Increasing the Scalability of the Fitting of Generalised Block Models for Social Networks

Jeffrey Chan, Samantha Lam, Conor Hayes
Digital Enterprise Research Institute
NUI Galway, Ireland
{jkc.chan, samantha.lam, conor.hayes}@deri.org

Abstract

In recent years, the summarisation and decomposition of social networks has become increasingly popular, from community finding to role equivalence. However, these approaches concentrate on one type of model only. Generalised blockmodelling decomposes a network into independent, interpretable, labeled blocks, where the block labels summarise the relationship between two sets of users. Existing algorithms for fitting generalised blockmodels do not scale beyond networks of 100 vertices. In this paper, we introduce two new algorithms, one based on genetic algorithms and the other on simulated annealing, that is at least two orders of magnitude faster than existing algorithms and obtaining similar accuracy. Using synthetic and real datasets, we demonstrate their efficiency and accuracy and show how generalised blockmodelling and our new approaches enable tractable network summarisation and modelling of medium sized networks.

1 Introduction

As social network and media data becomes increasingly popular, there is an growing need to analyse and model them in a scalable manner. To understand these large networks, we need to reduce and summarise them to their underlying structure. Popular approaches, including community finding [Clauset *et al.*, 2004] and blockmodelling [Wasserman and Faust, 1994], aims to group the strongly associated vertices together. However, in some networks, not all the interesting groupings involve strongly associated vertices.

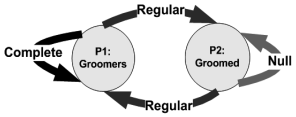
Consider the example of Figure 1, which represents the grooming behavior of a group of baboons [Doreian *et al.*, 2005]. Figure 1a is the adjacency matrix of the baboon grooming network, arranged into the two known groups – female baboons (a,c,d,f,h) who groom each other and the other baboons, and female and male baboons (j,k,b,e,g,i,l) who do not groom themselves but are groomed by the first group. Using a state of the art community finding algorithm [Rosvall and Bergstrom, 2008], we obtained only one partition, clearly illustrating the need to generalise to other definitions of network summarisation. This type of modelling is called *generalised blockmodelling*.

		P_1					P_2						
		a	c	d	f	h	j	k	b	e	g	i	l
P_1	a	0	0	1	0	0	0	1	1	1	0	0	1
	c	0	0	1	1	0	1	1	0	1	1	0	1
	d	1	1	0	1	0	1	0	0	1	1	0	1
	f	0	1	1	0	0	1	0	0	1	0	0	1
	h	0	0	0	0	0	1	0	0	1	1	1	0
P_2	j	0	1	1	1	1	0	0	0	0	0	0	0
	k	1	1	0	0	0	0	0	0	0	0	0	0
	b	1	0	0	0	0	0	0	0	0	0	0	0
	e	1	1	1	1	1	0	0	0	0	0	0	0
	g	0	1	1	0	1	0	0	0	0	0	0	0
	i	0	0	0	0	1	0	0	0	0	0	0	0
	l	1	1	1	1	0	0	0	0	0	0	0	0

(a) Rearranged adjacency matrix of the baboon network.

	P_1	P_2
P_1	Complete	Regular
P_2	Regular	Null

(b) Image matrix.



(c) Image diagram.

Figure 1: Adjacency matrix and image matrix and diagram for the Baboon grooming network.

Generalised blockmodelling [Doreian *et al.*, 2005] decomposes a network into partitions (groups of users) and assigns a relation type to each pair of partitions (called a block), which describe the relationship between them. Each block can have a different relation, making this a general modeling approach. Reconsider the baboon grooming network example of Figure 1. The blockmodel divides the grooming network into two partitions - P_1 and P_2. There are four possible relationships/blocks, illustrated as an image matrix in Figure 1b. On the diagonals, the complete type specifies that the baboons in the two partitions groom everyone else, while the null type specifies that the baboons do not groom anyone one else in the other partition. On the non-diagonal blocks, the regular type specifies that each baboon will at least groom one other in the other partition, and each baboon is at least groomed by someone else. The partitions with the relation types clearly shows and summarises the overall social structure of the baboon grooming behavior. The aim of generalised blockmodelling is to fit generalised blockmodels that summarizes the data into these interpretable blockmodels.

There are additional benefits of having multiple block types and directly incorporating block type labels into the model.

For example, in [Brendel and Krawczyk, 2009], the authors used a generalised blockmodel to distinguish between normal email communication patterns among communities and the abnormal patterns of external spammers by introducing new block types to represent the communications between communities and abnormal, external actors. Furthermore, directly incorporating block types into the problem allows confirmatory analysis to be easily performed, particularly important in social network analysis. E.g., to test relationship hypotheses, users can pre-specify block labels to parts of the blockmodel they want to test and as the labels are directly incorporated into the search process, the resulting blockmodels will reflect the initial hypotheses.

Up to now, the generalised blockmodel analysis of social networks has not received much attention, partly due to the computational demands of the existing algorithms. Therefore, we have designed approaches based on genetic algorithms and simulated annealing to fit generalised blockmodels. We have found both approaches are at least two orders of magnitude faster than the existing method. Using the increased scalability, we have fitted generalised blockmodels to the Enron email dataset that was previously too computationally difficult for the existing approach.

To summarize, the contributions of this paper are:

- We have designed two new algorithms that are at least 100 times more scalable than the existing method while matching its accuracy;
- As a demonstration, we have been able to fit generalised blockmodels to the Enron email dataset, previously too large to be fitted.

The rest of the paper is organized as follows. In Section 2, we introduce related work. Section 3 introduces our notation and formal concepts of generalised blockmodelling. We describe the existing and new blockmodelling approaches in Section 4, and present our results in Section 5. Finally, we conclude and present possible future work in Section 6.

2 Related Work

The summarisation of networks has been studied in a number of related areas. These include community finding, structural roles and blockmodelling. We discuss work in each of these areas and explain why generalised blockmodelling is different from each of them.

Community finding methods seek to find groups of vertices with more edges between its members than between its members and the rest of the network [Clauset *et al.*, 2004]. These approaches are able to find communities accurately, but community finding is a specialised problem of generalised blockmodelling. Community finding algorithms only fit blockmodels of complete block types along the diagonal of the image matrix and null types for all non-diagonal entries. It does not allow for other blockmodels to be explored.

Another method to group and classify vertices is by their structural position or roles [Borgatti and Everett, 1992][Lerner, 2004]. Vertices play the same structural role if they are linked to the same set of roles. Structural roles are intuitive, but are difficult to find in real life [Lerner, 2004]. Moreover, they only define one type of equivalence/block type on the whole network. One of the advantages of generalised blockmodelling is that it allows multiple definitions of equivalence throughout the network.

A similar approach to community finding is stochastic blockmodelling [Airoldi *et al.*, 2008]. It attempts to group vertices into groups by using statistical models to explain and learn the groupings [Airoldi *et al.*, 2008][Nowicki and Snijders, 2001]. These approaches are very powerful, but they are limited to dense and sparse block types, and do not allow pre-specification of block types and relationships, which is necessary for confirmatory analysis.

The closest work to ours is [James, 2010]. The authors proposed to use genetic algorithms to find blockmodels, via optimising an objective function that consists of maximising the diagonal block size, while satisfying hard density constraints. However, their definition of a blockmodel is basically dense diagonal blocks, while ignoring the density of non-diagonal blocks. We are able to map their hard density constraint to a soft (penalty) constraint. But we do not know how to treat the non-diagonal blocks adequately to fairly evaluate the objective of [James, 2010], hence we cannot directly compare their approach with ours and incorporate their formulation into our framework. We will describe the representation behind [James, 2010] in more detail in Section 4.3, as it is similar to the representation we are proposing.

3 Background

In this section, we formally define the generalised blockmodelling problem and introduce the notation we will use. A generalised blockmodel consists of grouping the vertices into partitions and an assignment of the types between all pairs of partitions. We use the nine standard ones provided by [Doreian *et al.*, 2005], but we only describe the five types illustrated in this paper in Section 3.1. We stress that the set of permitted block types can be extended or modified as needed. To fit a blockmodel to describe a given network, we formulate it as an optimisation problem, following the approach of [Doreian *et al.*, 2005].

A network $G(V, E)$ consists of a set of vertices V, and a set of edges E (E: $V \times V$). The edge relation can be represented by an adjacency matrix $\mathbf{A} = \mathbf{A}(G)$. Let the set of partitions, be denoted by $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$, where $P_1 \subseteq V$, $P_i \cap P_j = \emptyset, i \neq j$ and $\cup_i P_i = V$. Let $\mathcal{T}$ denote the set of possible block types, $\mathbf{B}(P_i, P_j)$ to denote the block type between two partitions and $\mathbf{B}$ to denote the image matrix that represents all pairwise block assignments. A generalised blockmodel is then defined as $\mathcal{S}(\mathcal{P}, \mathbf{B})$.

Let $\mathbf{A}(P_i, P_j)$ be a submatrix of the adjacency matrix $\mathbf{A}$, where $P_i, P_j \in \mathcal{P}$, and let $d(\mathbf{A}(P_i, P_j), \mathbf{I}(t))$ denote the error or distance between the block $\mathbf{A}(P_i, P_j)$ and the ideal submatrix structure of block type $t \in \mathcal{T}$. The objective cost of blockmodel $\mathcal{S}(\mathcal{P}, \mathbf{B})$ is defined as

$$C(\mathcal{S}(\mathcal{P}, \mathbf{B})) = \sum_{P_i, P_j \in \mathcal{P}} \min_{t \in \mathcal{T}} d(\mathbf{A}(P_i, P_j), \mathbf{I}(t)).$$

It measures the difference between an ideal blockmodel, with all blocks conforming to an ideal block structure, and the

blockmodel $\mathcal{S}(\mathcal{P}, \mathbf{B})$. We use the ideal block types described in Section 3.1 and block distance function defined in [Doreian *et al.*, 2005]; please refer to that reference for more details.

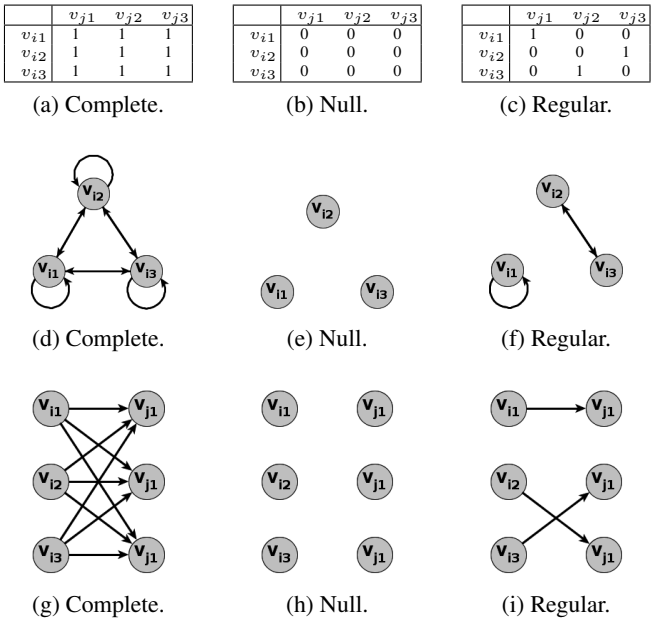
The generalised blockmodelling fitting problem is to find a set of partitions $\mathcal{P}$ and a block type assignment $\mathbf{B}$ that minimizes $C(\mathcal{S}(\mathcal{P}, \mathbf{B}))$.

3.1 Block Types

Each block type defines a relationship pattern/class between the interactions of two partitions. We describe five of the nine block types defined in [Doreian *et al.*, 2005], and illustrate three of them in Figure 2.

Let $P_i = \{v_{i1}, v_{i2}, \ldots, v_{in}\}$ and $P_j = \{v_{j1}, v_{j2}, \ldots, v_{jm}\}$ represent the two partitions. For each block type, there are two types of relationships: diagonal blocks ($P_i = P_j$) and non-diagonal blocks ($P_i \neq P_j$). In Figure 2, the first row is the ideal adjacency submatrix of the block, the second row is the corresponding graph structure for diagonal blocks, and the third row is the corresponding graph structure for non-diagonal blocks.

A complete block (first column of Figure 2) has an edge between all pairs of vertices in the two partitions, and represent complete connectivity. A null block (second column of Figure 2) has no edges between the partitions, and represent absence of connectivity. Finally, a regular block (third column of Figure 2) has two conditions: a) at least an edge originating from every vertex in the row partition P_i, and b) at least one edge coming into every vertex in the column partition P_j. Unlike the other two types, there are several substructures that can satisfy regular type and Figure 2 only shows one example. Row regular and column regular block types are not illustrated, but they are relaxation of the regular block type. Row regular only requires condition a) of regularity and column regular only condition b).



(a) Complete. (b) Null. (c) Regular.

(d) Complete. (e) Null. (f) Regular.

(g) Complete. (h) Null. (i) Regular.

Figure 2: Three block types and their ideal adjacency and block patterns. The example for the regular type is just one possibility. First row is the adjacency matrix, second and third row are the graph representation for diagonal and non-diagonal blocks respectively.

4 Algorithm

We describe three different approaches to fit generalised blockmodels. The first method, the KL-based approach, is a greedy method based on the well-known Kerninghan-Lin graph partitioning algorithm [Kernighan and Lin, 1970]. This KL-based approach is currently the only approach proposed to fit generalised blockmodels. In this paper, we propose two new approaches: one is based on simulated annealing, while the other is based on genetic algorithms.

4.1 KL-Based Algorithm

In [Doreian *et al.*, 2005], Doreian et al. proposed the greedy KL-based approach to fit blockmodels. The algorithm considers the solution neighborhood of each vertex, and greedily makes a move that reduces the objective cost the most. A neighborhood move is either a) a vertex moving from one partition to another, and b) the swapping of two vertices in different partitions. The authors did not describe how to fit the blocks types themselves. Therefore we introduce an additional step, where the blocks types are greedily optimized after the partitions are optimised.

The problem with this greedy approach is that the algorithm often gets stuck in local minimums and there are many expensive objective evaluation per run. And to get decent results and allow the adequate search of the solution space, the algorithm must be run many times – in fact, Doreian suggested to run this algorithm 50,000 times when fitting a blockmodel to a bi-partite 20 by 14 network [Doreian *et al.*, 2005].

4.2 Simulated Annealing

A successful approach to improving greedy searches by exiting local minimums is simulated annealing (SA). We use the standard SA approach [Johnson *et al.*, 1991], but with problem specific differences in the generation of an initial random solution, the objective cost function, and the generation of a neighborhood. In our case, a neighborhood operation is either a swap, move or block type change. We name this algorithm *saBM*

4.3 Genetic Algorithm

Genetic algorithms (GA) are known to be able to tackle hard combinatorial optimisation problems. There are some existing genetic algorithms work for partitioning problems [Bui and Moon, 2002], but as far as we know, the closest work to tackling generalised blockmodeling using genetic algorithms is [James, 2010]. Although they used a similar representation to this work, our representation additionally encodes the block types. Also in [James, 2010], mutation is not considered important, but for generalised blockmodelling it is needed to escape local minima. Furthermore, although the recombination of the partitioning of both approaches follow a similar process, our approach also recombines the type assignments, which is not a trivial issue. Therefore, our work

Parameter	Description	Default
popSize	Population size.	200
maxGen	Max no. of generations	1000
maxSteadyGen	Max no. of generations with no improvement in objective.	10
CrossoverPts	No. of crossover points	2
probCrossover	Prob. of crossover	0.6
probMut	Prob. of mutation	0.1

Table 1: Parameter settings of *gaBM* used in the fitting of the datasets.

and [James, 2010] are related, but there are differences in the problem solved and the solution.

Our genetic algorithm, named *gaBM* consists of four standard steps: selection, recombination, mutation and replacement (see Algorithm 1 for an outline). Initially, a population of *popSize* number of blockmodels are randomly generated (line 3 of Algorithm 1). Then *probCrossover* % of the population is selected for recombination using a deterministic tournament selection process of 2 chromosomes (line 6). Next, *probMut* % of the population are randomly selected for mutation (line 8). Finally, we use the plus replacement strategy to choose the next generation from the starting and the recombined and mutated population (line 9). We also employ an elitism strategy to keep the best solution. This process is repeated until either there is no improvement in the objective cost for *maxSteadyGen* number of generations, or we have iterated over *maxGen* number of generations.

After experimentation, we used the parameter settings outlined in Table 1. We describe the representation, crossover and mutation operators in the following.

Algorithm 1 Outline of the *gaBM*

1: **Input:** A graph, parameters of Table 1.
2: **Output:** A population of blockmodels
3: *Initialization:* a population of *popSize* no. of blockmodel solutions
4: gen = 0, steady = 0
5: **while** steady $<$ *maxSteadyGen* or gen $<$ *maxGen* **do**
6: *Selection:* Deterministic 2 chromosome tournament selection
7: *Recombination:* See section 4.3:Crossover Operator
8: *Mutation:* See section 4.3:Mutation Operators
9: *Replacement:* plus strategy + elitism (keep best solution)
10: gen++
11: steady = 0 **if** improvement in objective of best solution **else** steady++
12: **end while**

Representation

In the literature, a solution representation (a chromosome) called the grouping genetic algorithm (GGA) has been proposed [Michalewicz, 1996]. Unlike the popular integer representation [Michalewicz, 1996], which only stores the assigned partition of each vertex, the GGA representation additionally stores explicit partition information. Consider the blockmodel chromosomes for the baboon network augmented with partition information, illustrated in Figure 3a. Left of the vertical line of each chromosome (S_1 and S_2), each element (called a gene) represent a vertex and its assigned partition. Right of the vertical line, the three partitions represented in each chromosome are encoded as three extra genes. The addition of the partition genes allows operators to work on interchanging partitions explicitly, which avoids the redundancy and isomorphism problems of the integer representation (see [Michalewicz, 1996] for problems with the integer representation).

In addition to the partitioning information, our representation also encodes the block type assignments. For k-partitions, the assignments are represented by the $k \times k$ block type matrix. Each matrix row/column is mapped to a partition. The whole chromosome representation of *gaBM* is illustrated in Figure 3a.

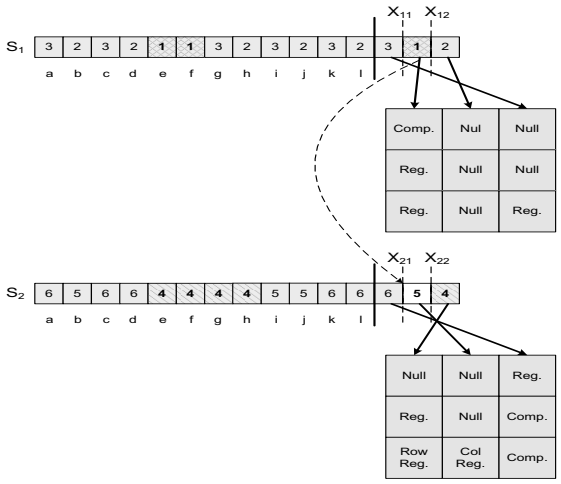
Crossover Operator

There is an crossover operator defined for the GGA representation; however, it is designed for the partitioning problem only, hence we extend and modify the operator to include the block type matrix. Given two parent chromosomes for crossover, the main idea is to choose a subset of partitions to inject from one chromosome into another. After injection, we repair any resulting inconsistencies.

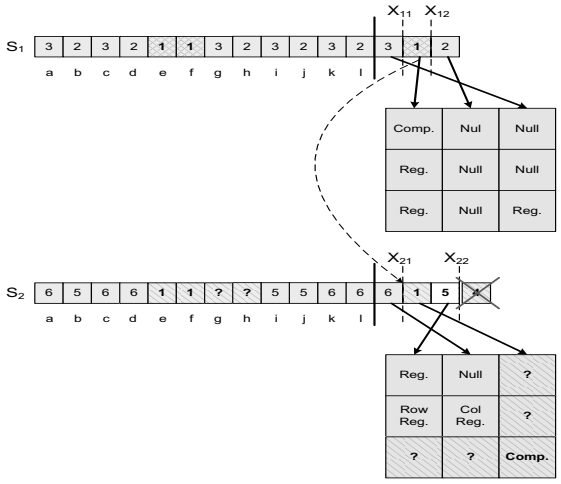
To help explain the crossover steps, we use the baboon example again; the various steps are illustrated in Figure 3 and we shall refer to this as we describe the crossover operation step by step.

1. Pick two crossover points in each of the two parent chromosomes S_1 and S_2. Let the two crossover points be labeled X_{11} and X_{12} for S_1 (X_{21} and X_{22} for S_2). This is illustrated in Figure 3a, where partition 1 is about to be inserted into S_2.
2. Let the partitions enclosed between X_{11} and X_{12} of S_1 (partition 1) be denoted by I_1. Inject the partitions of I_1 into crossover point into X_{21} of S_2.
3. If a vertex v_i in partition P_k of S_2 also exists in any partition P_l of I_1, delete partition P_k. This is to avoid the growth of too many partitions. In the example (see Figure 3b), partition 4 of S_2 has elements 'e' and 'f' overlapping with partition 1 of S_1, so it will be deleted.
4. For all vertices in S_2 that are not assigned to any partition (due to partition deletions), evaluate each partition in S_2 and insert into the partition that results in the best objective cost. For example, elements 'g' and 'h' are redistributed to the best partitions 6 and 1 respectively in Figure 3c.
5. Repeat steps 2 to 4 for injecting the partitions of X_{21} and X_{22} into S_1.

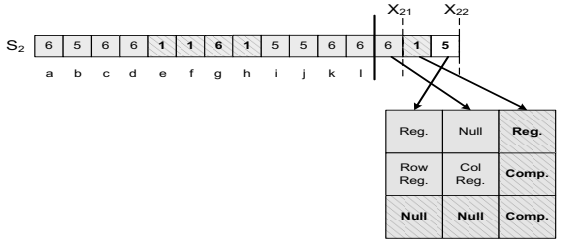
The schemata principle of genetic algorithms suggests good parts of solutions should be mixed between chromosomes. As the goodness of the block type assignments are closely dependent on the partitioning, as much of the assignment information of the injected partitions should be maintained and are not independently crossovered. However, we only have good assignment information among the injected partitions themselves (in Figure 3b, this is $\mathbf{B}(\mathcal{3}, \mathcal{3})$ of S_2) , not between the injected partitions and the existing partitions ("?" in the image matrix of S_2 of Figure 3b). For those assignments, we try each possible block type and choose the best type assignment. At the end of this crossover operator, we get two new offspring that share parts of the parents.



(a) The partitions crossovered are the partition ids listed between the crossover points X_{11} and X_{12} for S_1 and X_{21} and X_{22} for S_2.



(b) Partition 4 of S_2 has elements 'e' and 'f' overlapping with partition 1 of S_1, so it is deleted. Elements 'g' and 'h' are no longer associated with any partition.



(c) Elements 'g' and 'h' are redistributed to the best partitions 6 and 1 respectively.

Figure 3: Crossover example of the baboon brooming network. Partition 1 of S_1 is inserted into S_2. It causes partition 4 to be deleted, and vertices at positions 'g' and 'h' of chromosome S_2 are redistributed.

Mutation Operators

We have designed three different operators to perturb the existing chromosomes: swap two random vertices between two randomly chosen partitions, move a random vertex from a randomly chosen partition to another, and change a block type assignment to another allowable type. Generally, mutation is not as important as crossover. However, because the crossover operator does not crossover block type assignments directly, the assignment mutation operator is important to allow searching over assignment. Hence, we set the assignment mutation operator to occur on average three times more often than the other two mutation operators.

5 Evaluation

In this section, we evaluate the efficiency and optimisation performance of the three algorithms.

5.1 Datasets

To measure the scalability and optimising ability of the algorithms, we generated synthetic datasets using the community generating algorithm of Lancichinetti et al. [Lancichinetti *et al.*, 2008]. Briefly, the algorithm of Lancichinetti et al. generates a number of communities that have a set total number of vertices. The degrees of vertices and community sizes follow a power law (in this paper, exponent of 2 and 1 respectively). In addition, the amount of inter-community connections are governed by a mixing parameter, μ, with lower μ indicating more inter-community connections. In our experiments, we found μ had negligible effect on the performance of the three algorithms, hence we do not present the results of varying this parameter.

For real datasets, due to space limitations, we only demonstrate the fitting of the Enron email dataset. The Enron email corpus is a set of emails collected from the Enron corporation over 3.5 years and is known to have a job title hierarchy as detailed in [Diesner *et al.*, 2005]. Our goal is to infer these job title rankings from the interaction of the employees' emails. Moreover, we split the dataset into three time periods to reflect the interactions before, during and after the spreading of information of the 2001 crisis.

5.2 Generated Community-Blockmodel Results

We evaluate the performance of *KL-based*, *gaBM* and *saBM* algorithms on the synthetic community datasets, with and without the block types supplied to the algorithms. The task is easier when the block types are pre-specified by the user, as it becomes finding the best partitioning of the vertices that minimizes the objective costs according to the specified block types.

For each parameter setting, we generate 10 different community-model networks, ran the algorithms and analyzed the average objective cost (lower is better) of the fitted blockmodel and the average running time. All the experiments were conducted on a dual Xeon 2.27GHz server with 32GB of memory and running Ubuntu 10.04.

We first analyze when the blockmodels are not supplied, which is the most difficult fitting task. The results are illustrated in the top row of Figure 4. We first present the results

for two network sizes, 50 and 100 (Figures 4a and 4b), because the *KL-based* algorithm can only realistically fit models to networks of these sizes, and then the results for networks up to 800 vertices (Figures 4c and 4d), for the *gaBM* and *saBM* algorithms.

Consider the performance comparison of the three algorithms first (Figures 4a and 4b). Figure 4a show that the *gaBM* obtains similar quality solutions to the *KL-based* algorithm, with *saBM* results having three times the objective cost. Speed wise, Figure 4b show that *gaBM* is slightly slower than *saBM* but almost two factors faster than the *KL-based* algorithm. For small networks, this again shows that *gaBM* is a good balance between speed and accuracy.

For larger networks, the results illustrated in Figures 4c and 4d show that *gaBM* can obtain blockmodels with smaller objective costs (the 400 vertex result is due to one result having an objective cost of 2×10^5, which distorts the averages), but about one factor slower than *saBM*. The results indicate that for larger networks, if speed is important, than the *saBM* algorithm should be used, but if accuracy is more important, than the *gaBM* should be used.

Now consider the results where the known image matrix of a community-like blockmodel is used to initialise the type assignments of the algorithm (see bottom row of Figure 4). The results have similar behavior to the previous no blockmodel results. However, for larger network sizes (see Figures 4g and 4h), *gaBM* benefits greatly from the pre-specified blockmodel, with objective costs significantly improved over *saBM* and running times only 3-4 times slower. This indicates that for larger networks, having a pre-specified blockmodel can assist *gaBM* to better results.

5.3 Enron Email

We used the *gaBM* algorithm to explore this dataset over three time periods - prior, during and after the crisis, denoted T_1, T_2, and T_3, respectively. The number of vertices (and edges) for the graphs of T_1 to T_3 are 270 (1202), 264 (1718) and 284 (1876) respectively. As a guide to the communications between the employees we used results found by Diesner et al. [Diesner *et al.*, 2005] to help us construct our block type matrices. Moreover, to overcome the possibility of over-fitting the roles, as well as due to the similar behavior of some roles, we search for blockmodels of five partitions/roles.

The population parameter of the algorithm was varied between 80 to 120 and we set it to halt when it reached a steady state of 20 generations. The average (standard deviation) of the running time taken for each population range from 516.14 secs (158.27) to 1242.84 secs (212.01).

We specified our blockmodels according to the following five job titles or roles (in order of decreasing seniority): Executive Management, Senior Management, Lawyers, Managers/Traders, and Specialists/Associates. To rate the accuracy of our findings we computed the Variation of Information (VI) for comparing clusterings as described by Meila [Meila, 2003]. The 'gold standard clustering' with which we compared with were the known roles of each person. We found the optimal blocks for each time step along with their VI value (lower is more similar) and objective cost. As Table 2 shows, the VI values for the blockmodels are reasonably low, suggesting they are a decent fit to the known roles.

	00000		04141		05666
	40551		10615		10000
T_1	41010	T_2	14004	T_3	50014
	05505		41001		40601
	00410		10610		66660
VI	12.6	VI	28.2	VI	17.1
Obj. Cost	2358	Obj. Cost	5283	Obj. Cost	7351

Table 2: Optimal image matrices for each time step along with their VI value and objective cost, where numbers represent blocks such that 1 is complete, 0 is null, 4 is row-regular, 5 is column-regular, and 6 is regular.

The communication of the emails is such that emails are sent from row to column and one receives emails from column to row. For example, the image matrices in Table 2 show that the communication pattern of the executive managers, although little variation amongst themselves, increases from almost no communication at T_1 (with only little received from senior management and lawyers) to almost complete communication to all others at T_2, especially to lawyers and specialists/associates in both directions. Then communications from T2 to T3 change from complete to mostly regular types, reflecting that the executive management are reducing their communications and crisis management, possibly because they knew the company was about to declare bankruptcy.

Our findings are consistent with Diesner et al. [Diesner *et al.*, 2005]. The fitted generalised blockmodels summarise the roles, the key relationships (block types) between the different roles, and demonstrates that the *gaBM* algorithm can find interesting and useful blockmodels for larger sized real networks.

6 Conclusion

In this paper we presented two new algorithms for fitting network data to a generalised blockmodel. The algorithms makes use of the strengths of simulated annealing and genetic algorithms in tackling hard combinatorial optimization problems. Compared to the existing *KL-based* algorithm, we demonstrated their superior efficiency and accuracy on generated community networks, and using the *gaBM* algorithm illustrated the fitting of useful generalised blockmodels to the Enron dataset, which the *KL-based* method cannot do.

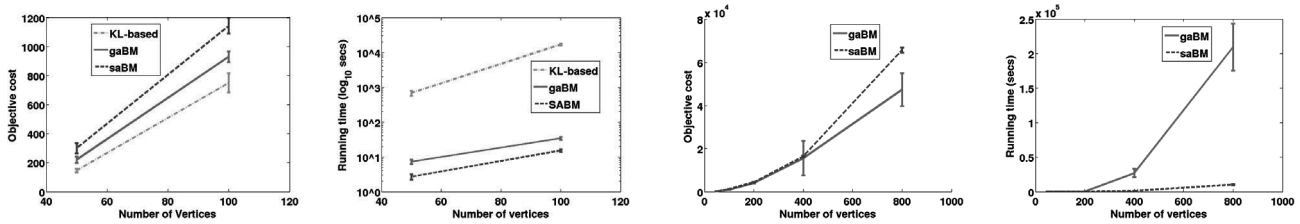
For future work, we aim to investigate additional ways to measure and rank discovered blockmodels. In addition, we wish to explore further speedups, possibly using hierarchical compression techniques or combining the *saBM* and *gaBM* algorithms.

7 Acknowledgments

This work was carried out in the CLIQUE Strategic Research Cluster which is funded by Science Foundation Ireland (SFI) under grant number 08/SRC/I1407.

References

[Airoldi *et al.*, 2008] E. Airoldi, D. Blei, S. Fienberg, and E. Xing. Mixed membership stochastic blockmodels.



(a) Objective cost vs. number of vertices, 3 algorithm comparison.

(b) Running time vs. number of vertices, 3 algorithm comparison.

(c) Objective cost vs. number of vertices, *gaBM* and *saBM* comparison.

(d) Running time vs. number of vertices, *gaBM* and *saBM* comparison.

(e) Objective cost vs. number of vertices, 3 algorithm comparison.

(f) Running time vs. number of vertices, 3 algorithm comparison.

(g) Objective cost vs. number of vertices, *gaBM* and *saBM* comparison.

(h) Running time vs. number of vertices, *gaBM* and *saBM* comparison.

Figure 4: Objective cost and running time results for the generated community datasets. Top row of figures: No Blockmodel supplied. Bottom row: Blockmodel supplied.

Journal of Machine Learning Research, 9:1981–2014, 2008.

[Borgatti and Everett, 1992] S. Borgatti and M. Everett. Notions of position in social network analysis. *Sociological Methodology*, 22:1–35, 1992.

[Brendel and Krawczyk, 2009] R. Brendel and H. Krawczyk. Extended generalized blockmodeling for compound communities and external actors. In *Proceedings of CASoN*, pages 32–39, 2009.

[Bui and Moon, 2002] T.N. Bui and B.R. Moon. Genetic algorithm and graph partitioning. *Computers, IEEE Transactions on*, 45(7):841–855, 2002.

[Clauset *et al.*, 2004] A. Clauset, M. Newman, and C. Moore. Finding community structure in very large networks. *Phys. Rev. E*, 70(6):066111, Dec 2004.

[Diesner *et al.*, 2005] J. Diesner, T. Frantz, and K. Carley. Communication Networks from the Enron Email Corpus It's Always About the People. Enron is no Different. *Computational & Mathematical Organization Theory*, 11(3):201–228, 2005.

[Doreian *et al.*, 2005] P. Doreian, V. Batagelj, and A. Ferligoj. *Generalized blockmodeling*. Cambridge University Press, 2005.

[James, 2010] T James. Grouping genetic algorithm for the blockmodel problem. *IEEE Transactions on Evolutionary Computing*, 14(1):103–111, 2010.

[Johnson *et al.*, 1991] D. Johnson, C. Aragon, L. Mcgeoch, and C. Schevon. Annealing : An Experimental by Simulated Optimization Part 11 , Graph Coloring and Evaluation ; Number Partitioning. *Operations Research*, 39(3):378–406, 1991.

[Kernighan and Lin, 1970] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307, 1970.

[Lancichinetti *et al.*, 2008] A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Phy. Rev. E*, 78(4):46110, 2008.

[Lerner, 2004] J. Lerner. Role assignments. In *Network Analysis*, pages 216–252, 2004.

[Meila, 2003] M. Meila. Comparing clusterings by the variation of information. In *Proceedings of the COLT/Kernel*, pages 173–187. Springer Verlag, 2003.

[Michalewicz, 1996] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs (3rd ed.)*. Springer-Verlag, London, UK, 1996.

[Nowicki and Snijders, 2001] K. Nowicki and T. Snijders. Estimation and prediction for stochastic blockstructures. *Journal of American Statistical Association*, 96(455):1077–1087, 2001.

[Rosvall and Bergstrom, 2008] M Rosvall and C T Bergstrom. Maps of random walks on complex networks reveal community structure. *PNAS*, 105:1118–1123, 2008.

[Wasserman and Faust, 1994] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1 edition, 1994.