabilistic model that takes the difference in the degree distributions of vertices into account. However, their formulation is specifically targeting heavy tailed graphs, while our model is general and can easily incorporate different types of graphs.

Reichardt et al. [11] proposed a null model formulation that sums the difference between the adjacency matrix and the blockmodel approximation of it. A simulated annealing approach was proposed to optimise this. The objective of Reichardt is most similar to ours, but our formulation allows the computation of the gradient and hence we can use faster gradient descent approaches for optimisation.

3. BLOCKMODELLING BACKGROUND

In this section, we summarise the key ideas of blockmodelling (please see [13] for more details). A graph $G(V, E)$ consists of a set of vertices V and a set of edges E, $E : V \times V$, which can be represened by an adjacency matrix $\mathbf{A}$. We initially consider unweighted, directed graphs in this paper, as the definition of blockmodels are traditionally defined in terms of this type of graphs [13].

A blockmodel is a form of dimension reduction and decomposes a graph ($\mathbf{A}$) into a set of vertex partitions (called *positions*), represented by a membership matrix $\mathbf{C} \in [0, 1]^{n \times k}$, and an image matrix $\mathbf{M} \in [0, 1]^{k \times k}$, which describes the likelihood of an edge between a vertex from one position to a vertex of another position (k is the number of positions). Each entry in $\mathbf{M}$ is called a block. The blockmodel decomposition approximates $\mathbf{A}$ as $\mathbf{CMC}^T$.

As the decomposition is an approximation, we estimate the error as the sum of squared differences and the aim is to find a blockmodel ($\mathbf{C} \geq \mathbf{0}$ and $\mathbf{M} \geq \mathbf{0}$) that minimises:

$$\min_{\mathbf{C},\mathbf{M}} ||\mathbf{A} - \mathbf{CMC}^T||_F^2 \qquad (1)$$

Equation 1 has the same form as non-negative matrix tri-factorisation [4]. In blockmodelling, there are different definitions of vertex equivalence and in this paper, we solve the popular structural equivalence [13], which states that two vertices belong to the same position if they have similar sets of out and in neighbours. In terms of matrix structure, this means the densities of the entries of the image matrix are ideally close to 0 or 1.

4. MATRIX FACTORISATION BLOCKMODELLING (FactorBlock)

We introduce two new constraints and weightings to address the sparsity and noise challenges.

Handling Graph Sparseness.

Reconsider Equation 1, which we rewrite as:

$$\sum_{i,j} \mu_{10}\mathbf{A}_{i,j}(1-(\mathbf{CMC}^T)_{i,j}) + \mu_{01}(1-\mathbf{A}_{i,j})(\mathbf{CMC}^T)_{i,j} \quad (2)$$

where μ_{10} and μ_{01} are the penalty weightings for an edge mismatch ($\mathbf{A}_{i,j} = 1$ and $(\mathbf{CMC}^T)_{i,j} < 1$) and a non-edge mismatch ($\mathbf{A}_{i,j} = 0$ and $(\mathbf{CMC}^T)_{i,j} > 0$) respectively. In Equation 1, $\mu_{10} = \mu_{01} = 1$ (equal weighting).

μ_{10} and μ_{01} determine the importance we place on avoiding each of the two types of errors. Real graphs tend to be sparse, hence it is much easier to match an actual non-edge with the blockmodel approximation. Reconsidering the karate club example, according to Equation 1, blockmodel A (Figure 1a) and blockmodel B (Figure 1b) have similar objective values, as the graph is sparse and therefore the non-edges are (mostly) matched by both blockmodels and hence both objective values are similar. But we know that the true blockmodel is more similar to blockmodel B, and to make that blockmodel more desirable in terms of the objective, we need the matching of an existing edge ($\mathbf{A}_{i,j} = 1$) to be given more weight than the matching of an existing non-edge ($\mathbf{A}_{i,j} = 0$) or $\mu_{10} > \mu_{01}$.

One possible scheme is to use the density of the graph as a background model: $\mu_{10} = \mathbf{A}_{i,j} - \frac{m}{n^2}$, $\mu_{01} = \frac{m}{n^2}$ where m is the number of edges. With this weightings, in the example μ_{10} (μ_{01}) is almost 1 (0), hence the emphasis is to match the edges. The optimal configuration for this is to place most of the edges into the two diagonal blocks, which is exactly what we seek. We can rewrite Equation 1 with the changed edge mismatching weighting as:

$$\min_{\mathbf{C},\mathbf{M}} \left(||(\mathbf{A} - \mathbf{CMC}^T) \circ (\mathbf{A} - \mathbf{R})||_F^2 \right) \qquad (3)$$

where $\mathbf{R} \in [0,1]^{n \times n}$, $\mathbf{R}_{i,j} = \frac{m}{n^2}$, $\forall i,j \in V$ and $\circ$ represents the element-wise multiplication operator. This weighting scheme can also be considered from the perspective of a null model, similiar to the modularity objective for community finding [10]. For example, our weighting scheme is the same as assuming a Erdos-Renyi null model [10].

Handling Background Noise.

In noisy graphs, it can be difficult to distinguish between the true structure of the graph and the background noise. We can differiate better by encouraging blocks to be either dense or sparse (according to the null model). If the density of the block is below what the null model predicts it should be, then we encourage the block to be sparse and hence does not attract edge assignments to it. Conversely, if the density is above what the null model predicts, then we encourage the block to be dense and attract more edges to it.

Recall that in the ideal case, the densities of the image matrix entries should either be 0 or 1 and we denote this as the *ideal image matrix*, $\mathbf{M_{ideal}} \in \{0,1\}^{k \times k}$. Then we seek an $\mathbf{M}$ that is as close to $\mathbf{M_{ideal}}$ as possible, i.e., it minimises the distance $||\mathbf{M_{ideal}} - \mathbf{M}||_F^2$. $\mathbf{M_{ideal}}$ is a function of $\mathbf{M}$, and can be defined as $\mathbf{M_{ideal}}(i,j) = argmin_{u=\{0,1\}}(|u - \mathbf{M}(i,j)|)$.

However, this is not differentiable and we cannot use faster gradient descent approaches. Hence we propose a sigmoid function to approximate it: $\mathbf{M_{ideal}} = \frac{1}{1+\gamma e^{-\upsilon(\mathbf{M}-\tau)}}$ with υ related to the growth rate, and γ and τ related to when the function approaches 1. Combining Equation 3 with this constraint term and letting $\mathbf{U} = \mathbf{A} - \mathbf{R}$, we aim to minimise the following objective:

$$L = ||(\mathbf{A} - \mathbf{CMC}^T) \circ \mathbf{U}||_F^2 + \beta \cdot ||\mathbf{M_{ideal}} - \mathbf{M}||_F^2 \qquad (4)$$

subject to $\mathbf{C} \geq \mathbf{0}$ and $\mathbf{0} \leq \mathbf{M} \leq \mathbf{1}$. The first term in Equation 4 tries to minimise the approximation of the blockmodelling tri-factorisation, while the second, constraint term tries to find an $\mathbf{M}$ that is as close as possible to the ideal for the particular equivalence required. $\beta \in [0,1]$ is a parameter that controls the amount of influence from the constraint.

The matrix τ, $\tau \in [0,1]^{k \times k}$, controls the boundary between what is considered as a sparse or dense block. When there is no null model, i.e., $\mathbf{U} = \mathbf{1}$, then the default is for $\tau = 0.5$ for all its entries. When a null model is used, the edge mismatch errors μ_{10} and μ_{01} determine the density boundary. When we use our weighting scheme, then $\tau = \frac{m}{n^2}$. υ controls the slope of the function, and after performing parameter evaluation, we found a setting of 100-1000 works well, as the sigmoid function is almost a step function and hence approximates the exact $\mathbf{M_{ideal}}$ formulation well.

We have three different objectives that we evaluate in Section 6. We call Equation 1 as *Euclidean*, and Equation 3, 4 and, 4 with $\mathbf{U} = \mathbf{1}$ as *adjusted*, *constrained adjusted* and *constrained Euclidean* respectively.

5. OPTIMISATION APPROACHES

Both non-negative matrix factorisation and blockmodelling of three or more positions are NP-Hard problems [5]. Hence, we developed alternating optimisation approaches to find good $\mathbf{M}$ and $\mathbf{C}$ solutions to Equation 4, which can also be used to solve the other objectives.

We introduce three approaches for optimising $\mathbf{C}$, one where we assume hard memberships (i.e., each vertex must belong to one and only one position), and a gradient descent and a coordinate descent algorithm for soft, overlapping memberships. In addition, we introduce two approaches, one based on gradient descent and the other on coordinate descent for optimising $\mathbf{M}$. Unless stated otherwise, the derivations for the algorithms and their gradients are available at `http://people.eng.unimelb.edu.au/jeffreyc`.

5.1 Position Optimisation

5.1.1 Hard Membership

When the position memberships are hard, then we are solving a discrete, NP-Hard problem. The objective function needs to be recomputed and reassessed after each membership change. Recomputing Equation 4 is expensive (complexity $O(n^2k^2)$), hence we introduce an incremental approach that only updates the necessary entries.

Let the current position membership and image matrices be denoted by $\mathbf{C}^{(t)}$ and $\mathbf{M}^{(t)}$. Without loss of generality, let the vertex v_i be reassigned from its current position P_g to its new position P_h, and let the corresponding matrices (of size $n \times k$) to perform these atomic operations be denoted by $\mathbf{\Delta^-_{i,g}}$ and $\mathbf{\Delta^+_{i,h}}$ respectively. $\mathbf{\Delta^-_{i,g}} = -1$ for (i, g), and 0 otherwise, and $\mathbf{\Delta^+_{i,h}} = 1$ for (i, h) and 0 otherwise. Note that $\mathbf{C}^{(t+1)} = \mathbf{C}^{(t)} - \mathbf{\Delta^-_g} + \mathbf{\Delta^+_h}$, hence we can rewrite Equation 4 (we drop the superscript for $\mathbf{M}^{(t)}$) as $\mathbf{D}^{(t+1)} = \mathbf{D}^{(t)} + \mathbf{PI}^{(t)}\Lambda^T + \Lambda\mathbf{IP}^{(t)} - \Lambda\mathbf{M}\Lambda^T$, where $\mathbf{D}^{(t)} = (\mathbf{A} - \mathbf{C}^{(t)}\mathbf{M}(\mathbf{C}^{(t)})^T)$, $\Lambda = \mathbf{\Delta^-_g} - \mathbf{\Delta^+_h}$, $\mathbf{PI}^{(t)} = \mathbf{C}^{(t)}\mathbf{M}$ and $\mathbf{IP}^{(t)} = \mathbf{M}(\mathbf{C}^{(t)})^T$. We can compute $\mathbf{D}^{(t+1)}$ by first assuming $\mathbf{D}^{(t+1)} = \mathbf{D}^{(t)}$ and then:

$$\mathbf{D}^{(t+1)}_{*,i} = \mathbf{PI}^{(t)}_{*,g} - \mathbf{PI}^{(t)}_{*,h} \tag{5}$$

$$\mathbf{D}^{(t+1)}_{i,*} = \mathbf{IP}^{(t)}_{g,*} - \mathbf{IP}^{(t)}_{h,*} \tag{6}$$

$$\mathbf{D}^{(t+1)}_{i,i} = M_{g,g} - M_{h,g} - M_{g,h} + M_{h,h} \tag{7}$$

Then for vertex v_i, the position that results in a minimal sum of the $\mathbf{D}^{(t+1)}$ terms in Equations 5 to 7 is its best position assignment. Computing Equations 5 to 7 has a complexity of $O(k)$ and since we precompute $\mathbf{D}^{(t)}$, $\mathbf{PI}^{(t)}$ and $\mathbf{IP}^{(t)}$, the complexity of position evaluation is also $O(k)$. $\mathbf{PI}^{(t+1)}$ and $\mathbf{IP}^{(t+1)}$ can be similarly updated using rules similar to Equations 5 and 6. We call this approach *HardIncr*.

5.1.2 Soft Memberships

Projected Gradient Descent: A simple approach to solve Equation 4 is to use projected gradient descent [2]. The idea is to compute the gradient of $\mathbf{C}$ and $\mathbf{M}$, then use a line search to find the appropriate step size (ψ) that results in the minimal objective value. To ensure that both $\mathbf{M}$ are $\mathbf{C}$ are non-negative, they are projected to the non-negative quadrant. We call this approach *SoftGrad*.

Coordinate Descent: Another well known optimisation approach is coordinate descent. We optimise $\mathbf{C}$ by optimising a sequence of sub-problems based on a set of conjugate bases. It has been shown that optimising for each of the conjugate bases sequentially will converge to a stationary point. We follow [6][14] and use the unit bases of $\mathbf{C}$, denoted $\mathbf{E} \in [0, 1]^{n \times k}$, where $\mathbf{E}_{i,j} = 1$ for (i, j), otherwise 0. Let ψ denote the step size for each conjugate basis and $\mathbf{C}^{(\prime)} = \mathbf{C} + \psi\mathbf{E}_{i,j}$. Each sub-problem can be written as $\min_\psi L_{i,j}(\psi)$, where

$$L_{i,j}(\psi) = ||(\mathbf{A} - \mathbf{C}^{(\prime)}\mathbf{M}(\mathbf{C}^{(\prime)})^T) \circ \mathbf{U}||^2_F + \beta \cdot ||\mathbf{M_{ideal}} - \mathbf{M}^{(t)}||^2_F \tag{8}$$

Following a similar approach taken by [14], we solve Equation 8 for ψ, which results in a polynomial of order 4. Since we use a similar approach to finding the roots as [14], please refer to [14] for details. We call this approach *SoftCoord*.

Name	Objective	Pos. Method	Im. Method
grad-H	Euclidean	HardIncr	ImageGrad
coor-H	Euclidean	HardIncr	ImageCoord
RGC-H	Euclidean	HardIncr	Mult. [9]
reic-H	Reichardt [11]	SA [11]	SA [11]
grad-H-Cn	Con. Euc.	HardIncr	ImageGrad
coor-H-Cn	Con. Euc.	HardIncr	ImageCoord
grad-H-Ad	Ad. Euc.	HardIncr	ImageGrad
coor-H-Ad	Ad. Euc.	HardIncr	ImageCoord
grad-H-CnAd	Con. Ad. Euc.	HardIncr	ImageGrad
coor-H-CnAd	Con. Ad. Euc.	HardIncr	ImageCoord
RGC-S	Euclidean	Mult. [9]	Mult. [9]
ANMF-S	Euclidean	Mult. [12]	Mult. [12]
BNMTF-S	Euclidean	Coord. D. [14]	Coord. D. [14]
grad-S-Ad	Ad. Euc.	SoftGrad	ImageGrad
coor-S-Ad	Ad. Euc.	SoftCoord	ImageCoord
grad-S-CnAd	Con. Ad. Euc.	SoftGrad	ImageGrad
coor-S-CnAd	Con. Ad. Euc.	SoftCoord	ImageCoord

Table 1: Summary of the algorithms and objectives. Names in italic represent our proposed algorithms.

Name	Vert. #	Edge #	Pos. #
Baboon	14	23	2
Monastery	18	34	4
Karate	34	78	2
Politic Books	105	441	2
College Football	115	613	12
Politic Blogs	1490	19090	2

Table 2: Statistics of the real graphs.

5.2 Image Matrix Optimisation

Optimising the image matrix involves optimising the sigmoid $\mathbf{M_{ideal}}$ term. There is no closed form solution to a sigmoid function, hence we do not use the traditional multiplicative method and instead use the gradient and coordinate descent approaches combined with a line search to optimise for $\mathbf{M}$.

Projected Gradient Descent: We apply the same projected gradient descent approach as for optimising $\mathbf{C}$. Please see the supplement for the gradient with respect to $\mathbf{M}$. We call this method *ImageGrad*.

Coordinate Descent: Similar to optimsing for $\mathbf{C}$, we now solve the following problem:

$$\min_\psi L_{i,j}(\psi) = ||(\mathbf{A} - \mathbf{C}\mathbf{M}'\mathbf{C}^T) \circ \mathbf{U}||^2_F + \beta||\mathbf{M_{ideal}}' - \mathbf{M}'||^2_F \tag{9}$$

where $\mathbf{M}' = \mathbf{M} + \psi\mathbf{E}_{i,j}$ ($\mathbf{E} \in [0, 1]^{k \times k}$) and $\mathbf{M_{ideal}}' = 1 + [\gamma e^{-\upsilon(\mathbf{M}' - \tau)}]^{-1}$. We use a line search to solve for ψ. We call this method *ImageCoord*.

6. EVALUATION

In this section, we compare the accuracy and running times of the different objectives and algorithms. We use both real and generated datasets to evaluate our algorithms.

6.1 Datasets and Evaluation Criteria

In Section 5, we introduced a number of objectives and position and image matrix optimisation approaches. As there are many possible combinations, we selected 12 of them for reporting (see Table 1) and compared them against five existing algorithms. We divided the algorithms into two groups. The first group uses our incremental hard position approach (hardIncr) and evaluates the Euclidean objective and its variants. We also compare our proposed approaches against Reichardt's simulated annealing approach (reic-H) and Long et al. [9] (RGC-H[3]) The second group are based on soft memberships. The existing algorithms of [9], [12] and [14] are proposed for soft membership (RGC-S, ANMF-S and BNMTF-S, respectively).

6.1.1 Datasets

We generate our synthetic datasets with the aim of evaluating how sparsity and noise affect the running time and accuracy of the different objectives and algorithms.

Our approach is the reverse of the blockmodelling problem. We first generate the memberships ($\mathbf{C}$) and the image matrix ($\mathbf{M}$). Then we generate the graph ($\mathbf{A}$) using $\mathbf{A} = \mathbf{CMC}^T$. We generate $\mathbf{C}$ by first generating the position sizes from a uniform distribution (note that other distributions can be used). Then we determine the position membership of vertices by drawing from a hyper-geometrical distribution, where the probability of each position is its relative size. We generated $\mathbf{M}$ such that they replicate three common graph structures: *community*, *core periphery* and *hierarchy*. To vary the sparsity, we keep $\mathbf{C}$ the same, but change the densities of the dense blocks in $\mathbf{M}$. To vary the noise, we use an approach similar to [7], which adds uniformally random background noise to $\mathbf{M}$, with parameter λ controlling the amount of noise.

We also evaluate the algorithms using six real networks[4] that have known vertex labels. These are graphs that are commonly used to evaluate blockmodelling algorithms. Their statistics are displayed in Figure 2.

6.1.2 Evaluation Criteria

Normalised mutual information (NMI) has been used to evaluate blockmodelling algorithms [9][14], hence we also use NMI as the basis of our evaluation. We use the clock running times to evaluate the scalability of the algorithms. All algorithms are implemented in Matlab 2012b. Experiments were performed on an Intel Core 2 2.53GHz laptop with 4GB of memory. For existing methods that have parameters, like reic-H, we used the default parameter settings of the original implementation. Due to space constraints, our evaluation of the parmeter sensitivity of the sigmoid function and the effect of graph size are available in the supplement.

6.2 Results

In this section, we present the results of our evaluation. For the synthetic datasets, we generated 5 graphs of 100 vertices per parameter setting, then ran each algorithm 10 times. For each run, we generated 10 random initialisations, obtained a blockmodel from each initialisation, and choose the blockmodel with the lowest objective value as the one to compare against. For each algorithm, we averaged the NMI and running times across the runs.

6.2.1 Sparsity Sensitivity

To evaluate the effect of sparsity, we varied the density of the dense blocks from 1.0 to 0.1. We obtained similar results for the three graph types, hence we only report the results for the community structure.

Figure 2 shows the results of our evaluation. Note that each column of figures use the same legend. We divided the evaluation into three groups. The first group, Figures 2a, 2b, 2f and 2g, compare the different objectives with our proposed algorithms. It shows that coor-H-Ad and coor-H-AdCn has the highest average NMI across most of the density range. In particular, the results for algorithms that use the adjusted Euclidean objective or variants are generally higher than those that only use the Euclidean objective as we decrease the density of the graphs.

All the algorithms find it difficult to find the true blockmodel when the density is 0.1. At that density, there are only a few edges in each diagonal block, which makes it almost like noise and hence difficult to optimise. The running times (Figure 2f) of the projected gradient descent algorithms are generally faster than coordinate descent ones (Figure 2g), so the results suggest to use coordinate descent if accuracy is more important, but projected gradient descent if speed is more important.

Next, we compare the two best algorithms (coor-H-Ad and grad-H-Ad) against the existing algorithms, RGC-H [9] and reic-H [11] for hard position blockmodelling, illustrated in Figures 2c and 2h. The results show that our algorithms are clearly much more accurate than the other two, and of comparable speed to RGC-H. Finally, we evaluate how the soft approaches perform for these datasets. We follow [9], [12] and [14] and use $k = \max_k \mathbf{C}_{i,k}$ to determine the position of vertex v_i (k). Figures 2d and 2i show the results. As can be seen, BNMTF-S [14] has the highest accuracy, but also the slowest running time. In addition, all the NMI results are clearly much lower than the results of the algorithms that use hardIncr, suggesting that these soft position algorithms might be inadequate for finding hard positions.

6.2.2 Noise Sensitivity

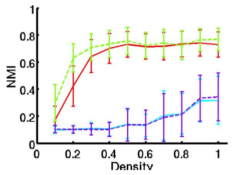
To evaluate how each of the algorithms perform as the amount of background noise increased, we vary λ from 0 (no noise) to 0.9 (almost uniformly random edge distribution). We divide the analysis into three groups again, but due to space limitations, we only report the best results for hard positions (Figures 2e and 2j). The results for soft positions algorithms have the same trends as the sparsity results. Of the algorithms proposed, coor-H-Ad is the most accurate for most of the background noise range, although coor-H-AdCn is actually the most accurate at a noise proportion of 0.9 (see supplement). The figures again show the two best algorithms coor-H-Ad and grad-H-Ad are much more accurate than the existing algorithms over a large range of noise levels, and have comparable running times.

6.2.3 Real Datasets

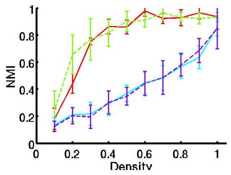
Our results for the real datasets are given in Table 3. Note that we stopped the evaluation of reic-H and BNMTF-S on the Political blogs dataset after 72 hours, which we believe is an unreasonable length of time to evaluate a graph with around 1500 vertices. Apart from the Baboon dataset, the results show that the adjusted Euclidean algorithms have the highest NMI while having comparable running speeds to the other hard position results. Our FactorBlock approaches

[3]We coupled our incremental position approach with their image matrix update, which we found to be faster and more accurate then their default approach.

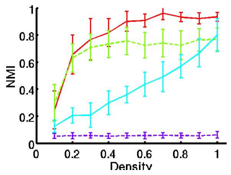
[4]Available at `http://www-personal.umich.edu/~mejn/netdata/`



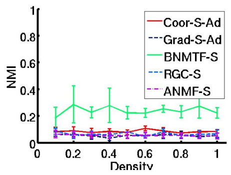
(a) Grad des alg., NMI.



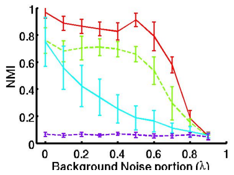
(b) Coor des alg., NMI.



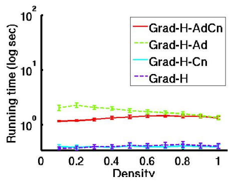
(c) Hard pos alg., NMI.



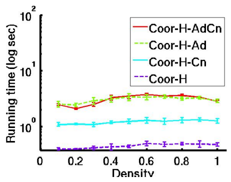
(d) Soft pos alg., NMI.



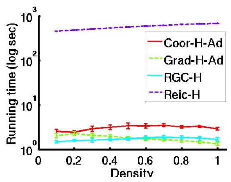
(e) Hard pos alg., NMI.



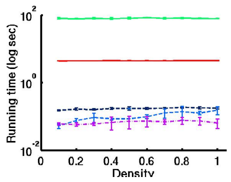
(f) Grad des alg., time.



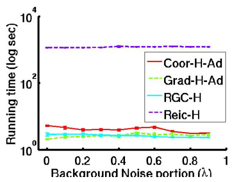
(g) Coor des alg., time.



(h) Hard pos alg., time.



(i) Soft pos alg., time.



(j) Hard pos alg., time.

Figure 2: Sparsity (first 4) and noise (last column) sensitivity results. The figures in each column share the same legend.

Data / **Alg.**	**Baboon**		**Monastery**		**Karate**		**Pol. Books**		**Football**		**Pol. Blogs**	
	NMI	Time	NMI	Time	NMI	Time	NMI	Time	NMI	Time	NMI	Time
RGC-H	**0.5295**	0.062	0.1551	0.169	0.0064	0.176	0.0131	0.540	0.2618	3.340	0.0002	922.4
Reic-H	0.0588	17.73	0.1898	49.02	0.038	34.67	0.0238	135.7	0.2352	2342.8	*	*
Grad-H-Ad	0.135	0.135	0.2551	0.190	0.152	0.139	**0.4451**	0.483	0.5849	4.003	**0.1025**	480.4
Coor-H-Ad	0.3163	0.181	**0.3083**	0.588	0.0064	0.300	0.3049	0.817	0.5133	6.564	0.0001	609.7
Grad-H-AdCn	0.1854	0.185	0.2207	0.132	**0.3592**	0.140	0.1704	0.429	**0.6031**	2.115	0.1012	610.3
Coor-H-AdCn	0.3163	0.273	0.2478	0.319	0.0064	0.389	0.0137	0.816	0.4375	13.48	0.001	487.9
RGC-S	0.0026	0.019	0.1817	0.014	0.0023	0.023	0.0014	0.038	0.2481	0.072	0.001	3.888
ANMF-S	0.0669	0.046	0.2538	0.020	0.0123	0.02	0.0035	0.039	0.2453	0.056	0.0013	5.335
BNMTF-S	0.0175	2.700	0.2181	7.044	0.0129	6.917	0.0171	24.95	0.3141	173.9	*	*

Table 3: Real dataset results (time in seconds). For each dataset, the algorithm with the best NMI is highlighted in bold. The algorithms proposed in this paper are in italics. * indicates that the algorithm did not finish after 72 hours.

have an improvement of 21.47% (Monastery dataset) or 0.3 or higher in absolute NMI for the Karate, Political blogs, Football and Political books datasets. In the Karate and Football datasets, the sigmoid constraints achieved the highest NMI, suggesting that the constraint term makes it easier for the algorithm to avoid sub-optimal minima. In addition, these tests show that the soft position algorithms have difficulties finding good blockmodels, and hence are a potential area of future research.

7. CONCLUSION

In conclusion, we have described the important problem of blockmodelling and shown why the current state of the art factorisation methods cannot discover blockmodels accurately in sparse and noisy graphs. We proposed our framework FactorBlock, which enables new approaches based on optimising a weighted and constrained objective with gradient and coordinate descent methods. In our evaluation, we showed that the weighting for sparseness and the sigmoid constraint term enabled our approaches to have much higher accuracy for sparse and noisy graphs, while having comparable running time to the state of the art methods.

8. REFERENCES

[1] Edoardo M. Airoldi, David M. Blei, Stephen E. Fienberg, and Eric P. Xing. Mixed membership stochastic blockmodels. *JLMR*, 9:1981–2014, 2008.

[2] M. Berry, M. Browne, A. Langville, V. Pauca, and R. Plemmons. Algorithms and Applications for Approximate Nonnegative Matrix Factorization. *CSDA*, 52(1):155–173, 2007.

[3] J. Chan, W. Liu, C. Leckie, J. Bailey, and R. Kotagiri. SeqiBloc: Mining Multi-time Spanning Blockmodels in Dynamic Graphs. In *KDD*, pages 651–659, 2012.

[4] C. Ding, T. Li, W. Peng, and H. Park. Orthogonal Nonnegative Matrix Tri-Factorizations for Clustering. In *KDD*, pages 126–135, 2006.

[5] J. Fiala and D. Paulusma. The computational complexity of the role assignment problem. Technical report, 2003.

[6] C. Hsieh and I. Dhillon. Fast coordinate descent methods with variable selection for non-negative matrix factorization. In *KDD*, pages 1064–1073, 2011.

[7] B. Karrer and M. Newman. Stochastic blockmodels and community structure in networks. *Phys. Rev. E*, 83(016107):1–11, 2011.

[8] D. Lee and H. Seung. Algorithms for Non-negative Matrix Factorization. *In NIPS*, 13:556–562, 2001.

[9] B. Long, Z. Zhang, and P. Yu. A general framework for relation graph clustering. *KAIS*, 24(3):393–413, 2009.

[10] M. Newman. Modularity and community structure in networks. *PNAS*, 103(23):8577–82, 2006.

[11] J. Reichardt and D. R. White. Role models for complex networks. *Eur. Phy. J B*, 60(2):217–224, 2007.

[12] F. Wang, T. Li, X. Wang, S. Zhu, and C. Ding. Community discovery using nonnegative matrix factorization. *DMKD*, 22(3):493–521, 2010.

[13] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambr. Univ. Press, 1994.

[14] Y. Zhang and D. Yeung. Overlapping Community Detection via Bounded Nonnegative Matrix Tri-Factorization. In *KDD*, pages 606–614, 2012.