Journal of Artificial Intelligence Research 70 (2021) 597-630 Submitted 07/2020; published 02/2021

Generic Constraint-Based Block Modeling using Constraint Programming

Alex Lucía Mattenet ALEX.MATTENET@UCLOUVAIN.BE
Institute for Information and Communication Technologies,
Electronics and Applied Mathematics, UCLouvain
3 place du Levant, 1348 Louvain-la-Neuve, Belgium

Ian Davidson DAVIDSON@CS.UCDAVIS.EDU
Computer Science Department, University of California, Davis
One Shields Ave., Davis, CA 95616-5270, USA

Siegfried Nijssen SIEGFRIED.NIJSSEN@UCLOUVAIN.BE
Pierre Schaus PIERRE.SCHAUS@UCLOUVAIN.BE
Institute for Information and Communication Technologies,
Electronics and Applied Mathematics, UCLouvain
3 place du Levant, 1348 Louvain-la-Neuve, Belgium

Abstract

Block modeling has been used extensively in many domains including social science, spatial temporal data analysis and even medical imaging. Original formulations of the problem modeled it as a mixed integer programming problem, but were not scalable. Subsequent work relaxed the discrete optimization requirement, and showed that adding constraints is not straightforward in existing approaches. In this work, we present a new approach based on constraint programming, allowing discrete optimization of block modeling in a manner that is not only scalable, but also allows the easy incorporation of constraints. We introduce a new constraint filtering algorithm that outperforms earlier approaches, in both constrained and unconstrained settings, for an exhaustive search and for a type of local search called Large Neighborhood Search. We show its use in the analysis of real datasets. Finally, we show an application of the CP framework for model selection using the Minimum Description Length principle.

1. Introduction

Block modeling has a long history in the analysis of social networks (Wasserman & Faust, 2018). The core problem is to take a graph and divide it into k clusters and interactions between those clusters described by a $k \times k$ image matrix. The purpose is to summarize a complex graph to be better understood by humans.

More formally, in its simplest formulation, the core problem is: given a graph $G(V, E)$ whose $n \times n$ adjacency matrix is X, simplify X into a symmetric trifactorization FMF^T. Here F is an $n \times k$ block allocation matrix with the blocks/clusters stacked column wise, and $F_{i,j} \in \{0, 1\}$. M is a $k \times k$ interaction (image) matrix showing the interaction between blocks. The objective function is to minimize the reconstruction error $||X - FMF^T||$.

This block modeling formulation has the advantage of identifying structural equivalence: if the reconstruction error is 0, any instance in cluster i must have the exact same neighbors in the graph. The reconstruction error ($||X - FMF^T||$) counts the number of edges that violate this property.

The original MIP formulations of block modeling were lacking in two ways. Firstly, they were not scalable; secondly, they often found results that were inconsistent with the expectations of domain experts. To overcome both problems, the use of constraints has been studied in the literature. For example i) Entry level constraints such as non-negativity (Wang et al., 2011), ii) Incorporating simple composition constraints on the blocks such as spatial continuity or together/apart constraints (Bai et al., 2017; Ganji et al., 2018b), iii) Constraints on the interaction/image matrix (Ganji et al., 2018a) and even iv) Simultaneous constraints on blocks and interaction/image matrices (Bai et al., 2018).

However, each of these studies yielded a different type of solver that is only scalable for one specific problem setting. For example, Bai et al. (2017) used an update rule for the lagrangian multiplier method, and Bai et al. (2018) a multiplicative update rule. Hence, it is impossible to use all of these constraints at the same time; the approaches are either not usable or not scalable without the predefined constraints.

In this paper we propose a novel approach to block modeling based on Constraint Programming. The advantage of CP is that it offers a generic and modular approach to solving constraint satisfaction and optimization problems by means of global constraints. Global constraints can be combined to solve problems involving multiple constraints. In this work, we introduce a global constraint for block modeling. This allows solving block modeling problems under additional constraints such as (a) upper and lower bounds on the cluster size; (b) complex requirements in conjunctive normal form, such as that if vertex i is in the same cluster as j, then k and l must not be; (c) constraints on the structure of the image graph M, forcing it to be a tree, a ring graph, a star graph, ...; (d) connectivity constraints: we can require that the subgraph induced by the nodes in each cluster is connected; (e) bin packing constraints: given a weight for each vertex, limit the total weight of each cluster; and more.

Such constraints now allow combining strong semantic knowledge (the constraints) along with empirical evidence (the graph).

The focus of this work is primarily on how to build a filtering algorithm for block modeling that works well in practice. We will demonstrate this on a number of experiments on both datasets used in earlier studies and new problems that we propose in this work. We will show that our propagator is correct for the constraint that it implements and outperforms an existing equivalent method by orders of magnitude.

Then, we adapt our solver to perform a metaheuristic called Large Neighborhood Search (Shaw, 1998). This allows our method to be used on larger instances — up to thousands of vertices. We validate the performance of our LNS implementation by comparing it to an existing local search for block modeling, and show its scalability on synthetic datasets.

Finally, we will tackle the problem of model selection for the block modeling problem. In traditional block modeling, the number of clusters k is assumed to be known in advance, but this is not always the case. We explore the application of the Minimum Description Length principle to the block modeling problem, and propose a simple adaptation of our solver to automatically discover the number of clusters that best describes the data.

The code for the algorithms and CP models described in this article are available online on `https://github.com/gadevoi/blockmodel-cp/`, along with the executables and graph files used for the experiments.

2. Related Work

Block modeling in practice has two core computational challenges: i) to give truly interpretable results, we must guarantee the optimality of the solution. ii) constraints are required to make results realistic in that they are consistent with human expectations.

Take for example the application of block modeling on Twitter data from the US elections. Each person/account should be allocated to a cluster, and we wish to efficiently find clusters consistent with our expectations (i.e. that Donald Trump will not be in the same cluster as Hillary Clinton).

There have been two lines of work to address both challenges, but no work attempts to address both. For the first challenge, Brusco and Steinley (2009) introduced a MIP formulation for the block modeling problem where the image matrix is fixed, which is then extended to search over all possible image matrices by Dabkowski et al. (2016)—but as we show in this paper (Table 4) their run time is extremely slow.

Most of the work in the block modeling literature has been on heuristic methods for finding a good, although not optimal, block model. A classical approach is to use a local search that starts from a given partition of the vertices, and swaps vertices from one cluster to another to improve the fit of the block model (Batagelj et al., 1992b), which is the basis of the transfer and exchange algorithm (TEA) presented by Doreian et al. (2005). Others relax the requirement that the matrices F and M be binary, settling instead for non-negative matrices and solving them with techniques from non-negative matrix factorization (Long et al., 2010; Wang et al., 2011). We also find metaheuristic approaches for variations of the block modeling problem using grouping genetic algorithm or artificial bee colony algorithm methods (Sundar & Singh, 2015).

Some related works using a continuous relaxation of the problem studied variations that added constraints on the block model. There is a plethora of such constraints, some of which we outline in Table 1. Unfortunately, these methods cannot be combined to create a block modeling solver that uses all constraints as they use different underlying solving methods. Furthermore, these solvers do not yield exact solutions for the discrete allocation problem. All of these constraints and others mentioned in the introduction (i.e. cardinality constraints) can, however, easily be encoded in our exact CP model.

We find frameworks using constraint programming for the addition of arbitrary constraints in the related field of clustering. In clustering, we are searching for highly connected clusters that have very few edges towards the rest of the graph. For this problem, it is desirable to be able to set arbitrary constraints on the clusters for the same reasons as outlined in our article. Dao et al. (2016) introduced a framework entirely in CP. Another approach is to first find a good clustering using classical clustering algorithms, then try to modify the clusters to respect the additional constraints (Kuo et al., 2017).

Existing works have introduced an MDL criterion for model selection in *stochastic block modeling*, which is a stochastic generalization of the block model in which ties between the clusters represent the probability of observing an edge between the vertices they contain

Table 1: A list of some complex constraints used to solve continuous optimization versions of block modeling. These methods cannot be combined as they use different underlying solvers, whereas our method can address all of these constraints.

Constraint	Description	Solver Used
Spatial Continuity (Bai et al., 2017)	A soft constraint based on a kernel	Additive Update Rule
Path (Bai et al., 2018)	All nodes in a block have a path to each other	Multiplicative Update Rule
Composition (Ganji et al., 2018b)	Must-link/cannot-link constraints	Gradient Descent
Image Structure (Ganji et al., 2018a)	Constraints on image matrix	Gradient Descent

(Holland et al., 1983). Notably, such framework is explored by Yan (2016) and by Peixoto first in (2013), and then more expansively in (2017a), which notably includes deriving theoretical limitations on the detectability of clusters and proposes MDL criteria and adequate local search procedures for variations on stochastic block modeling such as *degree-corrected block modeling* and *hierarchical block modeling*. Other works on model selection for the stochastic block modeling problem use the mutual information metric between the model and the graph, as is done by Rosvall and Bergstrom (2007).

In the previously cited works, the interpretation of stochastic block models as a probability distribution over all graphs of size n gives an appropriate coding scheme for which the description length of a graph with probability p is $-\log_2(p)$. In our setting, there is no such probabilistic interpretation. To derive the coding scheme, we can draw inspiration from the field of binary matrix factorisation (BMF). In block modeling, we look for a factorisation of the adjacency matrix X into FMF^t, with all matrices boolean and the conditions presented before. In BMF, we look for the factorization of any boolean matrix A into BC, with B and C both boolean matrices. In both settings, the fit of the model to the data is rarely perfect, instead we look for a model which encodes most of the structure with minimal error. A study of the application of MDL to BNF is given by Miettinen and Vreeken (2014).

3. Problem Statement: Block Modeling for Structural Equivalence

The assumption underlying block modeling is that every vertex plays a role in the network, and the ties that this vertex will have with other vertices depend on their respective role. Vertices playing an equivalent role are grouped in clusters, and the structure of the graph is summarized with the graph of connections between the different clusters (the *image graph*).

Different definitions of equivalence between vertices have been proposed in the block modeling literature. The one most commonly used, called "structural equivalence", dictates that two vertices are equivalent if they are connected to exactly the same other vertices in the network (Lorrain & White, 1971). Formally, given a graph $G = (V, E)$, vertices $u, v \in V$

$$\begin{tikzcd}
1 \arrow[r] \arrow[dr] & 3 \arrow[r] \arrow[loop above] \arrow[d, leftrightarrow] & 5 \\
2 \arrow[r] \arrow[ur] & 4 \arrow[loop right] \arrow[ur] &
\end{tikzcd}
\qquad
X = \left(\begin{array}{cc|cc|c}
\cdot & \cdot & 1 & 1 & \cdot \\
\cdot & \cdot & 1 & 1 & \cdot \\
\hline
\cdot & \cdot & 1 & 1 & 1 \\
\cdot & \cdot & 1 & 1 & 1 \\
\hline
\cdot & \cdot & \cdot & \cdot & \cdot
\end{array}\right)$$

Figure 1: Small digraph along with its adjacency matrix X. According to structural equivalence, $1 \equiv 2$ and $3 \equiv 4$.

$$\begin{tikzcd}
\{1,2\} \arrow[r] & \{3,4\} \arrow[r] \arrow[loop above] & \{5\}
\end{tikzcd}
\qquad
M = \begin{pmatrix}
\cdot & 1 & \cdot \\
\cdot & 1 & 1 \\
\cdot & \cdot & \cdot
\end{pmatrix}$$

Figure 2: Image graph of Figure 1 along with its image matrix M.

are structurally equivalent $u \equiv v$ if and only if $\forall x \in V : (u,x) \in E \iff (v,x) \in E \wedge (x,u) \in E \iff (x,v) \in E$.

For example, consider the digraph, along with its adjacency matrix, in Figure 1. Vertices 1 and 2 are structurally equivalent, since they are both connected to vertices 3 and 4 and nothing else. The equivalence classes according to $\equiv$ define a partition of the vertices into to three clusters, $V_1 = \{1,2\}, V_2 = \{3,4\}$ and $V_3 = \{5\}$. Observe that in the adjacency matrix, the rows and columns of equivalent vertices are identical. This gives rise to *blocks* in the matrix, delimited by lines in Figure 1. In this example, the vertices of the same block are numbered sequentially, but in practice the rows and columns have to be reordered to show the blocks in the matrix. Structural equivalence dictates that blocks be either *Null blocks* (containing only 0) or *Complete Blocks* (containing only 1) (Batagelj, 1997).

The image graph is shown in Figure 2. It has one vertex for each cluster, and the edges are given by the blocks in X. We can reconstruct the adjacency matrix X from the image matrix M in the following way. Let F be a 5×3 matrix such that $F_{ik} = 1$ if vertex i is in cluster k, otherwise $F_{ik} = 0$. Then we have $X = FMF^T$, where F^T is F transposed.

Structural equivalence is a very strong requirement. In order to deal with the noise in real-world data, we will look for an F and M which approximate the base graph X with the least error, for a fixed model size k. We define the error (the *cost* of the solution) as the number of edges which must be added or deleted from our graph in order to fit the model perfectly: $||X - FMF^T|| = \sum_{i=1}^n \sum_{j=1}^n |X_{ij} - (FMF^T)_{ij}|$. Formally, the minimization problem Blockmodel(X,k) that we are solving in the absence of other constraints is as follows: given $X \in \mathbb{B}^{n \times n}$ a binary adjacency matrix and number of clusters k, find F and M such that

$$\min_{F,M} \quad ||X - FMF^T|| \tag{1}$$

$$\text{s.t.} \quad \sum_{c=1}^{k} F_{ic} = 1 \qquad \forall i \in \{1..n\} \tag{2}$$

$$\sum_{i=1}^{n} F_{ic} \geq 1 \qquad \forall c \in \{1..k\}\,. \tag{3}$$

$F \in \mathbb{B}^{n \times k}$ is the indicator matrix and $M \in \mathbb{B}^{k \times k}$ is the image matrix of our model. Equation (2) ensures that vertices are assigned to one cluster only, while equation (3) ensures that there are no empty clusters. To this model, additional constraints can be added.

4. CP Model for Block Modeling with a Global Constraint

The main contributions of this article are (1) a CP model for the block modeling problem, (2) a global constraint used in this model, which we call BlockModelCost, (3) a tailored filtering algorithm for this constraint for an exhaustive search as well as for local search and (4) a formulation of the Minimum Description Length principle for the block modeling problem. We first describe the CP model, with its variables and constraints. Afterwards, we present the global constraint and its filtering algorithm. Then, we present a heuristic and symmetry breaking scheme for the CP solver based on the global constraint. Finally, we discuss their adaptation for the LNS and MDL settings.

We denote the domain of a CP variable v with dom(v) and write $\mathsf{v} = x$ when variable v is assigned to x, that is to say when dom(v) $= \{x\}$. There are four groups of variables in our model: the n cluster variables C, the k^2 image matrix variables M, the k^2 block cost variables cost and the total cost of our solution $\mathsf{totalCost}$. They are presented in this table, along with their initial domain:

Variable	Initial Domain	Interpretation
C_i	$\{1..k\}$	$\mathsf{C}_i = c$ if vertex i is in cluster c
M_{cd}	$\{0, 1\}$	$\mathsf{M}_{cd} = 0$ if the submatrix of rows in cluster c and columns in cluster d is a modelled as a Null block, and $\mathsf{M}_{cd} = 1$ if it is a Complete Block
cost_{cd}	$\{0..n^2\}$	Number of entries in the submatrix c, d which do not match M_{cd}
$\mathsf{totalCost}$	$\{0..n^2\}$	The cost of the solution $\|\|X - FMF^T\|\|$

The variables are subject to the following constraints:

- `sum`($\mathsf{cost}, \mathsf{totalCost}$), which ensures that the total cost of the solution and the individual cost of every block stays consistent: $\sum_{c=1}^{k} \sum_{d=1}^{k} \mathsf{cost}_{cd} = \mathsf{totalCost}$.

- `gcc`($\mathsf{C}, values = \{1..k\}, min = 1, max = |\mathsf{C}|$). The global cardinality constraint `gcc`($X, values, min, max$) ensures that the every value in $values$ appears between min and max times in the final assignment of the variables in the array X. In our model, it ensures that every value between 1 and k appears at least once in C — i.e. there are no empty clusters, as per Equation (3).

- `blockModelCost`($X, \mathsf{M}, \mathsf{C}, \mathsf{cost}, \mathsf{totalCost}$). This is the global constraint that we add to the solver, which filters the values of the different variables along the search. It ensures $\sum_{i=1}^{n} \sum_{j=1}^{n} |X_{ij} - M_{C_i C_j}| = \mathsf{totalCost}$ and $\sum_{i=1}^{n} \sum_{j=1}^{n} (C_i = c) \cdot (C_j = d) \cdot |X_{ij} - M_{cd}| = \mathsf{cost}_{cd}\ \forall c, d$; where $(C_i = x)$ denotes a function of x equal to 1 if C_i is equal to x and 0 otherwise.

Table 2: Adjacency matrix with its columns and rows reordered to show the partial assignment of vertices into clusters. Zeroes are rendered as · for legibility.

cluster	1				2	3		unbound	
vertex	1	2	7	9	5	3	4	6	8
1	·	·	·	1	·	1	1	1	·
2	·	·	·	·	1	1	1	1	·
7	·	1	·	·	·	1	·	·	·
9	·	·	·	·	·	·	1	1	·
5	·	·	·	·	1	1	1	1	1
3	1	1	·	·	·	·	·	·	·
4	·	1	1	1	·	·	1	·	·
6	**1**	**1**	·	**1**	·	1	·	·	·
8	·	·	·	·	1	1	1	1	·

Note that the constraint of equation (2) (vertices can only be in one cluster) is implicitly modeled by the variable C. Since in the final solution, all variables must be bound to a single value, no vertex is bound to more than one cluster.

The model can be extended with any set of existing additional constraints present in CP systems, on any of the variables, such as cardinality constraints or connectivity constraints.

5. A Global Constraint for Block Modeling

A global constraint (van Hoeve & Katriel, 2006) is a constraint that captures a relationship between a number of variables. Typically, a global constraint, as this one, can also be decomposed into several simpler constraints but considering it globally, often, permits filtering more inconsistent values and is often also faster (Bessiere & Van Hentenryck, 2003). Global constraints are thus key to prune the search tree and solve complex problems efficiently with CP. The filtering algorithm of the global constraint is called every time the domain of one variable in its scope changes. This filtering does not need to be complete, although it needs to be able to check the feasibility when all the variables are bound and it must also guarantee that no valid values are removed.

In this subsection, we present `blockModelCost`, a global constraint for block modeling. We first give a concrete example to illustrate the filtering strategies. Then, we describe the pseudo-code for the propagation method.

5.1 Illustration of the Different Filtering Strategies

To illustrate the filtering algorithm, let's consider the following partial assignment: $\mathrm{dom}(\mathsf{C}) = (\{1\},\{1\},\{3\},\{3\},\{2\},\{1,2,3\},\{1\},\{1,2,3\},\{1\}), \forall c,d : \mathrm{dom}(\mathsf{M}_{cd}) = \{0,1\}, \mathrm{dom}(\mathsf{cost}_{cd}) = \{0..13\}$ and $\mathrm{dom}(\mathsf{totalCost}) = \{0..13\}$. In Table 2, we show the adjacency matrix X for this example with its rows and columns reordered to show the current partial assignment.

5.1.1 Filtering cost_{cd}

If we look at the submatrix defined by what is already assigned to the block (1,1) — i.e. the northwestern block in Table 2 — we see that it contains fourteen 0s and two 1s. If $\mathsf{M}_{1,1} = 0$, the block should be filled with 0s so its cost will be at least 2, because of the two 1s. It could be more than 2 if other vertices are bound to cluster 1, but it can never be less than 2. If $\mathsf{M}_{1,1} = 1$, the cost will be at least 14, because of the fourteen 0s. Thus, we can increase the lower bound of the domain of $\mathsf{cost}_{1,1}$ to 2. Doing this for all blocks, the state of the cost variables become

$$\mathrm{dom}(\mathsf{cost}) = \begin{pmatrix} \{2..13\} & \{1..13\} & \{2..13\} \\ \{0..13\} & \{0..13\} & \{0..13\} \\ \{3..13\} & \{0..13\} & \{1..13\} \end{pmatrix}$$

After propagating the $\mathtt{sum}$ constraint, we get $dom(\mathsf{totalCost}) = \{9..13\}$ and

$$dom(\mathsf{cost}) = \begin{pmatrix} \{2..6\} & \{1..5\} & \{2..6\} \\ \{0..4\} & \{0..4\} & \{0..4\} \\ \{3..7\} & \{0..4\} & \{1..5\} \end{pmatrix}$$

5.1.2 Filtering M_{cd}

As observed previously, setting $\mathsf{M}_{1,1}$ to 1 would bring the minimum cost of the block to 14. However, the value 14 is not in the domain of $\mathsf{cost}_{1,1}$, so we can filter the value 1 from $\mathsf{M}_{1,1}$, in effect binding it to $\mathsf{M}_{1,1} = 0$.

5.1.3 Filtering C_i

If we were to assign vertex 6 to cluster 1, it would add six 1s to the (1,1) block — three from the partial column representing edges from vertices in cluster 1 to vertex 6, and three more from the partial row representing edges from vertex 6 to vertices in cluster 1. Remember that $\mathsf{M}_{1,1} = 0$, so each one would increase the cost of the block. The resulting cost (8) would exceed the maximum allowed value for $\mathsf{cost}_{1,1}$, so we can remove 1 from the domain of C_6.

5.1.4 Tightening the Lower Bound on $\mathsf{totalCost}$

In what has been described so far, the lower bound of $\mathsf{totalCost}$ is only the sum of the lower bounds of the individual cost variables. These take into account only the submatrix defined by the vertices already assigned to a specific cluster. We can improve the bound by also taking into account the unbound vertices (vertices 6 and 8 in our example). In Table 2, consider the horizontal rectangle in bold at row 6. It corresponds to the edges going from vertex 6 to vertices in cluster 1. Since $\mathrm{dom}(\mathsf{C}_6) = \{2, 3\}$, we do not know yet in which block it will be, but those 4 values will stay together in the final assignment. If the 4 values end up in a Null block, their cost will be 3, and if they end up in a Complete block, their cost will be one, so we can at least increase the lower bound on $\mathsf{totalCost}$ by one. The same can be done for all other rectangles in the “unbound” part of Table 2 except for the southeastern corner (edges between unbound vertices). If we add all of these contributions, we get $\mathrm{dom}(\mathsf{totalCost}) = \{12..13\}$.

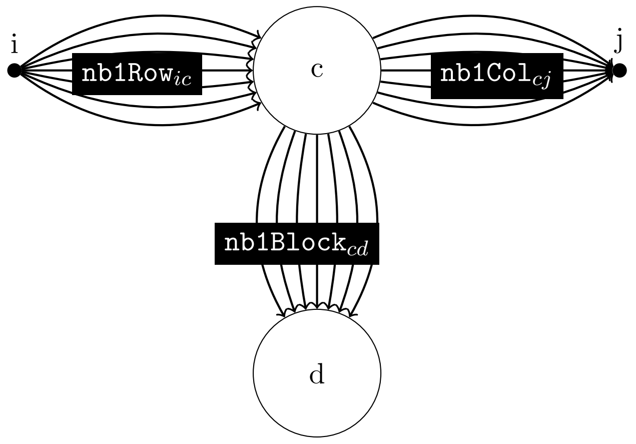

Figure 3: Illustration of the different counters used in the filtering algorithm.

5.2 Filtering Algorithm

The pseudo-code for our propagation method is shown in Algorithm 1. In order to filter the domains of our CP variables efficiently, the number of zeroes and ones in the different "blocks" of our reordered matrix are computed. For efficiency reasons, those counters are stored on a trail (Schulte, 1999), or more exactly inside reversible integers that are restored on backtracking. This permits an incremental update based on the changes since the last call to the filtering algorithm without having to worry about the restoration at backtracking. Specifically, these values are stored as reversible integers:

- `nb0Block`, a $k \times k$ array reflecting the number of zeroes already assigned to each block: $\texttt{nb0Block}_{cd} = \#\{X_{ij} \mid \mathsf{C}_i = \{c\}, \mathsf{C}_j = \{d\}, X_{ij} = 0\}$,
- `nb0Row`, a $n \times b$ array where for all unbound vertices i: $\texttt{nb0Row}_{ic} = \#\{X_{ij} \mid \mathsf{C}_j = \{c\}, X_{ij} = 0\}$,
- `nb0Col`, a $b \times n$ array where for all unbound vertices i: $\texttt{nb0Col}_{ci} = \#\{X_{ji} \mid \mathsf{C}_j = \{c\}, X_{ji} = 0\}$,

as well as their equivalent variables for the number of ones: `nb1Block`, `nb1Row` and `nb1Col`. They are illustrated in Figure 3. The set of unbound vertices $\texttt{unboundVertices} = \{i \in \{1..n\} \mid 1 < \#\,\mathrm{dom}(\mathsf{C}_i)\}$ is maintained in a reversible sparse set (le Clément de Saint-Marcq et al., 2013).

A lower bound on the cost of the block c, d is:

$$\underline{cost}(c,d) = \begin{cases} \texttt{nb0Block}_{cd} & \text{if } \mathsf{M}_{cd} = \{1\} \\ \texttt{nb1Block}_{cd} & \text{if } \mathsf{M}_{cd} = \{0\} \\ \min(\texttt{nb0Block}_{cd}, \texttt{nb1Block}_{cd}) & \text{otherwise.} \end{cases}$$

We obtain a better bound by also maintaining $\underline{\text{rowcost}}$, using the method described in the earlier paragraph "Tightening the lower bound on $\mathsf{totalCost}$", as follows:

$$\underline{rowcost}(c,i) = \begin{cases} \texttt{nb0Row}_{ic} & \text{if } \forall d: \mathsf{M}_{dc} = \{1\} \\ \texttt{nb1Row}_{ic} & \text{if } \forall d: \mathsf{M}_{dc} = \{0\} \\ \min(\texttt{nb0Row}_{ic}, \texttt{nb1Row}_{ic}) & \text{otherwise.} \end{cases}$$

Similarly we maintain $\underline{colcost}(c,i)$, defined equivalently from `nb0Col` and `nb1Col`. They put a lower bound on the cost incurred by rows and columns of vertices which have not been bound yet.

Finally, we can also calculate a lower bound on the added cost for block (c,d) if we put vertex i in cluster x, $\underline{\delta_{i\mapsto x}}(c,d) = \underline{cost_{i\mapsto x}}(c,d) - \underline{cost}(c,d)$ where $\underline{cost_{i\mapsto x}}(c,d)$ is the value of $\underline{cost}(c,d)$ if vertex i is assigned to cluster x.

The first step in our algorithm is to process all the vertices that have been bound to a cluster since the propagation method was last called, and update the constraint's variables. Then, we filter the CP variables cost_{cd} with the new lower bounds $\underline{cost}(c,d)$, and filter $\mathsf{totalCost}$ further with $\underline{rowcost}$ and $\underline{colcost}$. Then we filter the values of M_{cd} by removing the values which lead to a cost higher than $\max(\mathsf{cost}_{cd})$. Finally, we filter the values of C_i with the lower bounds of $\underline{\delta_{i\mapsto x}}(c,d)$. This order of steps was chosen as we found it to perform well in practice.

5.3 Theoretical Properties of the Algorithm

When discussing the filtering algorithm for global constraint `C`, one distinguishes *complete filtering* from *partial filtering*. If the algorithm removes every useless value from the domain of every variable that `C` is defined on, we say that it achieves *complete filtering*. If it removes only some useless values, we say that it performs *partial filtering*. (Rossi, 2006)

Formally, complete filtering algorithms achieve a property called generalized arc consistency (or arc consistency for short): Let `C` be a constraint on the variables $\mathsf{x_1}, \dots, \mathsf{x_k}$ with respective domains $\text{dom}(\mathsf{x_1}), \dots, \text{dom}(\mathsf{x_1})$. That is, $\mathtt{C} \subseteq \text{dom}(\mathsf{x_1}) \times \cdots \times \text{dom}(\mathsf{x_k})$. We say that `C` is generalized arc consistent if for every $1 \le i \le k$ and $v \in \text{dom}(\mathsf{x_i})$, there exists a tuple $(d_1, \dots, d_k) \in \mathtt{C}$ such that $d_i = v$.

In general, establishing arc consistency for a non-binary constraint (or global constraint) is NP-hard (Rossi, 2006). For instance, an arc-consistent filtering algorithm for `blockModelCost` would keep only the values of $\mathsf{totalCost}$ v such that there exists an assignment of the remaining variables from their respective domain defining a block model with cost v; and similarly for every other variable in the constraint ($\mathsf{C}_i, \mathsf{M}_{cd}, \mathsf{cost}_{cd}\ \forall i, c, d$). Our application does not need this much filtering: indeed, when minimizing $\mathsf{totalCost}$, we only care to know if the current partial assignment can lead to a better solution. Filtering out values of $\mathsf{totalCost}$ that are higher than the current minimum gives us no information to that end, it only tells us that some particular realization of a worse block model is not achievable.

Weaker levels of filtering have been defined, such as bound consistency: Let `C` be a constraint on the variables $\mathsf{x_1}, \dots, \mathsf{x_k}$ with respective interval $\text{dom}(\mathsf{x_1}), \dots, \text{dom}(\mathsf{x_k})$. We say that `C` is bound consistent if for every $1 \le i \le k$, there exists a tuple $(d_1, \dots, d_k) \in \mathtt{C}$ such that $d_i = \min \text{dom}(\mathsf{x_i})$ and there exists a tuple $(e_1, \dots, e_k) \in \mathtt{C}$ such that $e_i = \max \text{dom}(\mathsf{x_i})$.

Algorithm 1 Propagation of our global constraint.

$\Delta\mathsf{C}$ is a list of all the variables C_i which have been bound since the last propagation of this constraint.

```
1: /* update local counters */
2: for all $\mathsf{C}_i = \{c\} \in \Delta\mathsf{C}$ do
3:   $\texttt{unboundVertices} \leftarrow \texttt{unboundVertices} \setminus \{i\}$
4:   for all $j$ in $\texttt{unboundVertices}$ do
5:     $\texttt{nb1Col}_{cj} \mathrel{+}= X_{ij}; \texttt{nb0Col}_{cj} \mathrel{+}= (1 - X_{ij})$
6:     $\texttt{nb1Row}_{jc} \mathrel{+}= X_{ji}; \texttt{nb0Row}_{jc} \mathrel{+}= (1 - X_{ji})$
7:   end for
8:   for all $d = 1$ to $k$ do
9:     $\texttt{nb0Block}_{cd} \mathrel{+}= \texttt{nb0Row}_{id}; \texttt{nb1Block}_{cd} \mathrel{+}= \texttt{nb1Row}_{id}$
10:    $\texttt{nb0Block}_{dc} \mathrel{+}= \texttt{nb0Col}_{di}; \texttt{nb1Block}_{dc} \mathrel{+}= \texttt{nb1Col}_{di}$
11:  end for
12:  $\texttt{nb0Block}_{cc} \mathrel{+}= (1 - X_{ii}); \texttt{nb1Block}_{cc} \mathrel{+}= X_{ii}$
13: end for
14: /* filter cost and totalCost */
15: minCost $\leftarrow 0$
16: for all $c, d \in \{1..k\} \times \{1..k\}$ do
17:   update min of $\mathsf{cost}_{cd}$ to $\underline{cost}(c, d)$.
18:   minCost $\mathrel{+}= \underline{cost}(c, d)$.
19: end for
20: for all unbound vertex $i$, $c = 0$ to $k$ do
21:   minCost $\mathrel{+}= \underline{colcost}(c, i) + \underline{rowcost}(c, i)$
22: end for
23: update min of totalCost to minCost
24: /* filter M */
25: for all $c, d \in \{1..k\} \times \{1..k\}$ if $\mathsf{M}_{cd} = \{0, 1\}$ do
26:   if $\texttt{nb0Block}_{cd} > \max(\mathsf{cost}_{cd})$ then $\mathsf{M}_{cd} \leftarrow \mathsf{M}_{cd} \setminus \{0\}$ end if
27:   if $\texttt{nb1Block}_{cd} > \max(\mathsf{cost}_{cd})$ then $\mathsf{M}_{cd} \leftarrow \mathsf{M}_{cd} \setminus \{1\}$ end if
28: end for
29: /* filter C */
30: for all $i \in \texttt{unboundVertices}$, $c \in \mathsf{C}_i, d = 1$ to $k$ do
31:   if $\underline{cost}(c, d) + \underline{\delta_{i \mapsto c}}(c, d) > \max(\mathsf{cost}_{cd})$ or $\underline{cost}(d, c) + \underline{\delta_{i \mapsto c}}(d, c) > \max(\mathsf{cost}_{dc})$ then
32:     $\mathsf{C}_i \leftarrow \mathsf{C}_i \setminus \{c\}$
33:   end if
34: end for
```

Here again, this formal property does not fit conceptually for our objective: the upper and lower bounds of the C and M variables are not meaningful. For cost and totalCost, only the lower bound is used in the minimization.

Even in its weaker form, ensuring bound consistency on totalCost was judged to be out of scope for this article. Indeed, we would have to ensure that the lower bound of totalCost is supported by a at least one block model existing within the current partial assignment.

As no polynomial time algorithm to find this is known, we content ourselves with a partial filtering that focuses on keeping the tightest lower bound on $\mathsf{totalCost}$ with a reasonable time complexity.

5.3.1 Soundness, Completeness and Idempotency

The filtering is sound (any pruned value is inconsistent with respect to the objective) but it does not achieve the classical notions of consistency arc consistency or bound consistency. Our focus was on practical performance rather than theoretical guarantees. Also, note that the propagation is not idempotent. For example, at the last step of Algorithm 1 (filter C) if a variable C_i is bound, we do not update the local counters and miss all further filtering arising from that if the propagation is not called again. A while loop in the propagator would solve this, but we found that such a loop reduces performance: intermediate propagation by lighter other constraints helps in practice.

5.3.2 Time Complexity for One Execution

For practical block modeling applications, the complexity of one execution of Algorithm 1 is linear in terms of the number of unbound vertices. Let us define three variables: Δ_{C}, the number of variables in C bound since the last call, u_{C}, the number of unbound variables in C, and k the number of clusters. The different steps of the algorithm have these complexities:

Step 1: updating local counters : $\mathcal{O}(\Delta_{\mathsf{C}}(u_{\mathsf{C}} + k))$

Step 2: filtering cost and totalCost: $\mathcal{O}(k^2 + u_{\mathsf{C}}k)$

Step 3: filtering M: $\mathcal{O}(k^2)$

Step 4: filtering C: $\mathcal{O}(u_{\mathsf{C}}k^2)$

In total for one execution of the filtering algorithm this yields $\mathcal{O}(\Delta_{\mathsf{C}}(u_{\mathsf{C}} + k) + k^2 + u_{\mathsf{C}}k + k^2 + u_{\mathsf{C}}k^2) = \mathcal{O}(\Delta_{\mathsf{C}}u_{\mathsf{C}} + \Delta_{\mathsf{C}}k + u_{\mathsf{C}}k^2)$. The value Δ_{C} is assumed to be small between consecutive calls of the filtering algorithm, and the number of clusters k is typically small (10 at most) in block modeling applications, so we consider the complexity to be $\mathcal{O}(u_{\mathsf{C}})$.

5.3.3 Time Complexity Along a Branch

We will now consider the time complexity to reach the first solution from the root of the search tree. We consider the worst case, i.e. there is no additional constraint on the variables, and no constraint on the cost of the solution. We start from the root — all C and M variables unbound — and assign a value to the variables one by one.

Let's assign first the n variables in C, then the k^2 variables of M. For the first n variables, $\Delta_{\mathsf{C}} = 1$ and u_{C} decreases from $n-1$ to 0, giving a complexity at each search node of $\mathcal{O}(nk^2)$. For the last k^2 nodes of the search tree, $\Delta_{\mathsf{C}} = 0 = u_{\mathsf{C}}$, so the complexity is $\mathcal{O}(k^2)$. This gives a complexity along the branch of $\mathcal{O}(n^2k^2 + k^4)$.

6. Search Procedure for Block Modeling

In constraint programming, the formulation of the problem is kept separate from the search procedure. The most commonly used search procedure is a branch and bound depth-first-search. Two important components of this search procedure are the variable and value ordering heuristics. These should permit discovering good incumbent solutions rapidly in order to prune the search tree. Since the problem also exhibits value symmetries, we use a dynamic symmetry breaking scheme during the search.

In this section, we focus first on the search for exact solutions to the block modeling problem. We present our variable and value ordering heuristics. Then, we explore the symmetries of the search space. When the search space becomes too large, and there is no hope to explore completely the search tree, LNS (Large Neighborhood Search) (Shaw, 1998) can be used on top of CP to diversify the search and discover good solutions rapidly. This is studied in Section 6.2.

6.1 Search Procedure for Exact Solutions

In this subsection, we present the design choices made for the CP search procedure when solving the block modeling problem to optimality.

6.1.1 Variable and Value Ordering Heuristic

When arriving at a branching point in the search, the CP solver must decide which variable to branch on and what value to try first. These decisions are called *variable ordering* and *value ordering*. Selecting the right ordering for the problem can significantly improve the efficiency of the solver.

For the CP model presented here, there are two sets of variables we can branch on (C and M). Since the `blockModelCost` constraint filters mostly based on the vertices which have been bound, it is better to branch on those before branching on M variables. The ordering of the C variables can further be refined with modern first-fail learning heuristics (Gay et al., 2015; Hebrard & Siala, 2017; Michel & Hentenryck, 2012). Specifically, we used Max Weighted Degree ordering, as introduced by Boussemart et al. (2004).

A good value heuristic for the clusters can also be constructed from our global constraint. We calculate $\underline{\delta_{i\mapsto x}}(c,d)$, a lower bound on the added cost of assigning vertex i to cluster x, so a good heuristic is to branch first on $\mathsf{C}_i = \operatorname{argmin}_x \sum_{c,d} \underline{\delta_{i\mapsto x}}(c,d)$, i.e., branch first on the value for which we expect the least increase in cost, thus the most likely to lead to an optimal solution. Similarly, for M, we branch first on $\mathsf{M}_{cd} = 0$ if $\texttt{nb1Block}_{cd} < \texttt{nb0Block}_{cd}$, and $\mathsf{M}_{cd} = 1$ otherwise.

6.1.2 Symmetry Breaking for the Block Modeling Problem

Symmetry breaking permits to drastically reduce the search. Symmetries can generally be avoided by adding constraints to the model. Unfortunately, this approach suffers from a bad interaction with the search as good solutions that were discovered early may become unfeasible because of the symmetry breaking constraints (Van Hentenryck & Michel, 2008). Therefore, a dynamic symmetry breaking during search strategy is generally more efficient. At every stage of the search, all-but one child nodes leading to symmetrical states are discarded.

The search space for this CP formulation of the block modeling problem has a number of symmetries. Firstly, it is clear that as long as the clusters stay the same, their labels can be changed — i.e. for any permutation $\sigma : \{0..k\} \to \{0..k\}$ and any state $S = (\mathsf{C}, \mathsf{M})$, the permuted state $\sigma(S) = (\sigma(\mathsf{C}_*), \mathsf{M}_{\sigma(*)\sigma(*)})$ is symmetrical to S. If σ' is an automorphism of the graph X, then $S' = (\mathsf{C}_{\sigma'(*)}, \mathsf{M}_{**})$ is symmetrical to S. Finally, if σ'' is an automorphism of the graph M, then $S'' = (\mathsf{C}_*, \mathsf{M}_{\sigma''(*)\sigma''(*)})$ has the same error as S.

In our CP model, we are only concerned with the first kind of symmetries (permutations of the cluster labels); those are easier to break. The dynamic symmetry-breaking scheme is: when branching on a C_i variable, the solver explores branches $\mathsf{C}_i = 1, \mathsf{C}_i = 2, \ldots, \mathsf{C}_i = m+1$ where m is the largest value bound to a C variable $m = \max\{v \mid \exists i : \mathsf{C}_i = \{v\}\}$.

Breaking the symmetries on the graph automorphisms of X and M is much more complicated and has not been considered for this paper. It is nonetheless an interesting direction for further work on this problem. For a related treatment of symmetry breaking of graph automorphisms, see (Zampelli et al., 2006).

6.2 Local Search

Large Neighborhood Search (LNS) (Shaw, 1998) is a way of doing local search for better solutions using constraint programming. Given an initial solution, LNS will fix part of the variables to the values in the solution and search for new assignments to the remaining variables with the CP solver. If a better solution is found, it serves as the new start for the LNS procedure. If no new solution is found this way, the variables are fixed back to their value in the solution, and a new set of variables is chosen to be relaxed. This way, we can explore large parts of the neighborhood of our current best solution.

LNS can be used to find block models of good quality when the graph size makes complete search prohibitively slow. This approach provides a local search algorithm which retains the extensibility of our method, since arbitrary constraints can still be added to the CP model used to explore neighborhoods of the initial solution.

The pseudocode for our LNS procedure is given in Algorithm 2. For LNS, the conventional wisdom is that if a better solution is in the neighborhood, it will be found quickly. We avoid spending too much time exploring unpromising neighborhoods by putting a limit f_{nodes} on the number of failed states the solver can encounter before giving up the search. Following this, we can also dynamically adjust the size of the explored neighborhoods. If, after relaxing $\alpha\%$ of the variables, we can still explore the entire neighborhood with less than f_{nodes} fails, then we can safely expand the next neighborhoods by picking a larger α, and conversely. For the stopping criterion, we opted for a stability criterion: we stop the search after f_{runs} consecutive runs for which no improvement has been made. The method is stochastic, so we restart f_{restart} times and keep the best solution out of those.

We guide the search by relaxing with a higher probability the variables which are responsible for a larger part of the cost of the solution. For every vertex i, we can calculate its contribution to the cost of the block model c_i by counting the number of entries in its row or column which are different from what is predicted by the block model $c_i = \sum_j \left|X_{ij} - \mathsf{M}_{\mathsf{C}_i\mathsf{C}_j}\right| + \left|X_{ji} - \mathsf{M}_{\mathsf{C}_j\mathsf{C}_i}\right|$. We weight the probabilities of selecting each vertex i to be relaxed proportionally to c_i, in such a way that the expected number of selected variables is still $\alpha\%$ of n. We also keep a small probability $(\alpha/10)\%$ of selecting a variable whose c_i is 0.

The LNS procedure needs a first solution as its starting point. To ensure that this initial solution is compatible with additional constraints, if any, we find it by running the CP model and stopping at the first solution which satisfies all constraints. The code for the complete LNS algorithm is available online at `https://github.com/gadevoi/blockmodel-cp/blob/master/src/main/scala/blockmodel/executables/RunLNS.scala`.

Algorithm 2 LNS for block modeling.

```
1:  Given:
2:    a procedure $I()$ to get an initial solution
3:    initial relaxation factor $\alpha$
4:    max number of failed states $f_{\text{nodes}}$
5:    max number of consecutive failed runs $f_{\text{runs}}$
6:    number of restarts $f_{\text{restart}}$
7:  for $i$ from 0 to $f_{\text{restart}}$ do
8:    $b^* = I()$
9:    $n_{\text{runs}} = 0$
10:   while $n_{\text{runs}} < f_{\text{runs}}$ do
11:     select $\alpha\%$ of variables to be relaxed according to their cost in the current best block
        model, and fix all other variables to their value in the current best solution and
        run the CP solver with a limit of $f_{\text{nodes}}$ of failed states
12:     $n_{\text{runs}}$ += 1
13:     if A better solution $b'$ is found then
14:       $b^* = b'$
15:       $n_{\text{runs}} = 0$
16:     end if
17:     update $\alpha$
18:   end while
19:   if $b^*$ is better than the last best then save it end if
20: end for
```

7. Finding the Optimal Number Of Clusters with MDL

One of the main limitations of the block modeling approach described so far is that it assumes that the number of clusters is known in advance. Unfortunately, in most practical cases, this is number is completely unknown. Ideally, we would like to infer this number from the data in a principled way. Simply selecting the model with the lowest cost is not a viable approach, since we can always trivially have a cost of 0 by putting each vertex in its own cluster. We need to find a balance between fitting the data (lowering the cost) and keeping the model size small. One approach to do exactly this stems from the Minimum Description Length (MDL) principle, which states that the best model is the one which most compresses the data (Rissanen, 1978; Grünwald, 2007); which is to say, which model allows us to transmit the information of our graph in the least number of bits.

In this section, we explore this approach. First, we derive a formula for the description length of a graph given a block model that describes it. In the experiments, we show that this formula works in practice by successfully recovering the generating block model of synthetic datasets. Finally, we show how our CP implementation of the block modeling

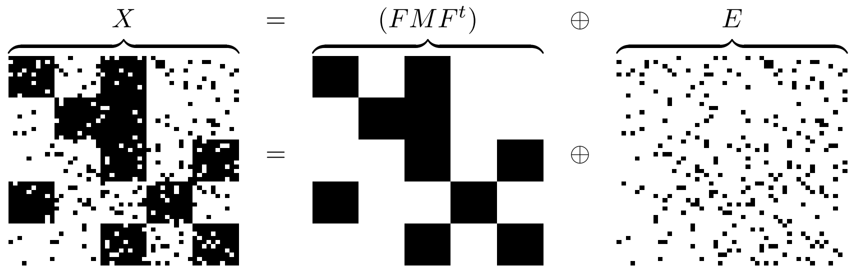

Figure 4: The adjacency matrix X reconstructed by the exclusive or between the block model (FMF^t) and the error E.

search can be efficiently used to search for the best block model according to the MDL criterion.

7.1 Description Length of a Block Model

Given a set of models $\mathcal{H}$, the best model $H \in \mathcal{H}$ is the one that minimizes

$$\mathcal{L}(H) + \mathcal{L}(D|H)$$

in which $\mathcal{L}(H)$ is the length, in bits, of the description of H and $\mathcal{L}(D|H)$ is the length, in bits, of the description of the data when encoded with H. This is called *two-part MDL* or *crude MDL*, as opposed to *refined MDL* where the two parts are encoded together (Grünwald, 2007).

MDL requires the compression to be *lossless* in order to allow for fair comparison between different $H \in \mathcal{H}$. Regardless of whether the block model we encode is perfect, we need to be able to reconstruct the adjacency matrix X without loss. We do this explicitly by transmitting both the block model and the difference between the original data X and its approximation by the block model FMF^t. That is, we define an $n \times n$ matrix E to be a boolean matrix such that

$$E = X \oplus (FMF^t), \tag{4}$$

with $\oplus$ being the elementwise *exclusive or* operation, that is the matrix sum with the sum defined as $1 + 1 = 0$ (i.e. addition done over the field $\mathbb{Z}_2$). This is shown in Figure 4.

We can define the total compressed size $\mathcal{L}(X, H)$ in bits, for an adjacency matrix X and a block model $H = (F, M)$ with $H \in \mathcal{H}$, as

$$\mathcal{L}(X, H) = \mathcal{L}(H) + \mathcal{L}(E), \tag{5}$$

where E follows from X and H using Equation (4). Following the MDL principle, the best block model for X is found by minimizing Equation (5).

We define first the number of bits needed to encode the block model $H = (F, M)$ of dimensions $n \times k$ and $k \times k$ for F and M respectively, as

$$\mathcal{L}(H) = \mathcal{L}_{\mathbb{N}}(n) + \mathcal{L}(k) + \mathcal{L}(F) + \mathcal{L}(M). \tag{6}$$

That is, we encode the dimensions n and k, and then the contents of the two matrices. By explicitly encoding the dimensions of the matrices, we can subsequently encode matrices F and M using optimal codes (Cover & Thomas, 2006):

$$\mathcal{L}_{\mathbb{N}}(n) = \log^*(n) + log(c_0), \tag{7}$$

where $\log^*(n) = \log(n) + \log\log(n) + \ldots$ and only the positive terms are included in the sum. To make $\mathcal{L}_{\mathbb{N}}$ a valid encoding, c_0 is chosen as $c_0 = \sum_{j\geq 1} 2^{-\mathcal{L}_{\mathbb{N}}(j)} \approx 2.865064$ such that the Kraft inequality is satisfied—that is, ensure that this is a valid encoding by having all probabilities sum up to 1.

To avoid bias towards certain model sizes, k is encoded with a fixed number of bits—this is called *block encoding* in the MDL literature—which gives us

$$\mathcal{L}(k) = \log(n). \tag{8}$$

We use n because larger values of k do not make sense. A model with n clusters can describe the data perfectly by putting each vertex in its own cluster, and $X = M$.

The coding scheme for the cluster assignment matrix F is devised as follows: for each vertex, we have to transmit the index of the cluster in which it has been placed. As there are n vertices and k clusters, the length of the code is

$$\mathcal{L}(F) = n\log(k). \tag{9}$$

We do not want to bias the search towards any particular structure in M, so we use a block encoding for it. As it is a $k \times k$ binary matrix, we can encode it with

$$\mathcal{L}(M) = k^2 \tag{10}$$

bits. This choice highlights an important assumption for our coding scheme: for a given number of clusters k, all image graphs M are equally likely. Other choices could be made, for example assuming that sparse M are more likely, but since the strength of block modeling compared to other graph clustering approaches is the richness of the information in the image matrix, it was chosen to give the same weight to "complex" image matrices as to simple ones.

For the coding scheme of E, we have to transmit the positions of the $|E|$ ones in the matrix. There are $\binom{n^2}{|E|}$ possible ways to position the ones in the matrix, so the coding length is

$$\mathcal{L}(E) = \log(n^2) + \log\binom{n^2}{|E|}. \tag{11}$$

Wrapping it all up, the final equation giving the coding length for an adjacency matrix X given a block model H is shown in Equation (12). Note that we do not have to actually encode the data to know the length of the encoded message. Knowing the size k and the number of errors $|E|$ is sufficient to calculate $\mathcal{L}(X, H)$:

$$\mathcal{L}(X,H) = \overbrace{\mathcal{L}_{\mathbb{N}}(n) + \log(n) + n\log(k) + k^2}^{\mathcal{L}(H)} + \overbrace{\log(n^2) + \log\binom{n^2}{|E|}}^{\mathcal{L}(E)}. \tag{12}$$

In the experiment of Section 8.5, we validate empirically the applicably of this equation by successfully retrieving the block model of synthetic graphs to which varying levels of noise have been applied.

7.2 CP Search for MDL

For a given graph X, the description length as defined before is

$$\mathcal{L}(X, H) = \mathcal{L}_{\mathbb{N}}(n) + \log(n) + n\log(k) + k^2 + \log(n^2) + \log\binom{n^2}{|E|}.$$

As the number of vertices n is fixed, this value depends only on the number of clusters k and the cost of the block model $|E|$. We can write it as $\mathcal{L}_X(k, |E|) = \mathcal{L}(X, H)$. Furthermore, for a fixed k and $|E| < n^2/2$, finding the model which minimizes $\mathcal{L}_X$ is equivalent to finding the model which minimizes $|E|$. Thus, for a fixed k, we can use our CP model to find the block model which minimizes $|E| = \mathsf{totalCost}$ to find the best block model for the MDL criterion.

This only works for a fixed k, so we repeat the search for every value of k between 1 and some arbitrary maximum number of clusters $k_{\max}$. From the $k_{\max}$ models found this way, we select the one with the lowest description length $\mathcal{L}_X$.

This procedure can be further optimized by remarking the following: if we know the best description length $l^* = \mathcal{L}_X(k-1, |E|)$ achievable for a number of clusters $k-1$, then we can calculate an upper bound on the cost $|E'|$ that a model of size k should have to have a smaller length $\mathcal{L}_X(k, |E'|) < l^*$. Specifically,

$$\mathcal{L}_X(k, |E'|) < \mathcal{L}_X(k-1, |E|) \iff |E'| \le \max\left\{e \in \left\{0..\lceil n^2/2\rceil\right\} \mid \mathcal{L}_X(k, e) < l^*\right\}.$$

The pseudocode for the search procedure using this upper bound is shown in Algorithm 3. For the search for the upper bound at line 6, we precalculate a table with the value of $\log\binom{n^2}{e}$ for $e \in \left\{0..\lceil n^2/2\rceil\right\}$, for which an incremental formula exists[1], and then perform a binary search over the table to find the max value of e respecting the condition $\mathcal{L}_X(k, e) = \mathcal{L}_{\mathbb{N}}(n) + \log(n) + n\log(k) + k^2 + \log(n^2) + \log\binom{n^2}{e} \le l^*$.

7.3 Combining MDL and LNS

The local search procedure described in Section 6.2 can be adapted to perform local search on the MDL block modeling problem. The idea is to substitute the CP solver in the procedure described in last section by a large neighborhood search. We stop the LNS for a given k when the stability criterion is met, then try improving the solution with one additional cluster. The initial solution for $k+1$ is simply the best block model found for k augmented with an empty cluster.

We ran this LNS for MDL on real world datasets[2]. The results are reported in Table 3. For each graph, we show the number of vertices n. We ran our MDL search with LNS for 15 minutes, with a maximum number of 100 clusters. The LNS procedure was run with initial relaxation factor $\alpha = 5\%$, max number of failed states $f_{\text{nodes}} = 1000$, and max number of

1. Equation 11 in `https://mathworld.wolfram.com/BinomialCoefficient.html`.
2. Retrieved from `http://www-personal.umich.edu/~mejn/netdata`

Algorithm 3 Search for the number of clusters k.

```
1: /* $l^*$ will store the best description length found so far */
2: $l^* \leftarrow +\infty$
3: /* $k^*$ will store the best $k$ found so far */
4: $k^* \leftarrow 0$
5: for all $k$ from 1 to $k_{\max}$ do
6:    ub $\leftarrow \max\left\{e \in \left\{0..\lceil n^2/2 \rceil\right\} \mid \mathcal{L}_X(k, e) < l^*\right\}$
7:    find the block model $H$ minimizing totalCost with the constraint totalCost $\leq$ ub
8:    if a solution is found then
9:       $k^* \leftarrow k$
10:      $l^* \leftarrow \mathcal{L}_X(k, \mathsf{totalCost})$
11:   end if
12: end for
```

Network	n	k	mdl [bits]	time [s]
Zachary's karate club (Zachary, 1977)	34	7	540.363	263
Dolphin social network (Lusseau et al., 2003)	62	6	1488.789	21
Characters in Les Misérables (Knuth, 1993)	77	10	1606.748	834
Political Books [3]	105	10	4035.252	22
American College Football (Girvan & Newman, 2002)	115	14	4848.725	48
C. elegans neural network (Watts & Strogatz, 1998)	297	7	15083.928	62
Political blogs (Adamic & Glance, 2005)	1222	4	153218.081	718

Table 3: Results of the LNS procedure for MDL on real-wold datasets.

consecutive failed runs $f_{\text{runs}} = 100$. We report the best solution with its number of clusters k, the description length (in bits) and the time our solver took to reach this solution.

As noted by Peixoto (2017b), there is no "one true" block model for real-world networks, but rather we should consider all block models with similar MDL score as equally valid descriptions of the dataset. We can still note that we get a number of clusters close to expected for most of the datasets. Notably, the American College Football datasets contains the network of American football games between Division IA colleges during regular season Fall 2000, which is composed of 12 congregation. In the block model returned by our method, we find those congregations, with two of them having been split up in half.

8. Experiments

In this section, we validate empirically the performance of our different algorihtms. Firt, we compare our CP system to an existing mixed integer programming approach, and show that we outperform it. Then, we show the scalability of our exact search procedure on synthetic graphs of varying size. For our large neighbourhood approach, we compare it to an existing approach bundled in the software package Pajek, then we show its scalability on synthetic graphs up to 7000 vertices. Finally, we run experiments on our MDL appraoch, first to validate our coding scheme, then to show the efficiency of our upper bound.

8.1 Comparison of Exact Search with MIP Model

The block modeling problem is often approximated using heuristic search. However, an approach to find the optimal solution is proposed by Dabkowski et al. (2016). It builds on the work of (Brusco & Steinley, 2009), which defines a MIP model to find the optimal partition given a fixed image matrix M. We expand this approach to find the optimal solution by generating a minimal, representative set of image matrices of size k—i.e. keeping only one matrix for each equivalence class of graphs under graph isomorphism—and running the MIP solver for each matrix in this set.

The MIP formulation is as follows. For every representative image matrix M, solve

$$\min \sum_{c,d=1}^{k} \sum_{i,j=1}^{n} f_{ic} f_{jd} (M_{cd} + X_{ij} - 2M_{cd}X_{ij}) \tag{13}$$

$$\text{s.t.} \sum_{c=1}^{k} f_{ic} = 1 \qquad \forall i \in \{1..n\} \tag{14}$$

$$\sum_{i=1}^{n} f_{ic} \geq 1 \qquad \forall c \in \{1..k\} \tag{15}$$

$$f_{ic} \in \{0,1\} \qquad \forall i \in \{1..n\}, c \in \{1..k\}, \tag{16}$$

where $f_{ic} \in \{0,1\}$ are the decision variables, $f_{ic} = 1$ indicating that vertex i is in cluster c and $f_{ic} = 0$ otherwise. Equation (14) ensures that each vertex is in one cluster only, and Equation (15) ensures that no cluster is empty.

Equation (13) has a multiplication, but it can be linearized by substituting the product $f_{ic}f_{jd}$ by a variable y_{ijcd} and adding the constraints:

$$\begin{aligned}
y_{ijcd} &\leq f_{ic} && \forall i,j \in \{1..n\}, c,d \in \{1..k\} \\
y_{ijcd} &\leq f_{jd} && \forall i,j \in \{1..n\}, c,d \in \{1..k\} \\
y_{ijcd} &\geq f_{ic} + f_{jd} - 1 && \forall i,j \in \{1..n\}, c,d \in \{1..k\} \\
y_{ijcd} &\geq 0 && \forall i,j \in \{1..n\}, c,d \in \{1..k\}
\end{aligned}$$

In this section, we compare the performance of the CP approach with this MIP approach. As both give exact solutions, the quality of the solutions are identical, and we only need to compare the running time. In order to evaluate the performance of our global constraint, we wrote three CP models. The first is used as a baseline. It follows the mathematical formulation of the problem, and uses our symmetry-breaking scheme[4]. The second uses our global constraint for filtering with the same search procedure. The third uses our global constraint with the value ordering heuristic. The MIP model and the 3 CP approaches are compared on four small well-studied social networks, published and analyzed in depth by Doreian et al. (2005, Chapters 2, 6), namely: (a) the Transatlantic Industries little league baseball team network (Fine, 1987), (b) the Sharpstone little league baseball team network

4. The code can be found on `https://github.com/gadevoi/blockmodel-cp/blob/master/src/main/scala/blockmodel/BaselineBlockmodelCPModel.scala`

Table 4: Run time of the MIP approach compared to a baseline CP approach (CP(bsl)), a CP approach with our global constraint (CP(our)), and our constraint + our value heuristic (+heuris.) for different number of clusters k. "–" indicates a timeout after 2 hours.

			CPU time (s)			
dataset	n	k	MIP	CP(bsl)	CP(our)	+heuris.
Transatlantic	13	2	1.73	0.80	0.45	0.28
		3	142.25	21.15	0.88	0.79
		4	–	386.20	2.94	2.07
		5	–	3602.69	1.87	
Sharpstone	13	2	1.24	0.50	0.44	0.19
		3	62.46	13.57	1.17	0.85
		4	2952.13	221.41	2.78	1.82
		5	–	1102.68	2.31	1.30
Political Actor	14	2	2.14	1.13	0.62	0.31
		3	155.90	60.15	1.32	0.89
		4	2178.42	1936.43	2.68	2.20
		5	–	–	2.93	2.25
Search And Rescue	20	2	13.31	22.04	0.85	0.48
		3	–	–	6.01	5.18
		4	–	–	14.22	
		5	–	–	52.96	

(Fine, 1987), (c) the political actor network (PA) (Doreian & Albert, 1989) and (d) the Kansas search and air rescue (SAR) network (Drabek et al., 1981).

The CP models were written and solved in OscaR (OscaR Team, 2012), and are available at `https://github.com/gadevoi/blockmodel-cp/blob/master/src/main/scala/blockmodel/executables/RunCPModel.scala`. The MIP model was written and solved in Java using Gurobi (Gurobi Optimization, 2018). All experiments were run on a computer with Xeon Platinum 8160 24c/48t HyperThread processors. The results are shown in Table 4.

We clearly observe that the MIP approach does not scale and is inapplicable for non-trivial sizes. The effect of our global constraint and our value heuristic are also evident, making the search orders of magnitude faster.

8.2 Scalability of the Exact Search

We now show the scalability of the complete search on larger instances. We generated artificial graphs with a known block model structure, and added 20% of noise with different seeds for the random number generator. This means that 20% of the entries of the adjacency matrix X were flipped from 1 to 0 and vice-versa. In Figure 7, we report the runtime until

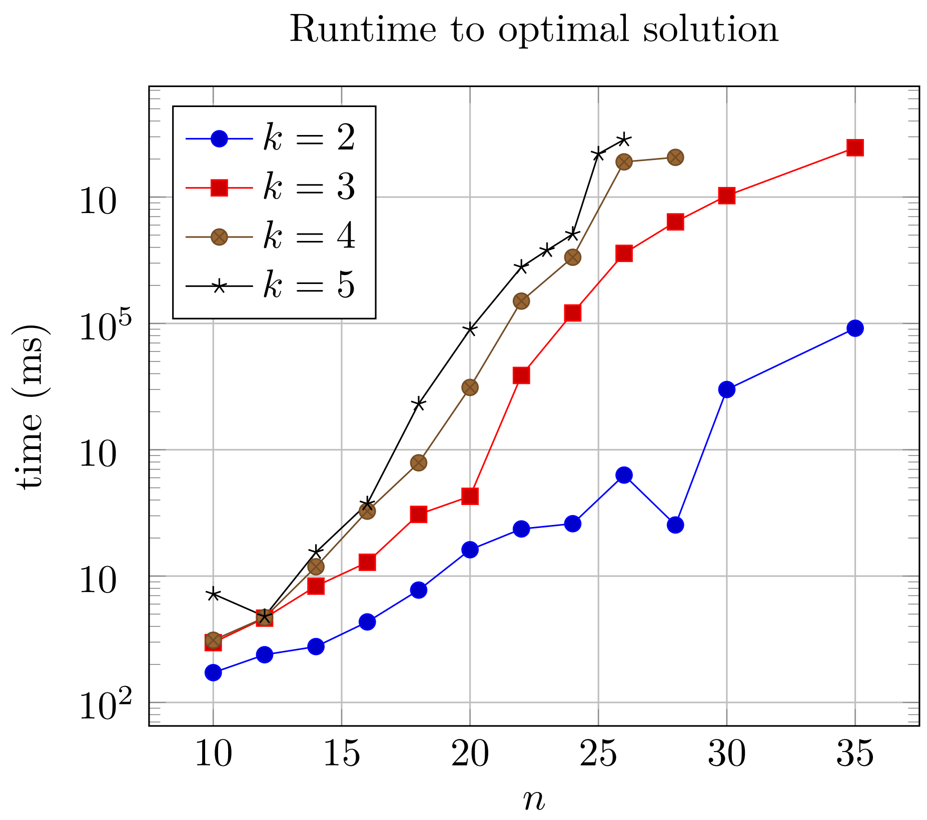

Figure 5: Scalability on graphs with known block model and 20% of noise, for varying number of clusters k. The graph shows runtime until the solution is proven optimal.

proving optimality for different sizes n and number of clusters k. We see that the execution time increases exponentially with the size n and number of clusters k, reflecting the explosion in the size of the search space.

8.3 Comparison of LNS with Local Search

In this subsection, we compare the performance of the LNS approach with a local search algorithm for block modeling (Batagelj et al., 1992a) bundled in the popular graph processing software Pajek[5].

We generated synthetic graphs with 50, 100, 150 and 200 vertices — the classical block modeling algorithm included in Pajek only supports graphs of less than 256 vertices — with a fixed block model structure of 5 clusters. We added 40% of noise to the data, then compared the evolution of the quality of the solution with time for both methods. The results are

5. `http://mrvar.fdv.uni-lj.si/pajek/`

shown in Figure 6. For all instances over 50 vertices, the LNS method outperformed Pajek's local search, in time and sometimes also in the quality of the solution.

The LNS procedure was run with initial relaxation factor $\alpha = 5\%$, max number of failed states $f_{\text{nodes}} = 1000$, max number of consecutive failed runs $f_{\text{runs}} = 100$, and number of restarts $f_{\text{restart}} = 10$. We can observe the impact of restarts that appear as sudden drops in the cost of the solution, for example for $n = 200$.

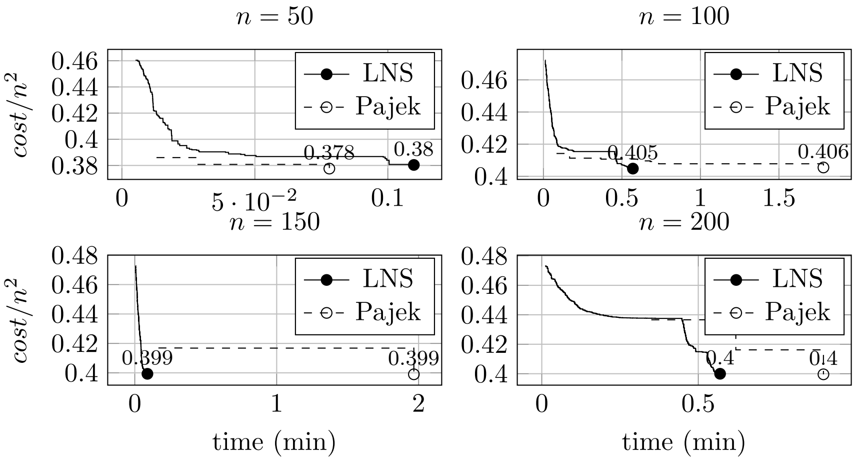

Figure 6: Comparison of LNS search with the local search bundled in Pajek (Batagelj et al., 1992a) for synthetic dataset with 40% of noise. Each graph shows a different instance of the problem, for $n = 50$ to $n = 200$ by increments of 50.

8.4 Scalability of LNS

We now show the scalability of the LNS method on larger instances. We once again generated synthetic graphs of different sizes n with a known block model structure with $k = 5$ and 20% of noise. In Figure 7, we plot the convergence of Large Neighborhood Search over 10 minutes, with initial relaxation factor $\alpha = 5\%$, max number of failed states $f_{\text{nodes}} = 125$, max number of consecutive failed runs $f_{\text{runs}} = 100$, and number of restarts $f_{\text{restart}} = 10$. The search is run 10 times for each graph, to account for the stochastic nature of the search. The y-axis shows the cost relative to the size of the graph. Since the $n \times n$ adjacency matrices were generated from perfect block models to which 20% of noise has been applied, we expect to find solutions with a cost $\approx 0.2n^2$. We see that in most cases, our LNS search converges quickly on a solution of optimal cost, even with thousands of vertices. We note that for two runs (one for $n = 3000$ and another for $n = 5000$), the search gets stuck in a local minimum and doesn't reach the optimal value of $0.2n^2$ in the 10 minutes. Note that all of these graphs are too large for Pajek's method, but were solved by our LNS search in a handful of minutes. A comparison of different α values can be found in the appendix.

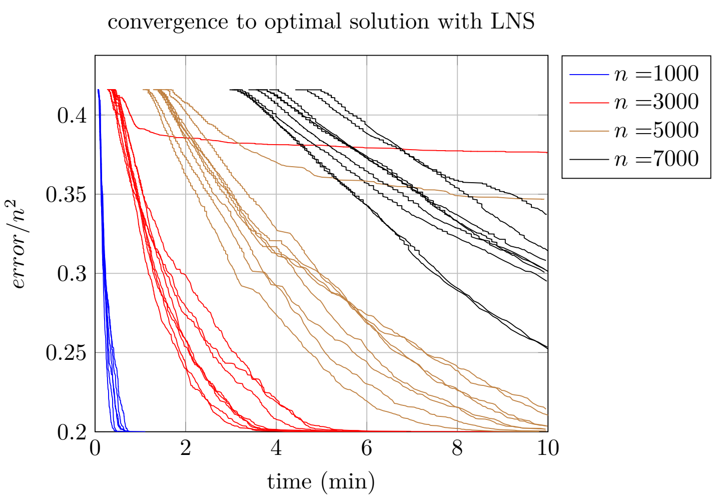

Figure 7: Scalability of LNS on synthetic graphs with known block model of $k = 5$ and 20% of noise. To account for the stochastic nature of the search, it is run 10 times for each graph starting from a different random seed.

8.5 Validation of the MDL Coding Scheme

The minimum description length principle gives us a principled way of comparing the quality of the fit of different block models to the data. Our aim is to use it to automatically determine the best number of clusters k, which is a compromise between the conciseness of the model (formalized by $\mathcal{L}(H)$) and how well the model fits the data ($\mathcal{L}(E)$). Given two block models H_1 and H_2, the "best" one in the MDL sense is the one that minimizes the description length $\mathcal{L}(X, H)$.

To validate that our coding scheme works in practice when used for this purpose, four types of synthetic block models with common structures have been chosen: *community* structure, *ring* structure, *star* structure and *stick* structure. They are exemplified in Figure 8. In the *community* structure, each cluster is fully connected (i.e. a clique). In the *ring* structure, each cluster is connected to a different cluster in such a way that they form a single cycle. In the *star* structure, every cluster is connected to a single central cluster. Note that for the star cluster, we made each cluster a clique, otherwise a star structure with $k > 2$ could be described just as well with a 2 cluster block model, with one cluster being the central cluster and all other clusters regrouped together. Finally, the *stick* structure is like a *ring* structure but with one edge missing.

For each of the 4 block model structures, we generated graphs with $n = 20$ vertices, $k_{\text{real}} = 5$ clusters, and varying levels of noise (0%, 5%, 10%, 15% and 20%). Then, for each generated graph and for each value $k \in \{1..10\}$, we find the block model that minimizes the description length $\mathcal{L}(X, H)$. Our coding scheme is useful if we are able to recover k_{real} as the value for which the coding length is minimal.

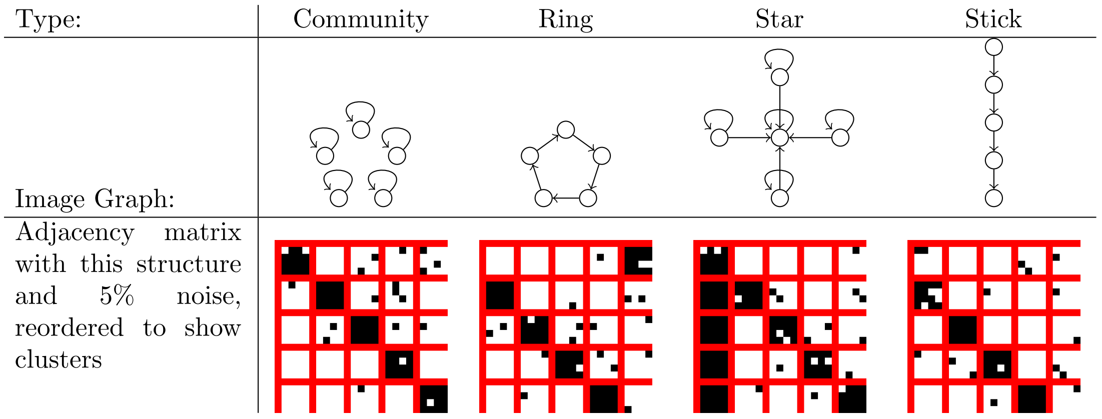
Type:
Community
Ring
Star
Stick
Image Graph:
Adjacency matrix with this structure and 5% noise, reordered to show clusters

Figure 8: Adjacency matrices for 4 different classic block model structures, with $k = 5$ clusters, $n = 20$ vertices and 5% random noise.

We show the results of the experiment in Figure 9. As we can see, minimizing the description length $\mathcal{L}(X, H)$ successfully recovers the correct number of clusters $k = 5$ for noise levels up to 15%. For 20% noise, it only recovers it for the ring and stick structure. At this high level of noise, the difference in $\mathcal{L}(X, H)$ between the various models is very small, which is interpreted as all models giving a description of the data of roughly the same quality with respect to their size.

8.6 Efficiency of the Upper Bound for CP Search for MDL

In Section 7.2, we presented a CP search for the optimal block model according to our MDL criterion. It uses an upper-bounding formula to speed up the search. The efficiency of this approach is shown in Figure 10. We ran the same experiments as in Figure 9 for 10% noise. For each k, we show the average runtime for the 4 community structures—community, ring, star, stick. We compare three different setups: (i) The baseline, marked with blue circles, does a complete search from scratch for every k. (ii) The red line with squares shows the improvement when for each step k we add the constraint that $\mathsf{totalCost}$ must be lower than the optimal $\mathsf{totalCost}$ value found at steps $1, \ldots, k-1$. (iii) The brown line in crossed dots shows the improvement of the procedure described in Algorithm 3. In the plot, vertical bars show the individual contribution of each step k.

The improvements of Algorithm 3 are clearly shown in this experiment, as the third line, showing the execution time with our algorithm, is much lower than the other two, and increases at a lower rate. We observe also that from a certain $k > 5$, the problem becomes unfeasible with the tightest upper bound, and our approach detects this in a couple of milliseconds.

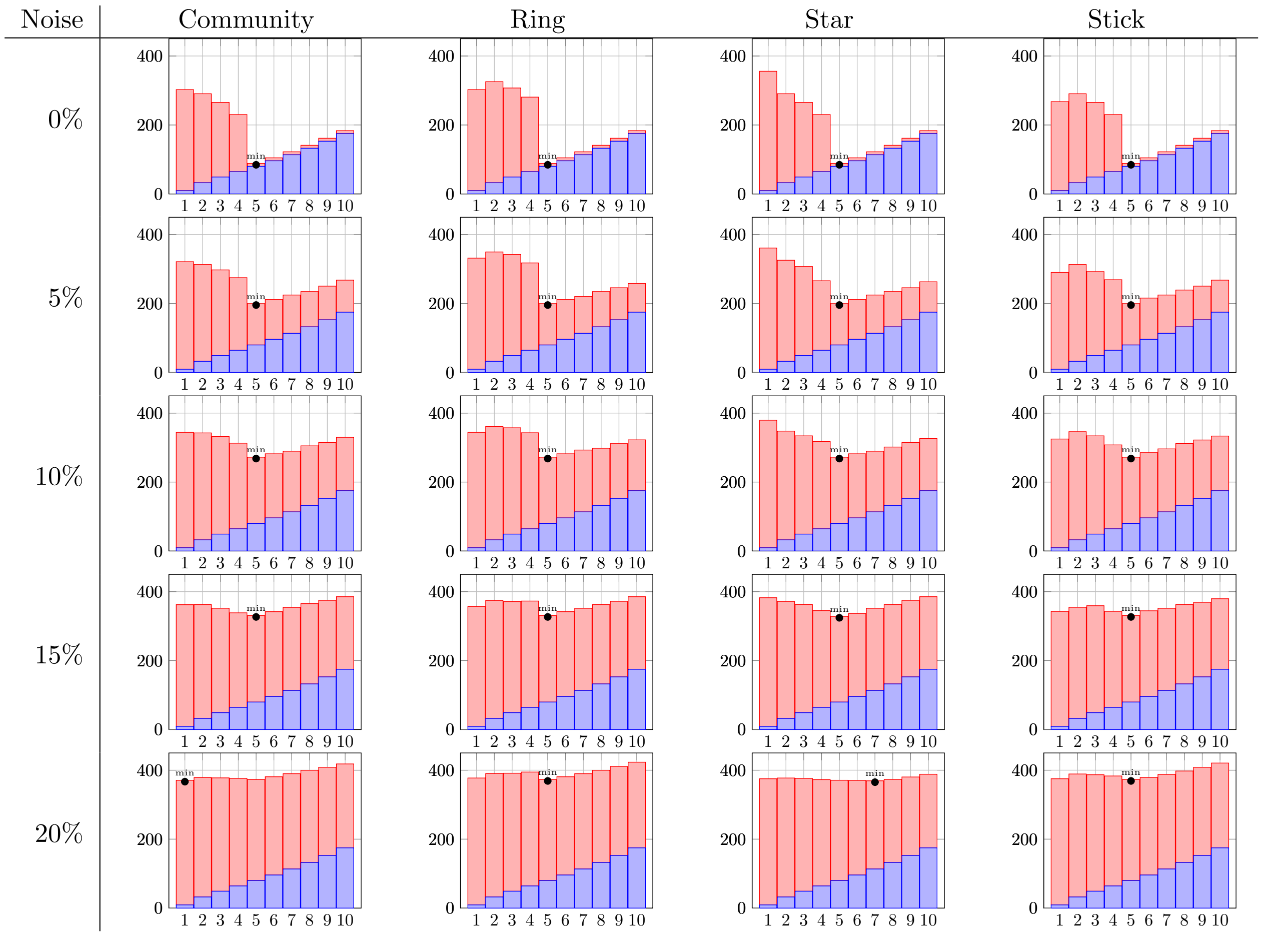

Figure 9: Description length, in bits with regards to the number of clusters k, of the best block model for graphs generated with a known block model structure of $k_{\text{real}} = 5$ clusters and random noise. The lower, blue bar is the description length of the model $\mathcal{L}(H)$ and the upper, red bar is the description length of the data $\mathcal{L}(E)$. They are stacked so that the total height of each bar is $\mathcal{L}(X,H) = \mathcal{L}(H) + \mathcal{L}(E)$. The bar with the global minimum description length is marked with a dot.

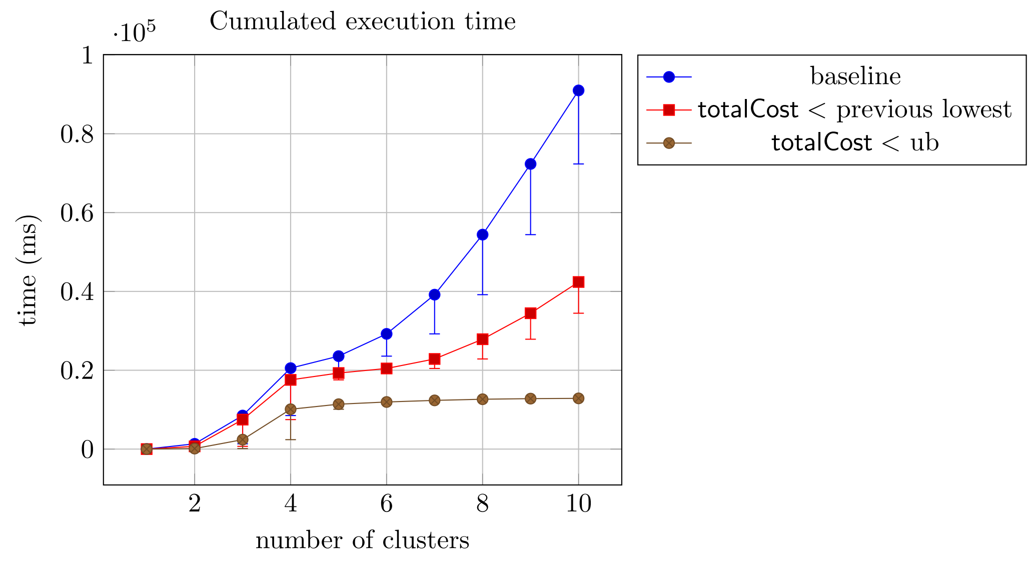

Figure 10: Average execution time wrt. k on artificial graphs with 10% noise. Vertical bars show the individual contribution of each step k. The baseline, on top marked with blue circles, does a complete search from scratch for every k. The red line in the middle with squares shows the improvement of constraining totalCost to be lower than the previous lowest. The lower brown line in crossed dots shows the improvement of the procedure described in Algorithm 3.

8.7 Beyond Traditional Block Modeling

A real strength of a constraint programming formulation is the ability to add complex constraints on the clusters or the image graph, to combine multiple instances of the same constraint, and to optimize any of the variables. As an illustration, we explore the use of block modeling on migration data in Europe. In the first illustration, Figure 11, we add the constraint that the clusters must be connected on the map — i.e. one can travel between any two countries of a cluster without leaving the cluster. This connectivity constraint is very complex to model in MIP but is an existing building block in CP (Prosser & Unsworth, 2006; Dooms, 2006; Bessiere et al., 2015). In the second illustration, Figure 12, we study a problem that involves multiple instances of our global constraint: we take the migration matrix at 5 different points in time. We build a block model for each year, with the constraint that the clusters are the same in all models. We have five block models, so we minimize the sum of their costs. This is similar to the non-negative RESCAL setting (Krompaß et al., 2013).

The migration graphs were built from an open dataset provided by the World Bank[6]. An edge $X_{ab} = 1$ indicates that the number of migrants born in a living in b is more than 0.01% of the population of a. The dataset was limited to countries in continental Europe,

6. The World Bank: Migration and Remittances Data, retrieved `http://www.worldbank.org/en/topic/migrationremittancesdiasporaissues/brief/migration-remittances-dat`

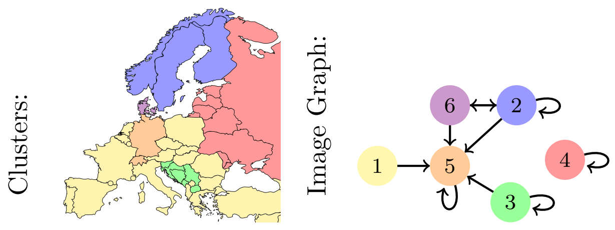

Figure 11: A block model of migrant stocks in continental Europe in 2015, with geographically connected clusters.

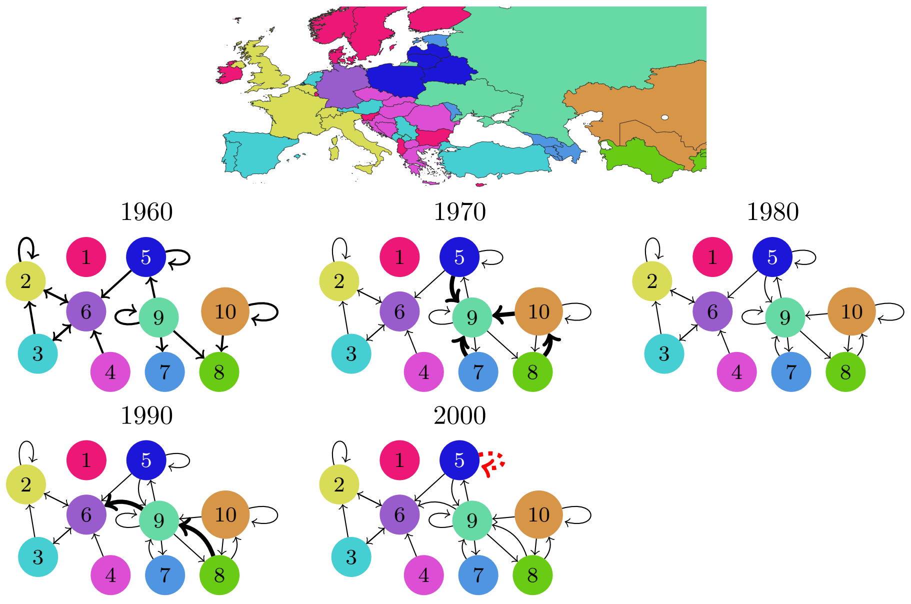

Figure 12: RESCAL model for the evolution of migrant stocks in Europe. The edges which appeared in a decade are rendered in thick stroke, and those which disappeared are in dotted red stroke.

excluding islands for the first illustration because of the connectivity constraint. The models were found after a Large Neighborhood Search of 10 minutes, with restarts after 1000 failed states relaxing 5% of the variables.

In Figure 11, we clearly see the ex-Soviet block appear in cluster 4, with mostly internal migration and not much migration to Western Europe. Germany and Switzerland appear as a core destination for migrants from most European countries. Denmark is the only member of its cluster, but it would have been in the same cluster as Fennoscandia if it did not violate the connectivity constraint. In Figure 12, we observe for example the migration of people from Russia to Germany in the nineties (thick arrow between 9 and 6), which we can probably link to the fall of the Iron Curtain.

9. Conclusion and Further Work

We have introduced a CP approach to the block modeling problem, using a dedicated global constraint. It has the advantage of being able to easily incorporate any combination of additional constraints, contrary to previous works. Our experiments show that our approach is orders of magnitude faster than competing solutions to find optimal block models.

The CP framework can also be used for local search. We implemented a LNS solver for the block modeling problem, and showed its performance compared to an existing local search for block modeling. Then, we showed the scalability of the method on synthetic graphs with thousands of vertices. Finally, we showed that the method is extendable by doing a RESCAL search on migration data.

Then, we introduced a coding scheme for the block modeling problem, and used it to derive an appropriate MDL criterion for model selection. We have shown that this criterion is well-adapted to the problem and allows recovering the correct model size for graphs with up to 20% noise. We have adapted our solver to find the best model size k according to this criterion, and shown its efficiency. We also adapted the local search procedure for the MDL setting, and showed its application on real-world networks.

This work could be further expanded with an equivalent global constraint for regular equivalence, or for generalized block modeling (Doreian et al., 2005), or for degree-corrected block modeling (Karrer & Newman, 2011), which all give different definitions for what constitutes a good cluster. The search could be accelerated by breaking symmetries on the automorphisms of X and M and considering more advanced variable and value ordering schemes.

There is room for improvement on the Large Neighborhood Search. A more exhaustive exploration of the different meta-parameters for the search (maximum number of failures, relaxation factor, ...) could be carried out. Different neighborhoods moves could be considered, such as relaxing one cluster at a time. The best approach for any given graph could be found using techniques such as Adaptative LNS (Ropke & Pisinger, 2006).

The work on MDL could be expanded by devising a LNS approach to find simultaneously the best model and model size. The search for the best model size could be sped up by parallelization, sharing information between parallel searches to improve their upper bound on the cost.

Acknowledgements

Alex Lucía Mattenet was funded by a FRIA grant by the Belgian Fond National de Recherche Scientifique.

Ian Davidson was partially funded by NSF IIS:1910306, Explaining Unsupervised Learning: Combinatorial Optimization Formulations, Methods and Applications. Part of this work was completed whilst he was an EU Collegium de Lyon Scholar.

Appendix A. Comparison of α values for LNS

In Figure 13, show the impact of the parameter choice for the initial relaxation factor α for the LNS search on the experiment in Figure 7. For the same 4 synthetic graphs of 1000,

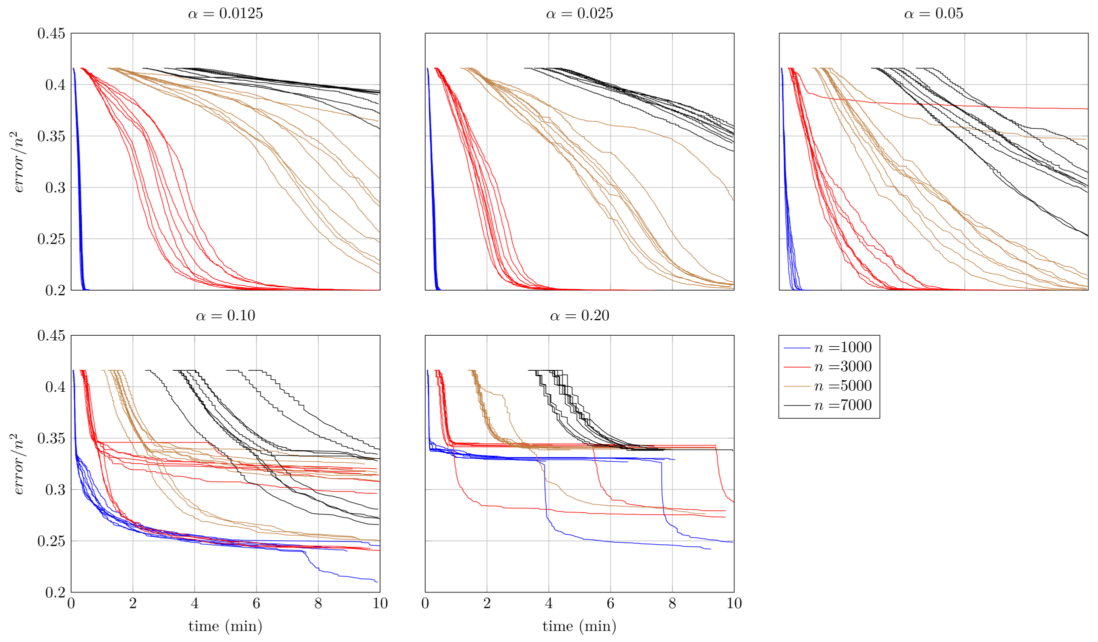

Figure 13: Effect of the α parameter on the convergence of LNS search on synthetic graphs with $k = 5$ and 20% of noise.

3000, 5000 and 7000 vertices respectively, we run our LNS search procedure for different values of α. For all runs, the parameters are the same as in Figure 7: max number of failed states $f_{\text{nodes}} = 125$, max number of consecutive failed runs $f_{\text{runs}} = 100$, and number of restarts $f_{\text{restart}} = 10$. The search is run 10 times for each graph, to account for the stochastic nature of the search.

We can observe that for $\alpha = 0.10$ and $\alpha = 0.20$, the search gets stuck in local minima, and does not reach the optimal value of $0.2n^2$. For the smaller values $\alpha = 0.025$ and $\alpha = 0.0125$, the performance is comparable to that of $\alpha = 0.5$ for $n = 1000$ and $n = 3000$ (the blue and red lines), but it converges slower for the larger graphs $n = 5000$ and 7000. Thus, we chose 0.05 as an overall best choice for the initial relaxation factor.

References

Adamic, L. A., & Glance, N. (2005). The political blogosphere and the 2004 US election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, pp. 36–43.

Bai, Z., Qian, B., & Davidson, I. (2018). Discovering Models from Structural and Behavioral Brain Imaging Data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1128–1137. ACM.

Bai, Z., Walker, P., Tschiffely, A., Wang, F., & Davidson, I. (2017). Unsupervised network discovery for brain imaging data. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 55–64. ACM.

Batagelj, V. (1997). Notes on blockmodeling. *Social Networks*, *19*(2), 143–155.

Batagelj, V., Ferligoj, A., & Doreian, P. (1992a). Direct and indirect methods for structural equivalence. *Social networks*, *14*(1-2), 63.

Batagelj, V., Ferligoj, A., & Doreian, P. (1992b). Direct and indirect methods for structural equivalence. *Social Networks*, *14*(1), 63–90.

Bessiere, C., Hebrard, E., Katsirelos, G., & Walsh, T. (2015). Reasoning about connectivity constraints. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Bessiere, C., & Van Hentenryck, P. (2003). To be or not to be... a global constraint. In *International conference on principles and practice of constraint programming*, pp. 789–794. Springer.

Boussemart, F., Hemery, F., Lecoutre, C., & Sais, L. (2004). Boosting systematic search by weighting constraints. In de Mántaras, R. L., & Saitta, L. (Eds.), *PAIS 2004, Valencia, Spain, August 22-27, 2004*, pp. 146–150. IOS Press.

Brusco, M. J., & Steinley, D. (2009). Integer programs for one- and two-mode blockmodeling based on prespecified image matrices for structural and regular equivalence. *Journal of Mathematical Psychology*, *53*(6), 577–585.

Cover, T. M., & Thomas, J. A. (2006). *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, New York, NY, USA.

Dabkowski, M., Fan, N., & Breiger, R. (2016). Exploratory blockmodeling for one-mode, unsigned, deterministic networks using integer programming and structural equivalence. *Social Networks*, *47*, 93–106.

Dao, T.-B.-H., Vrain, C., Duong, K.-C., & Davidson, I. (2016). A Framework for Actionable Clustering using Constraint Programming. In *22nd European Conference on Artificial Intelligence*, 22nd European Conference on Artificial Intelligence, The Hague, Netherlands.

Dooms, G. (2006). *The CP(Graph) computation domain in constraint programming*. Ph.D. thesis, UCL - Université Catholique de Louvain.

Doreian, P., & Albert, L. H. (1989). Partitioning Political Actor Networks: Some Quantitative Tools for Analyzing Qualitative Networks. *Journal of Quantitative Anthropology*, *1*, 279–291.

Doreian, P., Batagelj, V., & Ferligoj, A. (2005). *Generalized Blockmodeling*. Cambridge University Press.

Drabek, T., Tamminga, H., Kilijanek, T., & Adams, C. (1981). Managing multi-organizational emergency responses. *Boulder: University of Colorado, Institute of Behavioral Science*.

Fine, G. A. (1987). *With the Boys: Little League Baseball and Preadolescent Culture*. University of Chicago Press. Google-Books-ID: 2qWgZPuNjEYC.

Ganji, M., Chan, J., Stuckey, P., Bailey, J., Leckie, C., Ramamohanarao, K., & Davidson, I. (2018a). Image constrained blockmodelling: A constraint programming approach. In *Proceedings of the 2018 SIAM International Conference on Data Mining*, pp. 19–27. Society for Industrial and Applied Mathematics.

Ganji, M., Chan, J., Stuckey, P. J., Bailey, J., Leckie, C., Ramamohanarao, K., & Park, L. (2018b). Semi-supervised Blockmodelling with Pairwise Guidance. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 158–174. Springer.

Gay, S., Hartert, R., & Schaus, P. (2015). Conflict Ordering Search for Scheduling Problems. In *Principles and Practice of Constraint Programming*.

Girvan, M., & Newman, M. E. (2002). Community structure in social and biological networks. *Proceedings of the national academy of sciences*, *99*(12), 7821–7826.

Grünwald, P. D. (2007). *The Minimum Description Length Principle*. MIT Press, Cambridge, MA.

Gurobi Optimization, L. (2018). *Gurobi Optimizer Reference Manual*.

Hebrard, E., & Siala, M. (2017). Explanation-Based Weighted Degree. In *Integration of AI and OR Techniques in Constraint Programming - 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings*, pp. 167–175.

Holland, P. W., Laskey, K. B., & Leinhardt, S. (1983). Stochastic blockmodels: First steps. *Social Networks*, *5*(2), 109–137.

Karrer, B., & Newman, M. E. J. (2011). Stochastic blockmodels and community structure in networks. *Physical Review E*, *83*(1).

Knuth, D. E. (1993). *The Stanford GraphBase*. Addison-Wesley.

Krompaß, D., Nickel, M., Jiang, X., & Tresp, V. (2013). Non-negative tensor factorization with rescal. In *Tensor Methods for Machine Learning, ECML workshop*.

Kuo, C.-T., Ravi, S. S., Dao, T.-B.-H., Vrain, C., & Davidson, I. (2017). A Framework for Minimal Clustering Modification via Constraint Programming. In *the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, San Francisco, United States.

le Clément de Saint-Marcq, V., Schaus, P., Solnon, C., & Lecoutre, C. (2013). Sparse-sets for domain implementation. In *The 19th International Conference on Principles and Practice of Constraint Programming, Uppsala, Sweden, September 16 – 20, 2013.*

Long, B., Zhang, Z., & Yu, P. S. (2010). A general framework for relation graph clustering. *Knowledge and Information Systems; London*, *24*(3), 393–413.

Lorrain, F., & White, H. C. (1971). Structural equivalence of individuals in social networks. *The Journal of Mathematical Sociology*, *1*(1), 49–80.

Lusseau, D., Schneider, K., Boisseau, O. J., & Haase, P. (2003). The bottlenose dolphin community of Doubtful Sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology*, *54*(4), 396–405.

Michel, L., & Hentenryck, P. V. (2012). Activity-Based Search for Black-Box Constraint Programming Solvers. In *9th International Conference, CPAIOR 2012, Nantes, France, May 28 - June1, 2012. Proceedings*, pp. 228–243.

Miettinen, P., & Vreeken, J. (2014). MDL4BMF: Minimum Description Length for Boolean Matrix Factorization. *ACM Trans. Knowl. Discov. Data*, *8*(4), 18:1–18:31.

OscaR Team (2012). *OscaR: Scala in OR.*

Peixoto, T. P. (2013). Parsimonious Module Inference in Large Networks. *Phys. Rev. Lett.*, *110*(14), 148701.

Peixoto, T. P. (2017a). Bayesian stochastic blockmodeling. *arXiv:1705.10225 [cond-mat, physics:physics, stat]*. arXiv: 1705.10225.

Peixoto, T. P. (2017b). Nonparametric Bayesian inference of the microcanonical stochastic block model. *Phys. Rev. E*, *95*(1), 012317. arXiv: 1610.02703.

Prosser, P., & Unsworth, C. (2006). A connectivity constraint using bridges. In *ECAI*, pp. 707–708.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, *14*(5), 465–471.

Ropke, S., & Pisinger, D. (2006). An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, *40*(4), 455.

Rossi, F. (Ed.). (2006). *Handbook of constraint programming.* Elsevier, Amsterdam.

Rosvall, M., & Bergstrom, C. T. (2007). An information-theoretic framework for resolving community structure in complex networks. *PNAS*, *104*(18), 7327–7331.

Schulte, C. (1999). Comparing Trailing and Copying for Constraint Programming. In *Logic Programming: The 1999 International Conference, Las Cruces, New Mexico, USA, November 29 - December 4, 1999*, pp. 275–289.

Shaw, P. (1998). Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In Maher, M., & Puget, J.-F. (Eds.), *Principles and Practice of Constraint Programming — CP98*, Lecture Notes in Computer Science, pp. 417–431, Berlin, Heidelberg. Springer.

Sundar, S., & Singh, A. (2015). Metaheuristic Approaches for the Blockmodel Problem. *IEEE Systems Journal*, *9*(4), 1237–1247.

Van Hentenryck, P., & Michel, L. (2008). The Steel Mill Slab Design Problem Revisited. In Perron, L., & Trick, M. A. (Eds.), *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Lecture Notes in Computer Science, pp. 377–381. Springer Berlin Heidelberg.

van Hoeve, W.-J., & Katriel, I. (2006). Global constraints. In *Foundations of Artificial Intelligence*, Vol. 2, pp. 169–208. Elsevier.

Wang, F., Li, T., Wang, X., Zhu, S., & Ding, C. (2011). Community discovery using nonnegative matrix factorization. *Data Min Knowl Disc*, *22*(3), 493–521.

Wasserman, S., & Faust, K. (2018). *Social network analysis: methods and applications* (4th print. with corr edition). No. 8 in Structural analysis in the social sciences. Cambridge university press, Cambridge.

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of 'small-world'networks. *nature*, *393*(6684), 440.

Yan, X. (2016). Bayesian model selection of stochastic block models. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pp. 323–328. ISSN: null.

Zachary, W. W. (1977). An Information Flow Model for Conflict and Fission in Small Groups. *Journal of Anthropological Research*, *33*(4), 452–473.

Zampelli, S., Deville, Y., & Dupont, P. (2006). Symmetry Breaking in Subgraph Pattern Matching. In *Trends in Constraint Programming.*