Generalised Blockmodelling of Social and Relational Networks using Evolutionary Computing

Jeffrey Chan, Samantha Lam and Conor Hayes

Received: January 17, 2014/Accepted:

Abstract Blockmodels has been studied for many years in social network analysis. In blockmodelling, the goal is to uncover a network into its underlying, latent structure. It does so by reducing a graph into a set of roles, where each role is a set of vertices having similar interactions with other roles. For example, we could decompose an airport routing network into a core-periphery structure of hub and spokes airports. Much prior work has studied how to use probabilistic graphical models to find blockmodels, but there has been little work in allowing users' supervision and feedback into the fitting and modelling process. Generalised blockmodelling allows users to pre-specify role-to-role interaction patterns and other constraints, which allows users to test models and relationship hypotheses and directly incor-

porate known biases and constraints. In addition, generalised blockmodelling allows multiple definitions of relational equivalences to co-exist simultaneously in a single model, permitting more general models. The existing approaches to fit generalised blockmodels are not scalable beyond networks of 100 vertices. In this article, we present two new algorithms, a genetic algorithm-based and a simulated annealing based approach, that are multiple times faster than existing approaches. We also demonstrate the usefulness of generalised blockmodelling by fitting to several medium-sized real world networks that previous methods were not able to analyse, and evaluate the efficiency and accuracy on synthetic datasets.

Jeffrey Chan

Department of Computing and Information Systems, University of Melbourne, Australia

(The author conducted part of this work at the Digital Enterprise Research Institute)

E-mail: jeffrey.chan@unimelb.edu.au

Samantha Lam and Conor Hayes

Digital Enterprise Research Institute, NUI Galway, Ireland

E-mail: samantha@deri.org

1 Introduction

As more and more social network and media data becomes available, there is an increasing need to analyse, model and understand them in a scalable manner. To understand these networks, we need to reduce and summarise them to their underlying structure. One popular approach is to group the related vertices or actors together. Previous work in community finding [44] group

Fig. 1: Baboon grooming network example.

vertices into strongly associated partitions. However, in some networks, not all the relevant groupings involve strongly associated actors. Consider the example of Figure 1, which represents the grooming behaviour of a group of baboons [16]. Figures 1a and 1b are the adjacency matrices of the baboon grooming network, arranged according to one of the best community finding algorithms at present [44] and an alternative grouping. The alternative grouping consists of three groups (see Figures 1b) – P_1 : baboons who groom within their own group and the two other groups; P_2 : baboons who only groom baboons from P_1 ; and P_3 : baboons who do not groom among themselves but are groomed by others from P_1 . This decomposition was obtained by using only link information. In fact, this partitioning corresponds to females who groom themselves and other baboons (P_1), females who only groom other females

from P_1 (P_2) and finally, males who are groomed by females from P_1 (P_3). Figure 1c illustrates the alternative grouping of Figure 1b in a network form, with female baboons represented by green, dotted bordered vertices, and males by yellow, solid bordered vertices. Contrast this with the decomposition by a popular community algorithm [44] (result illustrated in Figure 1a). One group was optimal according to the community definition, clearly illustrating the need to consider other definitions of network summarisation apart from strongly self-associated groups. This type of modelling is called *generalised blockmodelling*.

Generalised blockmodelling [16] decomposes a network into positions, and assigns a relation type to each pair of positions to describe the relationship between the positions. Together, the set of positions and the assigned relational types summarises the underlying struc-

Fig. 2: The image matrix and diagram of the three position blockmodel of the baboon network of Figure 1.

tural of the network/relational data. Reconsider the baboon grooming network example of Figure 1. The blockmodel divides the grooming network into three positions - P_1 , P_2 and P_3 . Among the positions, there are $3 \times 3 = 9$ possible inter-position relations, illustrated as an image matrix in Figure 2a. On the diagonals, the *complete* type specifies that the baboons in P_1 groom among themselves, while the *null* types of P_2 and P_3 specifies that the baboons do not groom any other baboons within their respective positions. On the non-diagonal blocks, the *regular* type specifies that each baboon will groom at least one other in the other position, and each baboon is groomed by at least one other (e.g., P_1 to P_3). The positions with the relation types (specified in the image matrix/image diagram) clearly shows and summarises the overall social structure of the baboon grooming behaviour.

Blockmodelling [51] has been an active research area for many years. The existing models [3][43][1] and algorithms have successfully found interesting explanations

about the underlying structures of networks. However, we stress that these types of blockmodelling are different to generalised blockmodelling, as they do not optimise or assign relation types, like in Figure 2. Instead, they either concentrate on finding dense or sparse interaction patterns, and/or use parameterised probability distributions to model prior knowledge, and their final output is of the form of Figure 1c. Hence, these blockmodels are limited to discovering one type of equivalence; they do not and cannot find the block type labels. In addition, it is not intuitive to prespecify domain knowledge and known constraints by parameterised probability distributions. As generalised blockmodelling allows users to pre-specify and test different block types (which is a form of hypothesis of relationships), and naturally pre-assign membership and add other constraints, it means that a confirmatory type of analysis can be much more easily performed.

Furthermore, it is simple to introduce new block types, which means other relationship types can be explored. As an example, in [6], Brendel et al. used a generalised blockmodel to distinguish between normal email communication patterns among communities and the abnormal patterns of external spammers. They introduced additional block types to represent normal and abnormal communications between communities and external actors. This allows normal and abnormal external communication patterns to be clearly identified. As this example demonstrates, generalised blockmodelling can complement existing types of blockmodelling by easily allowing multiple models within one framework, which can be used to discover the block types and for confirmatory type of analysis.

Up to now, the generalised blockmodel analysis of social networks has not received much attention, partly

due to the computational demands of the standard generalised blockmodelling algorithm [16], which does not scale. Because the current state-of-the-art algorithm [16] does not scale beyond networks of 100 vertices, generalised blockmodelling could only be applied to a limited number of small networks. In addition, even though there are density-based blockmodelling approaches that scale beyond 100 vertices [53], the blockmodelling problem they solve do not have nor consider block types. Therefore, these existing approaches cannot be used to solve the generalised blockmodelling approach, as it requires the optimisation of the block types.

In earlier work [10], we proposed two new heuristic approaches based on simulated annealing and genetic algorithms to fit generalised blockmodels. We showed that both approaches are at least two orders of magnitude faster than the existing method using a synthetic dataset and fitted insightful generalised blockmodels to the Enron email communications network. In this article, we provide more mathematical detail into the generalised blockmodelling problem. We also describe each of the block types used and examples of where they are applicable. We provide more detail about our two algorithms, present more synthetic testing results and include the blockmodel decomposition of two additional, real datasets, namely a communication network among the participants at a doctoral summer school and an airport routing network.

To summarise, the contributions of this paper are:

- We propose two new algorithms to find generalised blockmodels that are at least two orders of magnitude more scalable than the existing method while matching its accuracy. This allows generalised blockmodel analysis to analyse larger networks.

- We describe previously introduced block types and provide guidance into their usage by explaining what they represent in real networks and where they might be used.
- We demonstrate, via real world and synthetic networks, that generalised blockmodelling is a valuable general and confirmatory modelling tool.

The rest of the paper is organised as follows. In Section 2, we introduce the background and related work in role-mining and analysis on social networks. Section 3 gives our notation and introduces the basic concepts in generalised blockmodel analysis. We describe the current blockmodelling algorithm and optimisation procedures in Section 4. We present our results in Section 5 from several traditional social network analysis datasets, synthetic datasets and three real datasets. Finally, in Section 6 we conclude the paper and provide pointers to future work.

2 Related Work

Analysing the structure and summarising networks has been studied in a number of related areas. These include community-finding, role equivalences, blockmodelling and feature-based role analysis. We discuss work in each of these areas and explain why generalised blockmodelling is different from each of them.

Community Finding: One popular approach to summarising networks is to find groups of vertices with more edges between its members than between its members and the rest of the network. In social networks, these groups of vertices can be interpreted as social communities (e.g, groups of friends, work colleagues etc) [13][22][44][23] or in dependency graphs these groups

can represent strongly functional modules [32]. A number of approaches have been proposed to find these groups or communities, including optimising a criterion called modularity [13]. Modularity compares the amount of association within a community against the expected association in the same set of vertices, but with their edges randomly redistributed. In addition, graph partitioning approaches have been proposed [54][21], where the whole graph is partitioned or cut into groups such that the number of cut edges (or total of weights of cut edges for weighted graphs) is minimised. Another approach is to find maximal quasi-cliques [42], where quasi-cliques are subgraphs whose density is above some specified threshold. All these approaches are known to be able to find communities accurately, but (generalised) blockmodelling is a generalisation of community finding. Community-finding can be considered as a blockmodelling problem, by seeking community-like blockmodels that have complete block types along the diagonal of the image matrix and non-diagonal entries being null. Hence, community finding algorithms are restricted to the fitting of one type of blockmodel, and do not allow for other blockmodels to be explored.

Role Equivalence: Another method to group and classify vertices is by their structural position or roles [5][36][7]. Vertices are considered to be in the same structural position or role if they are related/interact with similar sets of roles. As an example, reconsider the Baboon network from Section 1 (Figures 1 and 2). The female only groomers (P_2) are in the same structural position because they more or less interact with the other females of P_1 , and have no interactions with the males of P_3 .

Depending on the definition of what is considered as a role¹, different groupings are possible. In the literature, the two most popular role definitions/equivalences are the structural and regular equivalences [5]. Individuals are structurally equivalent if they are connected to the same individuals [37]. For example, computer science professors at an university teaches the same set (more or less) of computer science students. However, structural equivalence is generally considered too restrictive for real social networks; in the example, professors have to teach the *exact* same set of students to be considered to play the same professor role. Hence, regular equivalence was proposed [36][49]. In regular equivalence, individuals are considered to be equivalent/play the same role if they are connected to the same *set of roles*. Returning to the university example, a professor is defined as an individual who teaches (is connected to) students (any individual who play the student role). Hence, two professors in different universities can be regularly equivalent as long as they are connected to individuals who play the student role. Role equivalences can model more general relationships and roles, but can be difficult to find exactly in real life [36]. Moreover, role equivalence defines 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 rather than using one single definition throughout the network.

Other Blockmodelling Approaches: Blockmodelling was initially introduced in sociology [51][49]. It was used to find structural equivalence, where the blocks consisted of either total connectivity (all 1s) or no connectivity (all 0s). Stochastic blockmodels and latent models [3][1] relax the presumption of total and no connec-

¹ A role induces an equivalence relation in this context.

tivity. Edges between vertices are modelled as random variables of a probability distribution; i.e., the probability of an edge existing between two vertices is dependent on the edge generation distribution and its parameters. The distribution and the dependent parameters lead to different formulations and probabilistic models, including stochastic blockmodels.

In [3] and [41], the unit of analysis is a dyad (reciprocal edges are considered as one unit), to avoid the problem of assuming dependence or independence between them. Nowicki and Snijders [41] models the likelihood of an dyad existing dependent on the positions/roles of their incident vertices. The positions are assumed to be independent and drawn from Dirichlet distributions.

Airoldi et al. [1] extended the model of [41] by allowing users to belong to multiple positions - i.e., a mixture membership stochastic blockmodel. Airoldi et al. also modelled the position-to-position interactions, and allowed users to play different roles depending on who they are communicating with. In [53], Xing et al. extended the mixed membership model of [1] to dynamic networks. The position membership vectors of users and blockmodels are allowed to change with time.

Hoff et al. [29], and later Hancock et al. [27], pursued a different approach. They assumed the users are projected as points in a latent social space, and the likelihood of an edge existing between actor is dependent on their distance in this social space. Users who are likely to link will have small distances and similar set of features. Advantages of this approach are that users can have different probabilities of linking with each other, not just based on their roles, and it can directly incorporate homophily of user features. But the disadvantages are that the concept of positions and roles is no longer

explicit and it requires specification of the social space, which might not be a trivial task for some datasets.

All these statistical approaches can infer insightful models, but in general their fitting process can be computationally demanding. More importantly, for confirmatory analysis, the parametric models require setting of parameters, which are hard to translate from practical requirements (e.g., how to translate the parameter settings to the scenario where we want each member in a position to have a edge to at least one other member in another position) and they do not model the different possible block types into their inference. These aspects are necessary for generalised blockmodelling and a confirmatory type of analysis.

Role Analysis: There have been some alternative approaches to role detection by grouping the features of users. For example, in [25], roles like celebrity, newbie, lurker, flamer, troll and ranter have been qualitatively described. Similarly, roles like mentor and manager have been proposed in [47]. Although many roles have been proposed, there are no empirical or quantitative validation of these roles, which limits their usefulness for unexplored real-world data. The work in [20][50] provided analysis of a user's ego-centric network to determine roles such as technical editor and substantive expert in Wikipedia. These approaches had a significant element of manual analysis and thus are not easily scaled. The recent work of Chan and Hayes [9] derived structural and behavioural features from the ego-centric reply structure network of online forum users. Users were clustered and role features extracted from the most salient features in each cluster. These roles were then used to interpret forum functionality. While this is an automated and scalable approach that pro-

duces useful insights, its notion of a role does not capture the inter-dependencies between different positions.

Algorithmically, the closest work to ours is [30]. The authors proposed to use genetic algorithms to find communities, via optimising an objective function that consists of maximising the size of the communities subject to each community having an edge density greater than a specified density threshold. This objective formulation is different from generalised blockmodelling. The closest model is one where the diagonal blocks are of the density type. However, as the objective of [30] do not optimise or consider the non-diagonal blocks, it is not obvious which block types would be adequate if we wanted to compare generalised blockmodelling to the community objective of [30] (we describe the formulation of [30] in Appendix B). Hence, it is not meaningful or useful to empirically compare their approach with ours.

In summary, while there has been much work done on analysing and summarising networks via grouping vertices or finding roles and positions from features, none of the existing work can generate generalised blockmodels that summarises the grouping and how the positions relate to each other, particularly useful for confirmatory type of analysis. In the next section, we will introduce the generalised blockmodelling problem formally.

3 Background

In this section, we formally define the generalised blockmodelling problem. We also introduce the notation we will use throughout the paper and the block types we use in this article. For a summary of symbols used, please refer to Table 1.

Symbol	Description
$G(V, E)$	A graph with vertex set V , edge set E .
$\mathbf{A}$	Adjacency matrix.
v_i	A vertex.
$\mathcal{P}$	Set of vertex positions.
P_i	A vertex set or position.
$\mathbf{B}$	A block type matrix, also known as the image matrix.
$\mathbf{B}(P_i, P_j)$	The type label of the block between positions P_i and P_j . Where it is obvious, we will also use $B_{i,j}$.
$\mathbf{A}(P_i, P_j)$	A block, which is the submatrix of $\mathbf{A}$ with rows from P_i and columns from P_j .
$\mathcal{T}$	The set of possible block types.
$\mathbf{I}(t)$	The matrix structure for an ideal block of type t .
$\mathcal{S}(\mathcal{P}, \mathbf{B})$	A generalised blockmodel, with a set of positions $\mathcal{P}$ and block label matrix of $\mathbf{B}$.
$C(\mathcal{S}(\mathcal{P}, \mathbf{B}))$	The objective cost for a generalised blockmodel, where $C(\cdot)$ is the objective cost function.

Table 1: Description of symbols used in this paper.

A graph $G(V, E)$ consists of a set of vertices V and a set of edges E , $E : V \times V$, which connect two vertices if there is a relation between the entities. The edge relation can be represented by an adjacency matrix $\mathbf{A} = \mathbf{A}(G)$ whose rows and columns are indexed by the vertices of G , and $\mathbf{A}_{ij} = 1$ if vertex v_i is connected to vertex v_j , otherwise 0. In this article, we focus on unweighted graphs. It is possible to extend to weighted graphs; see [55] for more details.

A generalised blockmodel consists of a partition of the vertices into positions and an assignment of the type labels for all pairs of positions.

- Let a set of positions be denoted by $\mathcal{P} = \{P_1, P_2, \dots, P_k\}$, $\bigcup_{i=1}^k P_i = V$, $P_i \cap P_j = \emptyset$, $1 \leq i, j \leq k$. For example, P_1 , P_2 and P_3 in Figure 2b.

- Let $\mathcal{T}$ denote the set of possible block types (see next subsection for a description of the block types used in this paper).
- We denote the block type between two positions by $\mathbf{B}(P_i, P_j)$. Let the image matrix $\mathbf{B}$ denote the set of all pairwise block assignments, i.e., $\mathbf{B} : P_i \times P_j \rightarrow \mathcal{T}$. It can be presented as a $k \times k$ matrix, where $\mathbf{B}_{i,j} = \mathbf{B}(P_i, P_j)$.² The image matrix of Figure 2a illustrates the block types assigned to all pairs of positions in the baboon grooming example.
- A block $\mathbf{A}(P_i, P_j)$ is a submatrix of the adjacency matrix $\mathbf{A}$, and it is defined in terms of vertices belonging to positions P_i and P_j and consists of all edges from vertices in position P_i to vertices in position P_j . A block $\mathbf{A}(P_i, P_j)$ shows the interaction/edges from the vertices in position P_i to the actors in position P_j . An example block is $\mathbf{A}(P_1, P_1)$, which is the upper left block in Figure 1b.

Definition 1 A generalised blockmodel is defined as $\mathcal{S}(\mathcal{P}, \mathbf{B})$, with partitioning defined by $\mathcal{P}$ and block type assignments defined by $\mathbf{B}$.

Each block type has an *ideal* structure. For example, the ideal structure of a complete block type is full connectivity among the vertices of the two positions, while for a null type it is no connectivity. If appropriate error functions can be defined between a block and the ideal structure of its assigned type, then a measure of how well a generalised blockmodel fits the graph is to sum up the total error across its blocks. This is the approach taken by [16] and in this work.

- Let $d(\mathbf{A}(P_i, P_j), \mathbf{I}(t))$ denote the error or distance between blocks $\mathbf{A}(P_i, P_j)$ and the ideal submatrix structure of block type $t \in \mathcal{T}$.

² It is not always the case that index i is mapped to P_i , but for simplicity, we assume it is for the rest of the article.

Definition 2 For $t \in \mathcal{T}$, then global error or objective cost of blockmodel $\mathcal{S}(\mathcal{P}, \mathbf{B})$ is defined as:

$$C(\mathcal{S}(\mathcal{P}, \mathbf{B})) = \sum_{P_i, P_j \in \mathcal{P}} d(\mathbf{A}(P_i, P_j), \mathbf{I}(t)) \quad (1)$$

In this paper, we present 10 different block types, illustrated in Figures 3, 4 and 5. Nine of them are first defined in [16]. We introduce a new type, the functional block type. In Section 3.1, we describe these block types in more detail.

Definition 3 Given a graph $G(V, E)$, the generalised blockmodelling fitting problem is to find a set of positions $\mathcal{P}$ and a block type assignment $\mathbf{B}$ that minimises the objective error cost $C(\mathcal{S}(\mathcal{P}, \mathbf{B}))$:

$$\mathcal{S}(\mathcal{P}, \mathbf{B}) = \operatorname{argmin}_{\mathcal{P}, \mathbf{B}} C(\mathcal{S}(\mathcal{P}, \mathbf{B})) \quad (2)$$

Note that the problem involves partitioning the vertices as well as assigning block types to each block (position-to-position patterns). In [19], Faile and Paulusma proved that assigning vertices to roles using the role equivalence definitions of Section 2 is NP-Complete for three or more roles. As the generalised blockmodelling problem is one of role assignment (in addition to optimising the block types), we argue that it is NP-Complete too. Therefore, we seek and propose two heuristics to fit generalised blockmodels.

3.1 Block Types

Each block type defines a relationship pattern between the interactions of two partitions.

Let $P_i = \{v_{i1}, v_{i2}, \dots, v_{in}\}$ and $P_j = \{v_{j1}, v_{j2}, \dots, v_{jm}\}$ represent the two partitions. For each block type, there are two types of relationships: diagonal blocks ($P_i = P_j$) and non-diagonal blocks ($P_i \neq P_j$). Diagonal and non-diagonal blocks are treated differently because

in traditional social network analysis, self loops are considered to have no social meaning hence they were not modelled. This means that for diagonal blocks, the diagonal of that block itself is always 0, which will impact on several of the block types. In this article we treat the two types of blocks the same (i.e., we consider self-loops to have meaning). However, for completeness, we will explain the block types using the traditional social network analysis dichotomy. Throughout the rest of this section, we will refer to Figures 4 and 5. In both figures, 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.

We first introduce the complete, null and regular types which can model structural and regular equivalence. A complete block (first column of Figure 4) has an edge between all pairs of vertices in the two partitions, and represent complete connectivity between the two sets of vertices. A null block (second column of Figure 4) has no edges between the partitions, and represent absence of connectivity. Finally, a regular block (third column of Figure 4) satisfies 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 4 only shows one example.

Doreian et al. showed that it is sufficient to represent structural equivalence with only the block types of null and complete (see [4] for the proof). For two vertices to be structural equivalent and placed in the same position, they must:

- a) Connect to the same set of vertices (including themselves).

- b) Not be connected to the remaining set of vertices.

So for all vertices in a position, they either all connect to another set of vertices (a complete block), or they all do not link (a null block).

To find regular equivalence in this framework, only the null and regular block types are necessary (see Theorem 6.1 in [16]). For two vertices to be regularly equivalent and placed in the same position, they must be linked to, and by, the same *set* of positions. Translating these to block structures, there are either no links (null block), or there is at least one outgoing link for each source vertex and one incoming link for each target vertex (i.e., a regular block).

This shows that this formulation by Doreian and used in this work can model structural and regular equivalence. To generalise it further, Doreian et al. added 7 additional types.

Row-regular and column-regular block types are illustrated in the 4th and 5th columns of Figure 4. They are relaxation of the regular block type. Row-regular only requires condition a) of regularity, and column-regular only condition b).

Lemma 1 *Let $\mathcal{P}$ be a partitioning of G that conforms to the (a) ((b)) definition of regular equivalence. Then each block in this blockmodel is either of the null type or row (column) regular type. Conversely, if all the blocks are either null or row (column) regular for a partitioning $\mathcal{P}$, then (a) ((b)) holds.*

Proof Follows from Theorem 6.1 in [16].

Lemma 1 indicates that row- and column-regular block types can be used to model a weaker type of regular equivalence. So returning to the professor and student example in Section 2, if some of the original set of students take leave for a semester, than during that

semester, those students will not be taught by any professors, it follows that their interaction would be row-regular. Conversely, if some of the professors went on sabbatical, then they won't be teaching any students and their interaction is thus column-regular.

The functional, row-functional and column-functional types are similar to their regular counterparts (3rd, 4th and 5th columns of Figure 5). The difference is that the functional types restrict the cardinality of the relationships to 1, so it can be thought of as a more strict form of regularity. For the functional block type, it is a regular type but with a one-to-one correspondence between the two sets. Row-functional requires that each vertex of the source vertices (P_i) have one, and only one, outgoing edge. For column-functional, the requirement is for each vertex of the destination vertices (P_j) to have one and only one incoming edge. Functional block types are useful for modelling relationships where there is a one-to-one correspondence or matching/pairing. For example, suppose we are interested in the sexual relations between students and whether those dating are promiscuous or not. Then we can construct a blockmodel with two positions - loyal and promiscuous. Assuming the loyal couples are sexually active, then we expect the loyal couple to loyal couple block to fit the functional block type (one-to-one relationships), and the blocks to do with loyal couples and other students to (hopefully) fit the null block types well.

Finally, row-dominant types require at least one source vertex to have an edge to all destination vertices, and conversely, the column-dominant types require at least one destination vertex to have an edge from all the source vertices (see 1st and 2nd columns of Figure 5). These block types represent star structures, or the neighbourhood subgraph around source and sink vertices.

Fig. 3: Relationships between the 10 block types used in this article. A directed arrow exists between two block types if a block satisfying the conditions for the source type will also satisfy the destination type. E.g., a directed arrow goes from the complete to regular block type because a complete block type is also a regular block type. For the density block type, we use double headed arrow to indicate it is an intermediary type between the complete and null types.

For example, in the professor student example, a row-dominant block could represent professors who teaches every single student, while a column-dominant block could correspond to students that are taught by all professors.

Doreien et al. [16] introduced the density block type which isn't based on the cardinality of the interactions. It is basically a block with an ideal parameterised density (e.g., 60% dense block) that is meant to provide an intermediate block type between the complete and null block types (see Figure 3).

In Figure 3, we illustrate the relationships between the block types. Each rectangle vertex represents a block type, and a directed edge exist between a source type if the source type satisfies the definition of the target type; e.g., the complete type satisfies the requirements of a regular type hence it is also a regular type).

As we have discussed in this subsection, all the block types represents different types of interactions. In Table 2, we have listed the types of blocks needed to fit the vertices to certain types of equivalences. Recall that each of these block types have an ideal structure and an

Equivalence Type	Block types Required
Structural	Null, Complete
Regular	Null, Regular (Complete, Functional)
Row Regular	Null, Row-regular (Row-functional, Col.-dominant, Regular, Functional)
Col. Regular	Null, Col.-regular (Col.-functional, Row-dominant, Regular, Functional)

Table 2: Different equivalence types and the block types required to model them. The types in parentheses indicate that we can substitute the preceding types with it (e.g., in regular equivalence, the complete block type is also regular, hence we can use complete types to replace regular types.

associated error function. These error functions are necessary but not essential to understand the generalised blockmodelling framework; the error functions for each block type are detailed in Appendix A.

The block types cover all possible cardinality of relationships between two sets of vertices - one-to-one, one-to-many, many-to-one and many-to-many. As such, it allows many relationships to be composed from these basic types. In addition, additional block types can be added to the framework, as long as an adequate error function can be defined.

When choosing the possible block types to use, the default is to use all block types and let the algorithm decide. If the user has some domain knowledge of which specific block types are of interest, then they can easily restrict the block types to use. In addition, Table 2 provides a guide to which block types to use to discover blockmodels that adhere to traditional notions of vertex equivalences (e.g., regular equivalence).

4 Generalised Blockmodelling Algorithms

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 Kernighan-Lin graph partitioning algorithm [33]. This KL-based method is the only approach, to the best of our knowledge, that has been proposed to fit generalised blockmodels [16] and we use it as the baseline algorithm to compare against. In this article, we propose two new approaches, one based on simulated annealing (SA) and the other based on genetic algorithms. Both simulated annealing and genetic algorithms are known to find good solutions to very hard combinatorial problems [24], like the blockmodel fitting problem. In the following, we will first outline the baseline KL-based algorithm, then describe the proposed simulated annealing-based and genetic algorithms approaches in more detail.

4.1 KL-Based Algorithm

In [16], Doreian et al. proposed the greedy KL-based approach to fit blockmodels, outlined in Algorithm 1. The algorithm considers the solution neighbourhood of each vertex, and greedily makes a neighbourhood move that reduces the objective cost the most. Doreian et al. considered a neighbourhood move as a) a vertex moving from one position to another (lines 11 to 14 of Algorithm 1), and b) the swapping of two vertices in different positions (lines 15 to 18 of Algorithm 1) as possible neighbourhood moves. The authors of [16] did not describe how to optimise the blocks types themselves. Therefore we also introduce an additional step, where the blocks types are optimised after the positions are optimised (lines 26 to 33 of Algorithm 1).

Fig. 4: First five block types (complete, null, regular, row-regular and column-regular) 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 rows are the graph representation for diagonal and non-diagonal blocks respectively.

The problem with this greedy approach is that the algorithm often gets stuck in local minima, and like most complex combinatorial problems, the evaluation of the objective function is relatively costly. Analysing the algorithm, it can be seen there are many evaluations per run. And to get decent results and allow the initial randomised solution to search the search space adequately, the algorithm must be ran many times – in fact, Doreian suggested to run this algorithm 50,000 times when fitting a blockmodel to a 18-by-14 vertices network, which represents the participation of females at different events [16]. Because of these problems, we have designed two algorithms that reduces the search

space explored while trying to still sample the search space adequately.

4.2 Simulated Annealing

A successful approach to improving greedy searches exiting local minima is simulated annealing [34][52]. Simulated annealing is inspired by statistical mechanics and the annealing process of materials. The main idea is that sometimes, in order to exit local minima, the search process should take a step that makes the costs worse. The probability of taking a worse move is governed by a temperature parameter. Initially, the temperature is high, allowing more worse moves, hence permitting a wider search than a pure greedy approach.

Fig. 5: Second set of block types (row-dominant, column-dominant, functional, row-functional and column-functional) and their ideal adjacency and block patterns. First row is the adjacency matrix, second and third rows are the graph representation for diagonal and non-diagonal blocks respectively.

But the temperature is brought down quickly, and as the temperature is brought down, more and more optimisation occurs around the (local) minimum found at higher temperatures. We have adopted the simulated annealing approach for fitting blockmodels.

Algorithm 2 outlines the SA approach (named *saBM*). We used the standard SA approach [31], but with problem specific differences in the generation of an initial random solution, the objective cost function, and the generation of a neighbourhood. In our case, a neighbourhood operation is either a swap, move or block type change. There are five parameters associated with the algorithm. *maxSteadyGen* is the maximum number of iterations where no change has been made to the best

solution. *maxTrial* is the maximum number of trials at a temperature level and *maxChanges* is the maximum number of changes to the best solution at a temperature level. *maxTrial* must be greater than or equal to *maxChanges*. *reduceFact* is the temperature cooling factor. *initialT* is the initial temperature. The description of these parameters and their default values are available in Table 3. We obtained the default parameter values listed in Table 3 by running the algorithm 10 times for each parameter setting on synthetic data (see Section 5) and recording the set that gives the minimum mean objective cost.

Algorithm 1 Outline of one run of the KL-based Algorithm

```

1: Input: A graph -  $G(V, E)$ 
2: Output: A blockmodel -  $\mathcal{S}(\mathcal{P}, \mathbf{B})$ 
3: // Initialise with a random blockmodel solution
4:  $\mathcal{S}(\mathcal{P}, \mathbf{B}) = \text{randomSolution}()$ 
5: // Priority queue of all cost improving operations
6:  $\text{solnQueue} = []$ 
7:  $\text{bImprove} = \text{true}$ 
8: while  $\text{bImprove}$  do
9:    $\text{bImprove} = \text{false}$ 
10:  for each vertex in  $V$  do
11:    Try a move to all other positions
12:    if improvement in cost then
13:      add to  $\text{solnQueue}$ 
14:    end if
15:    Try a swap with all other positions
16:    if improvement in cost then
17:      add to  $\text{solnQueue}$ 
18:    end if
19:  end for
20:  // Apply the move and swap operators
21:  if  $\text{solnQueue}$  is not empty then
22:    apply the best solution
23:    update best cost
24:     $\text{bImprove} = \text{true}$ 
25:  end if
26:  // Now optimise blocks
27:  for each element  $\in \mathbf{B}$  do
28:    try other block types
29:    if cost decrease then
30:      keep new solution
31:       $\text{bImprove} = \text{true}$ 
32:    end if
33:  end for
34: end while

```

4.3 Genetic Algorithm

The inherent difficulty with fitting a generalised block-model is the need to solve the partitioning and block type assignment problems simultaneously. Both problems are inter-dependent, hence there are many local

Algorithm 2 Outline of one run of the SA-based Algorithm (*saBM*)

```

1: Input: A graph -  $G(V, E)$ , Initial temperature  $T$ 
2: Output: A blockmodel -  $\mathcal{S}(\mathcal{P}, \mathbf{B})$ 
3: // Initialise with a random blockmodel solution (current best solution)
4:  $\mathcal{S}(\mathcal{P}, \mathbf{B}) = \text{randomSolution}()$ 
5:  $\text{steady} = 0$ 
6: while  $\text{steady} < \text{maxSteadyGen}$  do
7:    $\text{trials} = 0, \text{changes} = 0$ 
8:   while  $\text{trials} < \text{maxTrial}$  and  $\text{changes} < \text{maxChanges}$  do
9:     // Generate a neighbouring solution
10:     $\mathcal{S}'(\mathcal{P}', \mathbf{B}')$ 
11:     $\Delta = C(\mathcal{S}(\mathcal{P}, \mathbf{B})) - C(\mathcal{S}(\mathcal{P}', \mathbf{B}'))$ 
12:    // Evaluate if neighbouring solution is better
13:    if  $\Delta < 0$  then
14:       $\text{changes} += 1$ 
15:      update best solution
16:    else
17:       $r = \text{rand}(0, 1)$ 
18:      // Test if we accept a worst solution
19:      if  $r \leq e^{\frac{-\Delta}{T}}$  then
20:         $\text{changes} += 1$ 
21:        update best solution
22:      end if
23:    end if
24:     $\text{trials} += 1$ 
25:  end while
26:  // Lower temperature
27:   $T = \text{reduceFact} * T$ 
28:  // Check if change has occurred
29:  if  $\text{changes} > 0$  then
30:     $\text{steady} = 0$ 
31:  else
32:     $\text{steady} += 1$ 
33:  end if
34: end while

```

Parameter	Description	Default
maxSteadyGen	Maximum no. of iterations with no improvement in objective.	10
maxTrial	Maximum no. of trials at a temperature level.	10
maxChange	Maximum number of changes at a temperature level.	10
reduceFact	Temperature cooling factor.	0.5
initial T	Initial temperature.	10.0

Table 3: Description of the parameters of *saBM* and their default values.

minima. This is the reason why greedy approaches like the KL algorithm generally do not perform well.

Genetic algorithms are known to be able to tackle hard combinatorial optimisation problems [12][28]. One of its strengths is that it is able to search multiple points of the solution space in parallel while still converging on good solutions.

There are some existing genetic algorithms work for partitioning problems [8], but as far as we know, the closest work to tackling generalised blockmodelling using genetic algorithms is [30]. Although they used a similar representation to this work, our representation additionally encodes the block types; this makes the problem much more difficult. Also in [30], 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 and [30] are related, but as explained earlier in Section 2, [30] solves a restricted version of community finding and cannot use or assign block label types.

The genetic algorithms consist of five standard parts: selection, recombination, mutation, transform and stopping criteria. A genetic algorithm (GA) consists of a number of generations (each loop in Algorithm 3). Given a population, the set of current solutions, a number of them are selected for reproduction and mutation. Typically, solutions with low objective costs are more likely to be selected, allowing convergence to a good solution. Reproduction, in the form of recombination, allows two or more solutions to swap sub-parts between themselves. The idea is that good solutions are made from a number of good sub-parts, and by swapping sub-parts, some of the offspring of the good solutions will improve further when it gets the good sub-parts from its parents. Mutation alters a part of the solution (e.g., move a vertex to another position, change the block label assignment). This is to allow the GA to be able to search unknown parts of the search space and to exit local minima. Finally, the stopping criteria determines when to stop the GA cycle, and output the final solution.

Our genetic algorithm, named *gaBM*, consists of the five standard steps (see Algorithm 3 for an outline). Initially, a population of *popSize* number of blockmodels are randomly generated (line 3 of Algorithm 3). Then *probRecomb* % 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 gen-

erations, or we have iterated over $maxGen$ number of generations.

Because of the dual partitioning and block type assignment problems, we have designed a non-standard representation, recombination and mutation operators. The list of parameters used in the fitting of the datasets are listed in Table 4. We describe the representation, recombination and mutation operators in the following.

Algorithm 3 Outline of the *gaBM*.

```

1: Input: A graph, parameters of Table 4.
2: Output: A population of blockmodels.
3: Initialisation: 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 se-
     lection
7:   Recombination: See section 4.3.2:Recombination Oper-
     ator
8:   Mutation: See section 4.3.3:Mutation Operators
9:   Replacement: plus strategy + elitism (keep best solu-
     tion)
10:   $gen++$ 
11:   $steady = 0$  if improvement in objective of best solution
     else  $steady++$ 
12: end while

```

4.3.1 Representation

Solutions in genetic algorithms are called chromosomes, and are typically represented as vectors. Each element of the vector is called a gene. A popular representation for partitioning problems is for each gene (i.e., each element of the vector) to represent a vertex, and assigned with the id of the position it belongs in. For example, consider Figure 6a, and an example chromosome representation of a two position solution to the baboon grooming example (Figure 1). The chromosome repre-

Parameter	Description	Default
popSize	Population size.	200
maxGen	Maximum no. of generations.	1000
minGen	Minimum no. of generations.	10
maxSteadyGen	Maximum no. of generations with no improvement in the objective.	10
CrossoverPts	No. of recombination points.	2
probRecomb	Probability of recombination.	0.6
probMut	Probability of mutation	0.1
probMove	Probability of a move.	0.25
probSwap	Probability of a swap.	0.25
probType	Probability of a type assignment.	0.75

Table 4: Parameter settings of *gaBM*, used in the fitting of the datasets.

sents the position assignment to each of the 12 vertices, where the values at each gene indicates which position the vertices belong in. For example, vertices of genes ‘a’ and ‘c’ belong to the same position 1.

Despite its popularity [17][40], this integer representation has two major flaws. It is highly redundant, for example, if we represent the example chromosome in Figure 6a as the string 121211222222, then the two chromosomes 112222222222 and 221111111111 encode the same solution, but will be treated as two different solutions in this representation. This makes the search space larger, making optimisation harder. In addition, it can be highly disruptive, e.g., one of the offspring of recombining 11111|2|111111 with 21111|1|111111 is 211112111111, where ‘|’ are the boundaries of where we replace all values between the two ‘|’ of one chromosome to the other. The original fact that the position 2 should be a singleton is lost. The problem here is that the rep-

1	2	1	2	1	1	2	2	2	2	2	2	2
a	b	c	d	e	f	g	h	i	j	k	l	

(a) Partition id chromosome representation.

1	2	1	2	1	1	2	2	2	2	2	2	2	1
a	b	c	d	e	f	g	h	i	j	k	l		

(b) Grouping genetic algorithm chromosome representation. Partition part specifically encodes the position information.

(c) *gaBM* representation, with block type matrix.

Fig. 6: Illustration of the chromosome representation.

representation should encode which vertices belong to the same position, not use its id as an indirect method to do so.

In the literature, another representation called the grouping genetic algorithm (GGA) has been proposed [18][40]. Rather than focusing on the elements, it focuses on the position - the objective cost is dependent on the partitioning, so the representation should explicitly represent the position information. The GGA representation does this by augmenting the integer representation with position information. Consider the chromosome for the baboon network augmented with position information, illustrated in Figure 6b. The two positions are represented as two extra genes, after the

vertical line. Although it seems like the same flaws are present in GGA, the important point is that the recombination operators only operate on the position part. The vertex assignment part only states which position each vertex belongs in. Combined with the operators we have designed, this avoids the redundancy and isomorphism problems of the standard representation.

In addition to the partitioning information, our representation also has to encode the block type assignments. For k -partitions, the assignments are represented by the $k \times k$ block type matrix. The genes in the matrix are the individual matrix entries, and the genes can take any specified block type label. To map between the positions and the block type matrix, an internal two way dictionary maps between the position id and the rows and columns representing its type in the matrix. The whole chromosome representation of *gaBM* is illustrated in Figure 6c. In Figure 6c, the arrows from the two positions point to the corresponding row (and column) they reference in the image matrix, e.g., regular is the block type assigned to the relation between P_2 to P_1 . Because there are two positions, then there is a 2×2 image matrix.

4.3.2 Recombination Operator

The recombination operator is important in genetic algorithms, as it is one mechanism to exchange potentially good parts of good solutions. There is a recombination operator defined for the GGA representation; however, it is designed for the partitioning problem only, hence we extend the operator to include the block type matrix. Given two selected parent chromosomes for recombination, the main idea is to choose a subset of positions to inject from one chromosome into another. After injection, there might be inconsistencies

(same position number, elements belong to two different partitions, etc). In our recombination operator, we repair these inconsistencies.

We first describe how the partitioning information is exchanged, then how the block type assignments are exchanged and optimised. To help explain the recombination steps, we use the baboon example again; the various steps are illustrated in Figure 7 and we shall refer to this as we describe the recombination operation step by step.

1. Pick two recombination points in each of the two parent chromosomes S_1 and S_2 . Let the recombination points be labelled X_{11} and X_{12} for S_1 (X_{21} and X_{22} for S_2). This is illustrated in Figure 7a, where position 1 is inserted into S_2 .
2. Let the positions between X_{11} and X_{12} of S_1 be denoted by I_1 . Inject the positions of I_1 into recombination point into X_{21} of S_2 .
3. If a vertex v_i in position P_k of S_2 also exists in any position P_l of I_1 , delete position P_k . In the example (see Figure 7b), position 4 of S_2 has elements ‘e’ and ‘f’ overlapping with position 1 of S_1 .
4. For all vertices in S_2 that are not assigned to any position (due to position deletions), evaluate each position in S_2 and insert into the position that results in the best objective cost. For example, elements ‘g’ and ‘h’ are redistributed to the best positions 6 and 1 respectively in Figure 7c.
5. Repeat steps 2 to 4 for injecting the positions of X_{21} and X_{22} into S_1 .

In the case of pre-specification of the image matrix (hence the number of positions is pre-specified and kept constant over the generations), we need to make a modification, as the current algorithm might not maintain the number of starting positions. For every position in-

serted, we only replace one existing position. If there are multiple positions that would have been deleted in the scheme, we choose the positions that have the least disruption - i.e., the positions that have the least number of vertices. If there are vertices that are assigned to two positions after injection, we assign it to the position that results in the best objective cost. By maintaining these conditions, we guarantee that the number of positions is constant over different generations.

The schemata principle of genetic algorithms [2] 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 positions should be maintained. Because of this, we argue that the assignment information should not be independently recombined. However, we only have good block type assignments among the injected positions themselves, and lack information of the best block types between the injected positions and the existing positions. For these latter assignments, we try each possible block type and choose the best type assignment. Therefore, we use the following procedure when transferring the type assignments with the injected positions.

1. Let *Indices* denote the set of $\mathbf{B}$ indices assorted with the injected positions of I_1 . For Figure 7a, this is 1, since position 1 has index 1 into the $\mathbf{B}$ of chromosome S_1 .
2. For all $i \in \text{Indices}$, copy $\mathbf{A}(i, i)$ to the $\mathbf{B}$ of S_2 . In the 2nd step of the example, Figure 7b copies $\mathbf{A}(1, 1)$ from S_1 to $\mathbf{A}(3, 3)$ of S_2 (position 1 is mapped to index 3 of $\mathbf{B}$ of S_2). Note that at times, step 4 in the recombination of the vertex assignments can result in changes to the membership of an injected position, e.g., vertex ‘h’ is added to the

Fig. 7: Crossover example. Partition 1 is inserted into 2nd chromosome. It causes position 4 to be deleted, and vertices at positions 'g' and 'h' of chromosome S_2 are redistributed.

injected position in Figure 7c. This can affect the best label assignment, hence if there are any changes to the membership of the injected position, we also do a search on the best block assignment, e.g., performing this search on $\mathbf{A}(\mathcal{I}, \mathcal{I})$, where the existing label is optimal in this example.

3. For all $i, j \in \text{Indices}$, $i \neq j$, copy $\mathbf{A}(i, j)$ and $\mathbf{A}(j, i)$ to the $\mathbf{B}$ of S_2 . As the *Indices* in the example has only one element, nothing is done in this step. Note that in the example, all the entries of indices 3 of $\mathbf{B}$ of S_2 , apart from $\mathbf{A}(\mathcal{I}, \mathcal{I})$, needs to be reassigned and patterned with diagonals in Figure 7b.
4. For all $i \in \text{Indices}$, all $h \notin \text{Indices}$, find best block type assignment for that position. In the example, all the entries of the image matrix are assigned to their best type.

At the end of this recombination operator, we get two new offspring that share parts of the parents.

In summary, in this subsection, we have described the recombination operator, which consists of swapping the position information and then updating the block type matrix.

4.3.3 Mutation Operators

Mutation in genetic algorithms is used to escape from local minima and maintain diversity in the population. We have designed three different operators to perturb the existing chromosomes: swap two random vertices between two randomly chosen positions, move a random vertex from a randomly chosen position to another, change block type assignment to another allowable type. These occur with *probSwap*, *probMove* and *probAssign* percentage of the time, respectively. Generally, mutation is not as important as recombination. However, because the recombination operator does not

recombine block type assignments directly, the assignment mutation operator is important to allow searching over assignment. Hence, we set the default assignment mutation operator to occur on average three times more often than the other two mutation operators (see Table 4).

4.4 Incorporating Additional Constraints

As highlighted in the introduction, one of the strengths of the generalised blockmodelling approach is the ability to perform confirmatory and exploratory types of analysis via pre-specifying block types or imposing constraints on the allowable blockmodels. In this section, we will describe the constraints extensions included into the three introduced algorithms.

In the introduction, we have already described two constraints that our approach allows: (partial) setting of the membership in blocks and setting of what block types are allowed for the blockmodel and for individual blocks. The formulation allows addition constraints to be added to the system as needed. Currently, we have the following additional constraints:

- Setting the range of the number of positions
- Minimal and maximal block size.

Often, users have an approximate idea of how many blocks they want to fit a network to. This is easily incorporated into the *gaBM* algorithm during the recombination process. As positions are recombined, occasionally multiple positions are deleted in the chromosome that is being inserted to, and occasionally less positions are deleted than inserted. If there isn't enough or too many positions, the algorithm can split the position that has the largest error or merge two positions that have the smallest error.

To maintain minimal (maximal) block sizes, the *gaBM* algorithm repairs infeasible solutions. After recombination, and mutation, some solutions might become invalid. To repair these, elements from blocks that are above (below) the block size thresholds are selected with uniform distribution and moved to blocks that are infeasible. This is a balance between efficiency (searching for best moves) and favouring balance blocks (if the block with greatest error is chosen, these are usually the largest blocks).

In this section, we have describe the algorithm of *gaBM*. It uses a GGA based representation with additional block type matrices, a recombination operator that swaps positioning information and block assignments, and mutation operators that introduces some randomness to help escape potential local minima.

5 Evaluation

In this section, we evaluate the speed and optimisation of the three algorithms and demonstrate that *gaBM* can find insightful generalised blockmodels for real datasets.

To evaluate the performance of our proposed algorithms, we used two types of datasets. First, we used synthetic datasets to evaluate the scalability and optimisation. Second, we fitted generalised blockmodels to three real-world dataset to demonstrate the usefulness of having block types, allowing pre-specification and the utility of having increased scalability. The three real datasets are the Enron email dataset, a communication network at a doctoral summer school, and an airport routing network.

In the following subsections, we first explain how we generated the synthetic datasets, then present the results from these datasets. Next, we present the blockmodels decomposing the three real datasets.

5.1 Generated Community-Blockmodel Data

To evaluate the scalability and optimising ability of the three algorithms, we generated synthetic datasets by following the community generating algorithm given by Lancichinetti et al. [35]. The advantage of using a community generator is that it generates networks with a known blockmodel to compare against. Also, it enables us to test the performance of the algorithms when the model is specified (community type blockmodels; i.e., complete block types on the diagonal, null types elsewhere) and also when the model is not specified.

Briefly, the algorithm of Lancichinetti et al. generates an unspecified number of communities that have a set number of vertices, $|V|$. These vertices have an average degree, but the degrees of the vertices can vary and are distributed according to a power-law. In addition, the community sizes also follow a power-law distribution, and the amount of inter-community connections was governed by a mixing parameter, μ , with lower μ indicating more inter-community connections. We set the exponents of the degree and component size distributions to 2 and 1 respectively, which many networks follow [14], and also set the average degree to 5% and 10% of the number of vertices.

We evaluate the performance of the *KL-based*, *gaBM* and *saBM* algorithms on the generated synthetic community datasets, with and without the blockmodel supplied to the algorithms. For each configuration of the synthetic datasets, we generate 10 different community-model networks, ran the algorithms and analysed the total objective cost (lower is better) of the fitted blockmodel and the total running time (lower is better). All the experiments were conducted on a dual Xeon 2.27GHz server with 32GB of memory and running Ubuntu 10.04.

We first present the results of when the blockmodels are not prespecified, illustrated in Figure 8. First, consider the effecting of varying the mixing parameter μ . μ gives an indication of how nodes link with other nodes of their community and to the rest of the network. As μ increases in value the generated network contains communities that are less uniform, and consequently harder to find. In the presented set of experiments, the network size is 50 vertices, but we found the same trends for all network sizes.

Figure 8d shows that the running time for *gaBM* and *saBM* are at least one to two factors faster than the *KL-based* algorithm, and for all three algorithms, changing the mixing parameter μ does not have a significant effect on the running times. Now consider Figure 8a, which shows the effect of μ on the objective costs. The results show that the objective cost of the *KL-based* algorithm is slightly lower than *gaBM*, while both these algorithms obtained much better fitting blockmodels than the *saBM* algorithm. Furthermore, note that for $\mu = 0.2$ and 0.3 , the error bars (one standard deviation width) are very wide for *gaBM*. Our reasoning is that there was one result for each mixing setting that were much bigger than the other results. Recall *gaBM* is a stochastic algorithm, so sometimes it will get stuck in a local minimum that produces large objective cost blockmodels. In addition, Figure 8a illustrates the objective cost of *KL-based* is slightly lower than *gaBM*, and both *KL-based* and *gaBM* obtained much better fitted blockmodels than the *saBM* algorithm. The results suggest that all three algorithms are insensitive to changes in the mixing parameter μ , with *gaBM* providing a good balance between running times and good blockmodels.

Next, we compare the objective cost and running times of the three algorithms when the generated network sizes change. We present two types of results. For all three algorithms, we only show results for two network sizes, 50 and 100 (Figures 8b and 8e), because the *KL-based* algorithm can only realistically fit models to networks of 100 vertices. For the *gaBM* and *saBM* algorithms, we also show results for network sizes up to 800 vertices (see Figures 8c and 8f).

First, consider the performance comparison of the three algorithms (Figures 8b and 8e). Figure 8b show that the *gaBM* algorithm discovers similar quality solutions to the *KL-based* algorithm, with the *saBM* results having three times the objective cost. Speed-wise, Figure 8b show that *gaBM* is slightly less than a factor 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.

Finally, for larger networks, the results illustrated in Figures 8c and 8f 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, then 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 community blockmodel is used to initialise the algorithms. The mixing and small network size results (Figures 9a, 9d, 9b and 9e) have similar trends as the results of having no blockmodel prespecified. However, for the larger network sizes results (Figures 9c and 9f), *gaBM* benefits greatly from the prespecified blockmodel, with objec-

(a) Objective cost vs. mixing. Network size is 50 vertices.

(b) Objective cost vs. number of vertices, for all three algorithms and network sizes up to 100 vertices.

(c) Objective cost vs. number of vertices, for *gaBM* and *saBM* and network sizes up to 800 vertices.

(d) Running time vs. mixing. Network size is 50 vertices.

(e) Running time vs. number of vertices, for all three algorithms and network sizes up to 100 vertices.

(f) Running time vs. number of vertices, for *gaBM* and *saBM* and network sizes up to 800 vertices.

Fig. 8: Results for the synthetic datasets when no blockmodels are prespecified.

tive costs significantly improved over *saBM* and running times only 3-4 times slower. This indicates that for larger networks, having a prespecified blockmodel can assist *gaBM* to better results. In the rest of the experiments, we use *gaBM* to analyse all the networks.

5.2 Enron Email Network

Using *gaBM*, we analyse the email communications among Enron employees at three different time periods and present three fitted blockmodels, one for each period. We then show and discuss how the blockmodels, with their block type labels, can reveal the important changes inside Enron. We also ran the KL-based algorithm on these communication networks, but all the experiments were still running after 30 days. For a semi-supervised

approach, 30 plus days is too long to wait, hence we terminated the runs and we argue that it is fair to state that KL-based algorithms cannot scale to this Enron dataset.

The Enron email corpus is a set of emails collected from the Enron corporation over 3.5 years. During this collection period, Enron went through a crisis and subsequent bankruptcy. We build blockmodels to analyse how the communication patterns changed as the crisis unfolded. There are several versions of the Enron email database available publicly, but we used the version that contains only email among Enron employees [46]. From the database of emails, we constructed the email communication (directed) graphs; the vertices in the graphs are the employees of Enron and the di-

(a) Objective cost vs. mixing. Network size is 50 vertices.

(b) Objective cost vs. number of vertices, for all three algorithms and network sizes up to 100 vertices.

(c) Objective cost vs. number of vertices, for *gaBM* and *saBM* and network sizes up to 800 vertices.

(d) Running time vs. mixing. Network size is 50 vertices.

(e) Running time vs. number of vertices, for all three algorithms and network sizes up to 100 vertices.

(f) Running time vs. number of vertices, for *gaBM* and *saBM* and network sizes up to 800 vertices.

Fig. 9: Results for the synthetic datasets when blockmodels are prespecified.

rected edges representing one or more emails sent between two employees. We constructed and analysed the emails over three periods of time - prior (31/03/2000–30/11/2000), during (01/12/2000–30/07/2001) and after (01/08/2001–15/12/2001) the crisis, denoted T_1 , T_2 , and T_3 , respectively. The number of vertices (edges) for the graphs of T_1 to T_3 are 270 (1202), 264 (1718) and 284 (1876) respectively.

To examine how well *gaBM* performs confirmatory analyses, we use prespecified image matrices to construct the blockmodels. In [15], Diesner et al. analysed the amount of communication between different groups of Enron employees across time. The groupings were based on the extracted job titles. They identified seven job groupings (Executive Management, Senior

Management, (middle) Management, Lawyers, Traders, and Specialist and Associates). We found that the email patterns of Managers and Traders, and Specialists and Associates were similar, therefore to overcome the possibility of over-fitting, we merge each pair of job groups together. Hence, using interaction information from [15] and the five job groupings (Executive Management, Senior Management, Management/Trader, Lawyers and Specialist/Associates), we construct a prespecified image matrices for each time period and found the associated blockmodels using *gaBM*.

We used the default parameter settings for *gaBM*, apart from varying the population parameter between 80, 100 and 120 and setting our generation to halt when

it reached a steady state of 20 (i.e., when the objective function showed no improvement after 20 iterations).

We illustrate the three blockmodels we obtained for each of the time periods in Figure 10. We also show the average and standard deviation of the running time taken for each population size in Table 5. As it can be observed, the running times increases with population sizes. For T_1 and T_2 periods, we found a population of 80 produced meaningful blockmodels, while for T_3 , we found a population of 100 or 120 was preferable.

To rate the accuracy of our findings we computed the Normalised Variation of Information³ (NVI) for comparing clusterings as described by Meila [39]. The ‘true clustering’ we used were taken from the known work titles of each person. We found the optimal blocks for each time step along with their NVI value (lower is more similar) and objective cost. As Figure 10 shows (under each caption), the NVI values for the blockmodels are reasonably low, suggesting they are a close fit to the known roles.

Next, we discuss the block type labels assigned to the blockmodels and what type of information they can reveal. For example, Figure 10 show that the communication pattern of the executive managers (labelled ‘A’ in the figure). The blockmodels indicate there is little variation of the communication patterns among themselves, but the inter-position/role communications reveal a different story. They increase from almost no communication at T_1 (with only little received from senior management and lawyers - this is expected, as in a company under normal circumstances, it is usually the secretaries that do most of the emailing for the executive management) to almost complete communication

to all others at T_2 , especially to lawyers and specialists/associates in both directions. Then communication from T_2 to T_3 change from complete to mostly regular types, reflecting that the executive management reduced their communication and crisis management, possibly because they knew the company was about to declare bankruptcy. Other roles also exhibit this change in behaviour from T_2 to T_3 . During T_2 , there is more intense email communication between all roles (the increase of non-null interaction block types between roles, as well as some (row/column) regular patterns becoming complete patterns). But in T_3 , many of the complete patterns become regular patterns, reinforcing the idea that either the employees knew bankruptcy was coming, or they were instructed to keep conversations from digital channels.

In summary for the Enron analysis, we showed that prespecified image matrices can be used to find generalised blockmodels that summarise the roles, highlight the fundamental changes, and also the key relationships (block types) between the different roles. measures.

5.3 Doctoral Summer School Communications Graph

As part of a mini-project at a doctoral summer school in 2011, participants were asked to indicate who they had communicated before and during the summer school via an online survey. We built a communication graph from the aggregated version of this data⁴, where the vertices represent participants and directed edges representing communication. The graph consists of 73 vertices (from 74 participants, one did not agree to their data being used) and 1139 edges. However, we note that only 40 participants actually responded to the survey, but because others may have communicated with them

³ $NVI(\mathcal{C}, \mathcal{C}') = \frac{H(\mathcal{C}) + H(\mathcal{C}') - 2I(\mathcal{C}, \mathcal{C}')}{\log(n)}$, where n is the total number of elements in a clustering.

⁴ Available at <http://clique.ucd.ie/data>.

80	mean	s.d.	100	mean	s.d.	120	mean	s.d.
T_1	516.14	158.27	T_1	787.04	198.81	T_1	861.59	143.07
T_2	948.21	261.17	T_2	1004.52	182.03	T_2	1242.84	212.01
T_3	579.61	181.35	T_3	1075.41	181.35	T_3	1022.44	189.57

Table 5: Mean and standard deviation of the running time (in seconds) for each time frame with populations 80, 100, 120.

Fig. 10: Image diagrams for each time step along with their NVI value and objective cost (Obj. Cost). The 'A' node represents the Executive Management, 'B' for Senior Management, 'C' for Lawyers, 'D' for Managers/Traders, and 'E' for Specialists/Associates.

they are included. As we do not have a known model to use for prespecifying the image matrix, we use a block-model generated from domain knowledge to construct our image matrices. We will demonstrate a fitted generalised blockmodel that corresponds to the known roles of the participants and show that the block types succinctly summarises the role-to-role patterns.

The data was anonymised but we can briefly describe these expected roles as follows:

- **Core organising committee:** associated with the host organisation and would have communicated to almost everyone before and during the summer school. They would also be actively taking part in the summer school.
- **Local organisers:** associated with host organisation and would have communicated mostly with core organising committee. May or may not have actively taken part in the summer school.
- **Active participants:** may or may not be associated with host organisation but possible to have

communicated (collaborators, research, mutual acquaintances etc.). Actively communicated during the school.

- **Less/non-active participants:** similar to Active participants but less likely to have communicated much during. Likely to not have attended full duration of school (so had less opportunity to socialise). The non-responding (to the survey) participants may also be in this group.

Thus, using these as a guide, we constructed the blockmodel GBM_4 .

We show the best-fitted result in Figures 11 and the average times taken to run, as well as mean objective cost differences, using *gaBM* in Table 6. These times are all in the scale of seconds which is significantly faster than a *KL-based* algorithm which spans hours. We note that for the *KL-based* algorithm we get an optimal objective cost of exactly 701 in all instances which indicates that either the algorithm had found the global, or that there was a very prominent local mini-

Algorithm	Obj. Cost		Time	
	mean	s.d.	mean	s.d.
GA	859.3	114.1	24.2	4.2
KL-based	701	0	27056.3	845.5

Table 6: Mean and standard deviation of objective cost and running time in seconds for Summer School blockmodel GBM_4 with (population).

mum. Either way, compared with the objective cost of $gaBM$, we get a much better trade-off because the running time difference outweighs the objective cost gains.

From Figure 11, we can see that GBM_4 shows a very clear structure which matched quite closely to the four roles we described. Upon manual inspection, the members of the core organising committee were identified most accurately and this is reflected quite clearly in the blockmodel (position ‘A’); they would have communicated with almost everyone at the summer school and this is reciprocated by nearly all members apart from those who didn’t participate in the survey or inactive (position ‘D’), i.e. the largest group. There is also a clear distinction between the other two groups — local organisers (position ‘B’) and active participants (position ‘C’). The local organisers and active participants have very similar patterns of interactions to the organising committee (A) and inactives (D), but differ in that both groups have strong association among themselves. It probably indicates that the local organisers knew each other before the conference, and since not all of them are participants in the school, there is a reduced need to communicate with members of the active participants. This blockmodel shows that we can use $gaBM$ with pre-specified image matrices to decompose the summer school communications network into identifiable positions and also label and summarise their different position-position patterns.

5.4 Airport Routing Graph

The airport routing network represents the flight routes between airports⁵. Each vertex represents an airport, with directed edges representing a flight route. To be able to interpret the results, we concentrated our analysis to one region; we chose Europe⁷. The network consisted of 561 nodes and 10,956 edges, which is too large for the KL-based algorithm to handle (KL-based runs do not finish after one week).

Similar to the other real data analyses, we used several pre-specified image matrices to look for structure in the network; in particular, a core-periphery structure which is well-established in the literature [49]. The type of blockmodels we pre-specified are 3x3 and 4x4 image matrices based on regular equivalence, and 3x3 and 4x4 image matrices based on a block density of 0.5. The 4x4 based on regular equivalence didn’t yield interpretable results (no visible structure from visual inspection of adjacency matrices), hence we do not show its results. In Figure 12 we show the best-fitting image matrices and blockmodels for the remaining three blockmodels. Table 7 shows the average time taken for each blockmodel to be found using $gaBM$; the 4x4 density block-

⁵ The data was taken from OpenFlights.org⁶ in December 2010

⁷ For our purposes, it was decided that geographical location defined what the countries were, specifically: Albania, Armenia, Austria, Azerbaijan, Belarus, Belgium, Bosnia and Herzegovina, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Georgia, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Liechtenstein (not in database), Lithuania, Luxembourg, Macedonia, Malta, Moldova, Monaco, Montenegro, Netherlands, Norway, Poland, Portugal, Romania, Serbia, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey, Ukraine, United Kingdom, Jersey, Guernsey, Isle of Man, and the Faroe Islands; Russia was omitted as its landmass spanned further East than was required.

Fig. 11: Image graph and adjacency matrix showing positions obtained from *gaBM* with pre-specified blockmodel GBM_4 . Nodes A to D represent positions going from top left to right.

Fig. 12: Adjacency matrices and labelling of their positions as 'A' to 'D' for 4x4 density, 3x3 regular and 3x3 density blockmodels.

Blockmodel type	mean	s.d.
3x3 regular (80)	1543	334.33
3x3 density (100)	1436	302.04
4x4 density (100)	4064	528.26

Table 7: Mean and standard deviation of the running time in seconds for each airport route blockmodel with (population).

model took the longest, at just over one hour on average and for the *KL-based* algorithm it didn't finish after 350 hours, hence we do not report this.

For this dataset, there is no ground truth for validation. Therefore, we used knowledge of the top 20 European and World connected airports as ranked according to their connectivity [38] to determine the plausibility of the clusterings. There are 28 airports in total with 12 (Prague Ruzyně (PRG), Frankfurt (FRA), Düsseldorf (DUS), Paris Charles de Gaulle (CDG), Copenhagen (CPH), Amsterdam Schiphol (AMS), FCO, Manchester (MAN), Stockholm Arlanda (ARN), Munich (MUC), Madrid Barajas (MAD), Barcelona El Prat (BCN)) overlapping so these airports we emphasise their appearance

in core clusters. Table 8 shows the distribution of these airports corresponding to the blockmodel shown in Figure 12. It can be seen that the 4x4 density-based image matrix gives the most accurate result with all the majority of the top 20 airports being in the most dense (fourth) cluster and the remainder in the next densest in the core-periphery structure.

The next blockmodel in the table is a 3x3 regular-equivalence (no density) one. Here, the majority of the most well-connected airports are found in the first cluster and for the ‘top of the hub’ position ‘C’, 6 out of its 14 members are amongst the 20 most connected. Barcelona El Prat (BCN) was found in the most sparse cluster where there were 281 members. The final blockmodel we show is a 3x3 density-based one. Although Figure 12c appears to show a promising hub structure, upon closer inspection, Table 8 shows that the most connected airports are much more dispersed among the cluster. This demonstrate that even when using a density specification would result in more accurate blockmodels, a 3x3 regular equivalence based blockmodel is still able to find relevant airport to role assignments and insightful role-to-role (block) labels and provide an alternative decomposition.

6 Conclusion

Generalised blockmodelling extends traditional blockmodelling by allowing block types which allows a discovered blockmodel to explain the role-to-role interactions in terms of one of the existing interaction patterns. It also permits partial or complete pre-specification to these types, allowing confirmatory and exploratory types of analysis to be performed. Because the current algorithm to fitting generalised blockmodels does not scale, in this article, we have presented two optimised algo-

rithms for fitting network data to a generalised blockmodel. The algorithms makes use of the strengths of genetic algorithms and simulated annealing in tackling hard combinatorial optimisation problems. We demonstrated their efficiency and accuracy in comparison to current approaches on several datasets. Using the improved scalability, we have fitted generalised blockmodels to three, real medium sized datasets and showed how the addition of block types and pre-specification can find revealing and insightful information.

For future work, we would like to incorporate more constraints into the algorithms. For example, constraints that states which pairs of vertices must be or must not be in the same role/position [48] can be useful, as practitioners usually have some idea of which vertices should or should not be in the same role a priori, and these type of constraints would allow them to express these biases.

In addition, another interesting area of research would be to extend generalised blockmodelling to dynamic networks [45]. This would involve incorporating time into the objective, either in terms of events like [26] or perhaps as smoothed blockmodels like in [53][11]. A dynamic generalised blockmodelling framework would allow us to detect when roles change and when role-to-role (block types) change, indicating a fundamental change in role interaction.

Finally, a natural extension is to consider soft positions, where a vertex can be members of several positions simultaneously. For example, an assistant professor can be both teaching as a professor (to undergraduates) and learning as a student (from full professors).

7 Acknowledgements

This work is partly funded by the Australian Research Council under grant number DP110102621 and the Sci-

4x4 density		3x3 regular			3x3 density		
Pos. D (22)	Pos. C (6)	Pos. C (6)	Pos. A (21)	Pos. B (1)	Pos. C (5)	Pos. B (13)	Pos. A (10)
ARN	AMS	ARN	AMS	BCN	CPH	ARN	AMS
BCN	FCO	FRA	CDG		LGW	BCN	CDG
CDG	MAD	MAN	CPH		LHR	FRA	DUS
CPH	LGW	LGW	DUS		AGP	MAD	FCO
DUS	DUB	AGP	FCO		CGN	PRG	MAN
FRA	STN	OSL	MAD			ATH	MUC
MAN			MUC			HEL	BRU
MUC			PRG			ZRH	MXP
PRG			ATH			VIE	PMI
ATH			BRU			DUB	STN
BRU			HEL			NCE	
HEL			LHR			OSL	
LHR			MXP			WAW	
MXP			VIE				
VIE			ZRH				
ZRH			CGN				
AGP			DUB				
CGN			NCE				
NCE			PMI				
OSL			STN				
PMI			WAW				
WAW							

Table 8: The three blockmodels and the distribution of the top 20 World and European connected airports in their positions. Next to each position label, we note the number of these top 20 airports in each of the positions. Note: full list of associated cities and countries with these IATA codes can be found at <http://blogs.nonado.net/sammi/2012/03/01/list-of-european-airports-and-their-city-codes/>.

ence Foundation Ireland (SFI) under grant number 08/SRC/I1407. We would like to thank Andrea Lancichinetti, Santo Fortunato, Filippo Radicchi for providing their benchmark software, and Aaron McDaid for the summer school data.

A Block Types and Error functions

For each block type, we have the error functions described in Table 9. Apart from the error function for the functional type, the other functions are available at [16]. The error function for functional are designed to be natural extensions of the ones for row and column functional. Recall that the error

functions, $d(\mathbf{A}(b), \mathbf{I}(t))$, measure the amount of deviation of a block $(\mathbf{A}(b))$ and the ideal structure for type t $(\mathbf{I}(t))$.

The symbols in the table are defined as follows:

- n_1 - number of one's in the block;
- e_{diag} - $\min(n_{diag}, nr - n_{diag})$ (assuming square blocks)
- n_{diag} - number of ones along the diagonal
- nr - number of rows
- nc - number of columns
- mr - maximal row sum
- mc - maximal column sum
- npr - number of rows that at least have a one
- npc - number of columns that at least have a one
- γ - density parameter

Type	Error Function	Comments
Null	n_1 $n_1 + e_{diag} - n_{diag}$	non-diagonal diagonal
Complete	$nr * nc - n_1$ $nr * nc - n_1 + e_{diag} + n_{diag} - nr$	non-diagonal diagonal
Row Dominant	$(nc - mr - 1) * nr$ $(nc - mr) * nr$	diagonal, $n_{diag} = 0$ otherwise
Col Dominant	$(nr - mc - 1) * nc$ $(nr - mc) * nc$	diagonal, $n_{diag} = 0$ otherwise
Row-regular	$(nr - npr) * nc$	
Col-regular	$(nc - npc) * nr$	
Regular	$(nr - npr) * nc + (nc - npc) * nr$	
Row-functional	$n_1 - npr + (nr - npr) * nc$	
Col-functional	$n_1 - npc + (nc - npc) * nr$	
Functional	$n_1 - npr + (nr - npr) * nc + n_1 - npc + (nc - npc) * nr$	
Density	$max(0, \gamma * nr * nc - n_1)$	

Table 9: Table of the block types and their error functions.

B Objective of [30]

$$\max \sum_{P_i \in \mathcal{P}} |P_i|^2 \quad (3)$$

subject to

$$\bigcup_{i=1}^k P_i = V, P_i \cap P_j = \emptyset, 1 \leq i, j \leq k \quad (4)$$

$$(m_1(\mathbf{A}(P_i, P_i)) - \gamma |P_i|^2) \geq 0, \forall P_i \in \mathcal{P} \quad (5)$$

The objective of [30] looks for dense diagonal blocks, which is one of the popular definitions for communities. It tries to maximises the size of the communities found (Equation 3), subject to the communities being partitioned (Equation 4) and the density of each community being greater than or equal to γ , a parameter (Equation 5). It does not consider the non-diagonal blocks at all, nor assign block types to the communities themselves.

In contrast, generalised blockmodelling considers all blocks, even non-diagonal ones. The generalised blockmodel that is most similar to [30]’s diagonal blocks are the density block types with density γ , and null block types for the non-diagonal blocks. Reconsider the formulation for the density block:

$$\max(\gamma |P_i|^2 - m_1(\mathbf{A}(P_i, P_i)), 0) \quad (6)$$

As Equation 6 shows, the objective of the density block is almost the same as the constraint of [30]. However, the density

objective is a soft or error one; it measures how far the density of a block is to a desired density. In contrast, [30] optimises the *size of the positions*, and require *all* blocks to be above a certain density. There is no indication of how close the density of a block is to the desired density. Therefore, these are two different objectives, designed for different problems, and there is no straightforward way to specify the non-diagonal blocks, and hence comparing the two empirically is not transparently meaningful.

References

1. E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing. Mixed membership stochastic blockmodels. *Journal of Machine Learning Research*, 9:1981–2014, June 2008.
2. L. Altenberg. The schema theorem and Prices theorem. *Foundations of Genetic Algorithms*, 3:23–49, 1995.
3. J. Anderson and S. Wasserman. Building stochastic blockmodels. *Social Networks*, 14(1992):137–161, 1993.
4. V. Batagelj, A. Ferligoj, and P. Doreian. Direct and indirect methods for structural equivalence. *Social Networks*, 14:63–90, 1992.
5. S. P. Borgatti and M. G. Everett. Notions of position in social network analysis. *Sociological Methodology*, 22:1–35, 1992.

6. R. Brendel and H. Krawczyk. Extended generalized blockmodeling for compound communities and external actors. pages 32–39, 2009.
7. J. Brynielsson, L. Kaati, and P. Svenson. Social positions and simulation relations. *Social Network Analysis and Mining*, 2:39–52, 2012.
8. T. Bui and B. Moon. Genetic algorithm and graph partitioning. *IEEE Transactions on Computers*, 45(7):841–855, 2002.
9. J. Chan, E. Daly, and C. Hayes. Decomposing discussion forums and boards using user roles. In *Proceedings of the AAAI Conference on Weblogs and Social Media*, 2010.
10. J. Chan, S. Lam, and C. Hayes. Increasing the scalability of the fitting of generalised block models for social networks. In *Proceedings of 22nd AAAI International Joint Conference on Artificial Intelligence*, 2011.
11. J. Chan, W. Liu, C. Leckie, J. Bailey, and R. Kotagiri. SeqBloc: Mining Multi-time Spanning Blockmodels in Dynamic Graphs. In *Proceedings of 18th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2012.
12. C. H. Cheng, W. K. Lee, and K. F. Wong. A genetic algorithm-based clustering approach for database partitioning. *IEEE Transactions on Systems, Man, and Cybernetics*, 32(3):215–230, 2002.
13. A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Physics Review E*, 70(6):066111, Dec 2004.
14. A. Clauset, C. R. Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Rev.*, 51:661–703, November 2009.
15. 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.
16. P. Doreian, V. Batagelj, and A. Ferligoj. *Generalized blockmodeling*. Cambridge Univ Press, 2005.
17. E. Falkenauer. A new representation and operators for genetic algorithms applied to grouping problems. *Evolutionary Computation*, 2(2):123–144, 1994.
18. E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of heuristics*, 2(1):5–30, 1996.
19. J. Fiala and D. Paulusma. The computational complexity of the role assignment problem. Technical report, 2003.
20. D. Fisher, M. Smith, and H. T. Welsher. You are who you talk to: detecting roles in usenet newsgroups. In *Proceedings of the 39th Annual Hawaii International Conference*, volume 3, pages 59–68, January 2006.
21. G. W. Flake, S. Lawrence, and C. L. Giles. Efficient identification of web communities. In *Proceeding of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 150–160, 2000.
22. S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75 – 174, 2010.
23. F. Gilbert, P. Simonetto, F. Zaidi, F. Jourdan, and R. Bourqui. Communities and hierarchical structures in dynamic social networks: analysis and visualization. *Social Network Analysis and Mining*, 1:83–95, 2011.
24. D. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-wesley, 1989.
25. S. A. Golder and J. Donath. Social roles in electronic communities. In *Proceedings of the Association of Internet Researchers*, 2004.
26. D. Greene, D. Doyle, and P. Cunningham. Tracking the evolution of communities in dynamic social networks. In *Proceedings of the 2010 International Conference on Advances in Social Network Analysis and Mining (ASONAM)*, pages 176 –183, 2010.
27. M. S. Handcock and A. E. Raftery. Model-based clustering for social networks. *Journal of Royal Statistical Society*, 170(2):301–354, 2007.
28. D. He, Z. Wang, B. Yang, and C. Zhou. Genetic Algorithm with Ensemble Learning for Detecting Community Structure in Complex Networks. In *Proceedings of the 4th International Conference on Computer Sciences and Convergence Information Technology*, pages 702–707, 2009.
29. P. D. Hoff, A. E. Raftery, and M. S. Handcock. Latent space approaches to social network analysis. *Journal of the American Statistical Association*, 97(460):1090–1098, 2002.
30. T. James, E. Brown, and C. T. Ragsdale. Grouping genetic algorithm for the blockmodel problem. *IEEE Transactions on Evolutionary Computing*, 14(1):103–111, 2010.
31. D. S. Johnson, C. R. Aragon, L. A. 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.
32. G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. *VLSI Design*, 11(3):285–300, 2000.

33. B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307, 1970.
34. S. Kirkpatrick, C. D. Gelatti, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.
35. A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78(4):46110, 2008.
36. J. Lerner. Role assignments. In *Network Analysis*, pages 216–252, 2004.
37. F. Lorrain and H. White. Structural equivalence of individuals in social networks. *The Journal of Mathematical Sociology*, 1(1):49–80, 1971.
38. P. Malighetti, S. Paleari, and R. Redondi. Connectivity of the european airport network. *Journal of Air Transport Management*, 14(2):53–65, 2008.
39. 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.
40. Z. Michalewicz. *Genetic algorithms + data structures = evolution programs*. Springer-Verlag, London, UK, 3rd edition, 1996.
41. K. Nowicki and T. Snijders. Estimation and prediction for stochastic blockstructures. *Journal of American Statistical Association*, 96(455):1077–1087, 2001.
42. J. Pei, D. Jiang, and A. Zhang. On mining cross-graph quasi-cliques. In *Proceeding of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 228–238, New York, New York, USA, 2005. ACM Press.
43. J. Reichardt and D. R. White. Role models for complex networks. *The European Physical Journal B*, 60(2):217–224, 2007.
44. M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105:1118–1123, 2008.
45. J. Scott. Social network analysis: developments, advances, and prospects. *Social Network Analysis and Mining*, 1:21–26, 2011.
46. J. Shetty and J. Adibi. The enron email dataset: Database schema and brief statistical report. Technical report, ISI, University of Southern California, 2004.
47. T. C. Turner and K. E. Fisher. The impact of social types within information communities: Findings from technical newsgroups. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences*, page 135.2, Washington, DC, USA, 2006. IEEE Computer Society.
48. K. Wagstaff and C. Cardie. Clustering with instance-level constraints. In *Proceedings of the 17th International Conference on Machine Learning*, pages 1103–1110, 2000.
49. S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1 edition, 1994.
50. H. Welser, G. Kossinets, M. Smith, and D. Cosley. Finding social roles in wikipedia. In *Presented at the Annual Meeting of the American Sociological Association Annual Meeting*, pages 1–11, July 2008.
51. H. C. White, S. A. Boorman, and R. L. Beiger. Social Structure from Multiple Networks. I. Blockmodels of Roles and Positions. *American Journal of Sociology*, 81(4):730–780, 1976.
52. H. William, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Simulated Annealing Methods*, chapter 10, pages 444–455. Cambridge University Press, 2nd edition, 1992.
53. E. P. Xing, W. Fu, and L. Song. A state-space mixed membership blockmodel for dynamic network tomography. *Annals of Applied Statistics*, 4(2):535–566, 2010.
54. H. Zha, X. He, C. Ding, H. Simon, and M. Gu. Bipartite graph partitioning and data clustering. *Proceedings of the 10th International Conference on Information and Knowledge Management*, page 25, 2001.
55. A. Ziberna. *Generalized blockmodeling of valued networks*. PhD thesis, University of Ljubljana, 2007.

Minerva Access is the Institutional Repository of The University of Melbourne

Author/s:

Chan, J;Lam, S;Hayes, C

Title:

Generalised blockmodelling of social and relational networks using evolutionary computing

Date:

2014-12-01

Citation:

Chan, J., Lam, S. & Hayes, C. (2014). Generalised blockmodelling of social and relational networks using evolutionary computing. SOCIAL NETWORK ANALYSIS AND MINING, 4 (1), <https://doi.org/10.1007/s13278-014-0155-y>.

Persistent Link:

<http://hdl.handle.net/11343/283023>