Generic Constraint-based Block Modeling using Constraint Programming

Alex Mattenet*[1], Ian Davidson[2], Siegfried Nijssen[1], and Pierre Schaus[1]

[1] ICTEAM, UCLouvain, Belgium {alex.mattenet, siegfried.nijssen, pierre.schaus}@uclouvain.be
[2] Computer Science Department, University of California at Davis, USA davidson@cs.ucdavis.edu

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 the problem 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. We show its use in the analysis of real datasets.

1 Introduction

Block modeling has a long history in the analysis of social networks [31]. 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 directions. Firstly, they were not scalable; secondly, they often found results that

* This author is supported by a FRIA grant.

were inconsistent with the expectations of domain experts. To solve both problems, the use of constraints has been studied in the literature. For example: i) Entry level constraints such as non-negativity [30], ii) Incorporating simple composition constraints on the blocks such as spatial continuity or together/apart constraints [3,16], iii) Constraints on the interaction/image matrix [15] and even iv) simultaneous constraints on blocks and interaction/image matrices [2].

However, each of these studies yielded a different type of solver that is only scalable for one specific problem setting. For example, in [3] an update rule for the LM method is used, and in [2] 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 other methods by orders of magnitude.

2 Related Work

Block modeling in practice has two core computational challenges: i) the problem needs to be solved as a discrete optimization problem to be truly interpretable. 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. There are some MIP formulations of block modeling

[9], but as we show in this paper (Table 3) their run time is extremely slow. Instead, most work has focused on relaxing the problem to a continuous problem and adding constraints. There are 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.

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 [3]	A soft constraint based on a kernel	Additive Update Rule
Path [2]	All nodes in a block have a path to each other	Multiplicative Update Rule
Composition [16]	Must-link/cannot-link constraints	Gradient Descent
Image Structure [15]	Constraints on image matrix	Gradient Descent

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 [22]. Formally, given a graph $G = (V, E)$, vertices $u, v \in V$ are structurally equivalent $u \equiv v$ if and only if $\forall x \in V : (u, x) \in E \iff (v, x) \in E \land (x, u) \in E \iff (x, v) \in E$.

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) [4].

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

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

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

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

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 $\textsc{Blockmodel}(X, k)$ 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 paper are (1) a CP model for the block modeling problem, (2) a global constraint used in this model, that we call `blockModelCost`, and (3) a tailored filtering algorithm for this constraint. We first describe the CP model, with its variables and constraints. Afterwards, we present the global constraint and its filtering algorithm. Finally, we present a heuristic and symmetry breaking scheme for the CP solver based on the global constraint.

There are four groups of variables in our model: the cluster variables C, the image matrix variables M, the block cost variables cost and the total cost of our solution $\mathsf{totalCost}$. They are presented in this table:

Variable	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 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:

- $\mathtt{sum}(\mathsf{cost}, \mathsf{totalCost})$, which ensures that the total cost of the solution and the individual cost of every block stays consistent: $\sum_{c=1}^{k} \sum_{d=1}^{k} \mathsf{cost}_{cd} = \mathsf{totalCost}$. This constraint is already implemented in most CP systems.
- $\mathtt{atLeast}(1, \mathsf{C}, c), \forall c \in \{1..k\}$, which 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.
- $\mathtt{blockModelCost}(X, \mathsf{M}, \mathsf{C}, \mathsf{cost}, \mathsf{totalCost})$. This is the global constraint that we add to the solver, which filters the values of the different variables along the search. It ensures $\sum_{i=1}^{n} \sum_{j=1}^{n} |X_{ij} - M_{C_i C_j}| \leq \mathsf{totalCost}$ and $\sum_{i=1}^{n} \sum_{j=1}^{n} (C_i = c) \cdot (C_j = d) \cdot |X_{ij} - M_{cd}| \leq \mathsf{cost}_{cd} \; \forall c, d$.

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 [20] 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 permits filtering more impossible values and is often also faster [7]. 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.

Table 2. Adjacency matrix with its columns and rows reordered to show the partial assignment of vertices into clusters.

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	·

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: $\mathsf{C} = (\{1\}, \{1\}, \{3\}, \{3\}, \{2\}, \{1, 2, 3\}, \{1\}, \{1, 2, 3\}, \{1\}), \forall c, d : \mathsf{M}_{cd} \in \{0, 1\}$, $\mathsf{cost}_{cd} \in \{0..13\}$ and $\mathsf{totalCost} = \{0..13\}$. In Table 2, we show the adjacency matrix X for this example with its rows and columns reordered to show the current partial assignment.

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 the 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, we get

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

After propagating the `sum` constraint, we get $\mathsf{totalCost} \in \{9..13\}$ and

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

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 $\mathsf{M}_{1,1} = 0$.

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.

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 $\mathsf{C}_6 \in \{2, 3\}$, 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 $\mathsf{totalCost} \in \{12..13\}$.

5.2 Filtering Algorithm

The pseudocode 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 [27], 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: $\mathtt{nb0Block}_{cd} = \#\{X_{ij} \mid \mathsf{C}_i = c, \mathsf{C}_j = d, X_{ij} = 0\}$,
- `nb0Row`, a $n \times b$ array where for all unbound vertices i: $\mathtt{nb0Row}_{ic} = \#\{X_{ij} \mid \mathsf{C}_j = c, X_{ij} = 0\}$,
- `nb0Col`, a $b \times n$ array where for all unbound vertices i: $\mathtt{nb0Col}_{ci} = \#\{X_{ji} \mid \mathsf{C}_j = c, X_{ji} = 0\}$,

as well as their equivalent variables for the number of ones: `nb1Block`, `nb1Row` and `nb1Row`. The set of unbound vertices $\mathtt{unboundVertices} = \{i \in \{1..n\} \mid 1 < |\,\mathrm{dom}(\mathsf{C}_i)|\}$ is maintained in a reversible sparse set [26].

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

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

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

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

Similarly we maintain $\underline{colcost}(c,i)$, defined equivalently from $\mathtt{nb0Col}$ and $\mathtt{nb1Col}$. 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

The algorithm was designed with practical performance for block modeling in mind. For example, we only implemented filtering on the lower bound of the cost variables, since we are trying to minimize them. Of course, filtering their upper bound would also be possible, but was not implemented since it does not help solve the block modeling problem. Similarly, our algorithm does not have some of the theoretical guarantees ensured by well-known other global constraints, such as bound or arc consistency and idempotency. These would be very complex to implement for this problem, and would not improve the performance. We will nontheless discuss them in this section.

Soundness, Completeness and Idempotency: The filtering is sound (any pruned value is inconsistent with respect to the objective) but it does not achieve any classical notion of consistency. Our focus was on practical performance rather than theoretical guarantees. We added fine-grained filtering only when it improved efficiency. Consequently, propagation is also 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 other, lighter constraints helps in practice.

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.

```
/* update local counters */
for all $\mathsf{C}_i = \{c\} \in \Delta\mathsf{C}$ do
  $\texttt{unboundVertices} \leftarrow \texttt{unboundVertices} \setminus \{i\}$
  for all $j$ in $\texttt{unboundVertices}$ do
    $\texttt{nb1Col}_{cj} \mathrel{+}= X_{ij}; \texttt{nb0Col}_{cj} \mathrel{+}= (1 - X_{ij})$
    $\texttt{nb1Row}_{jc} \mathrel{+}= X_{ji}; \texttt{nb0Row}_{jc} \mathrel{+}= (1 - X_{ji})$
  end for
  for all $d = 1$ to $k$ do
    $\texttt{nb0Block}_{cd} \mathrel{+}= \texttt{nb0Row}_{id}; \texttt{nb1Block}_{cd} \mathrel{+}= \texttt{nb1Row}_{id}$
    $\texttt{nb0Block}_{dc} \mathrel{+}= \texttt{nb0Col}_{di}; \texttt{nb1Block}_{dc} \mathrel{+}= \texttt{nb1Col}_{di}$
  end for
  $\texttt{nb0Block}_{cc} \mathrel{+}= (1 - X_{ii}); \texttt{nb1Block}_{cc} \mathrel{+}= X_{ii}$
end for
/* filter cost and totalCost */
minCost $\leftarrow 0$
for all $c, d \in \{1..k\} \times \{1..k\}$ do
  update min of $\mathsf{cost}_{cd}$ to $\underline{cost}(c, d)$.
  minCost $\mathrel{+}= \underline{cost}(c, d)$.
end for
for all unbound vertex $i$, $c = 0$ to $k$ do
  minCost $\mathrel{+}= \underline{colcost}(c, i) + \underline{rowcost}(c, i)$
end for
update min of totalCost to minCost
/* filter M */
for all $c, d \in \{1..k\} \times \{1..k\}$ if $\mathsf{M}_{cd} = \{0, 1\}$ do
  if $\texttt{nb0Block}_{cd} > \max(\mathsf{cost}_{cd})$ then $\mathsf{M}_{cd} \leftarrow \mathsf{M}_{cd} \setminus \{0\}$ end if
  if $\texttt{nb1Block}_{cd} > \max(\mathsf{cost}_{cd})$ then $\mathsf{M}_{cd} \leftarrow \mathsf{M}_{cd} \setminus \{1\}$ end if
end for
/* filter C */
for all $i \in \texttt{unboundVertices}$, $c \in \mathsf{C}_i, d = 1$ to $k$ do
  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
    $\mathsf{C}_i \leftarrow \mathsf{C}_i \setminus \{c\}$
  end if
end for
```

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 $\mathcal{O}(u_{\mathsf{C}})$.

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 search procedure is a branch and bound depth-first-search. Two important components of a search procedure are the variable and value ordering heuristics. These should permit discovering rapidly good incumbent solutions in order to prune the search tree. Since the problem also exhibits value symmetries, we use a dynamic symmetry breaking scheme during the search. When the search space becomes too large, and there is no hope to explore completely the search tree, LNS (Large Neighborhood Search) [28] can be used on top of CP to diversify the search and discover good solutions rapidly.

Value and Variable 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 [17,19,23]. 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. Similarly for M, we branch first on $\mathsf{M}_{cd} = 0$ if $\texttt{nb1Block}_{cd} < \texttt{nb0Block}_{cd}$, and $\mathsf{M}_{cd} = 1$ otherwise.

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 [29]. 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 $\sigma : \{0..k\} \rightarrow \{0..k\}$ and any state $S = (\mathsf{C}, \mathsf{M})$, the permutated 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 [32].

7 Experiments

7.1 Comparison with MIP Model

The block modeling problem is often approximated using heuristic search. However, an approach to find the optimal solution is proposed in [9]. It builds on the work of [8], 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 and running the MIP solver for each matrix in this set.

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. 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 in [12, Chapters 2, 6], namely: (a) the Transatlantic Industries little league baseball team network, [14], (b) the Sharpstone little league baseball team network, [14], (c) the political actor network (PA), and [11] (d) the Kansas search and air rescue (SAR) network. [13].

The CP models were written and solved in OscaR [24]. The MIP model was written and solved in Java using Gurobi [18]. All experiments were run on a computer with Xeon Platinum 8160 24c/48t HyperThread processors. The results are shown in Table 3.

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.

Table 3. 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
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

7.2 Comparison with Local Search

The global constraint can also be used in local search by doing Large Neighborhood Search [28]. In this subsection, we compare the performance of the LNS approach with a local search algorithm for block modeling bundled in the popular graph processing software Pajek[3].

We generated synthetic graphs with 50, 100, 150 and 200 vertices — the classical block modeling algorithm included in Pajek [5] only supports graphs of less than 256 vertices — with a fixed block model structure of 5 clusters. We

[3] http://mrvar.fdv.uni-lj.si/pajek/

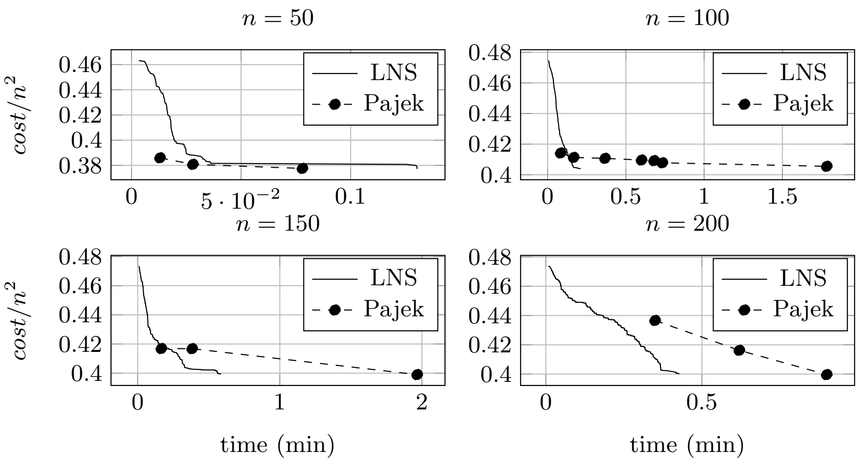

Fig. 3. Comparison of LNS search with the local search bundled in Pajek for synthetic dataset with 40% of noise. Each graph shows a different instance of the problem, for $n = 50$ to $n = 200$ by increments of 50.

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 shown in Figure 3. For all instances over 50 vertices, the LNS method outperformed Pajek's local search.

7.3 Scalability

We now show the scalability of the complete search and LNS method on larger instances. We once again generated synthetic graphs of different sizes n with a known block model structure and 20% of noise. In the first plot of Figure 4, we report the runtime until proving optimality for different sizes n and number of clusters k. In the second plot of Figure 4, we plot the convergence of Large Neighborhood Search over 10 minutes, with restarts every 1000 failed states and relaxation of 5% of the variables. We see that, while proving optimality is still prohibitively hard for large graphs, the LNS search converges quickly on a solution of optimal cost, even with thousands of vertices. 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.

7.4 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.

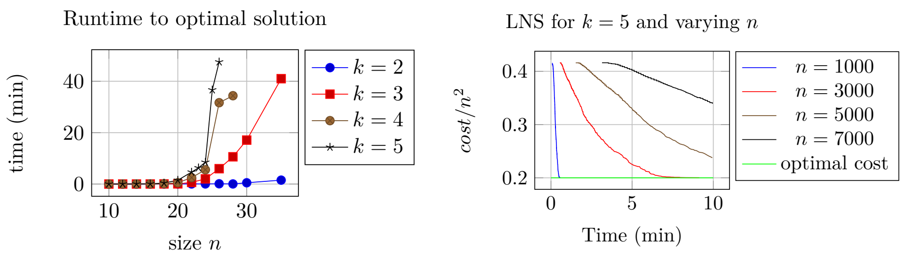

Fig. 4. Scalability on graphs with known block model and 20% of noise. The first graph shows runtime until the solution is proven optimal. The second shows the convergence of the solution with LNS.

In the first illustration, Figure 5, 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 [25,10,6]. In the second illustration, Figure 6, 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 blockmodel 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 [21].

The migration graphs were built from an open dataset provided by the World Bank [1]. 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, 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 5, 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 6, 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.

8 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

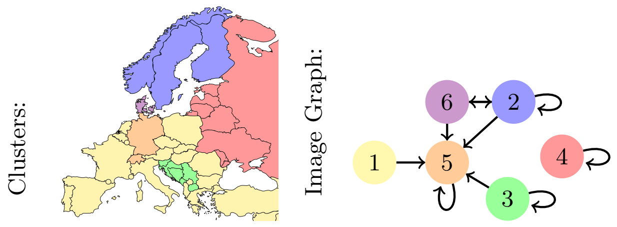

Fig. 5. A block model of migrant stocks in continental Europe in 2015, with geographically connected clusters.

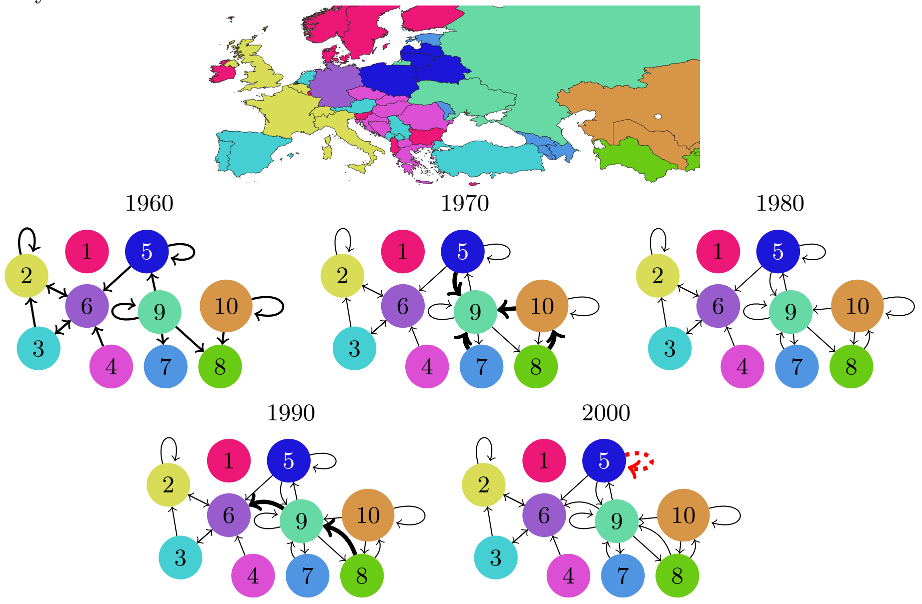

Fig. 6. RESCAL model for the evolution of migrant stocks in Europe. The edges which appeared in a decade are rendered in thick stroke, and those which disappeared are in dotted red stroke.

solutions to find optimal block models. Our CP formulation can also be used for heuristic search with Large Neighborhood Search.

This work could be further expanded with an equivalent global constraint for regular equivalence or generalized blockmodeling [12]. The search could be accelerated by breaking symmetries on the automorphisms of X and M and considering more advanced variable ordering schemes.

References

1. The world bank: Migration and remittances data, http://www.worldbank.org/en/topic/migrationremittancesdiasporaissues/brief/migration-remittances-dat

2. Bai, Z., Qian, B., Davidson, I.: 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 (2018)
3. Bai, Z., Walker, P., Tschiffely, A., Wang, F., Davidson, I.: 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 (2017)
4. Batagelj, V.: Notes on blockmodeling. Social Networks **19**(2), 143–155 (Apr 1997). https://doi.org/10.1016/S0378-8733(96)00297-3
5. Batagelj, V., Mrvar, A., Ferligoj, A., Doreian, P.: Generalized blockmodeling with pajek. Metodoloski zvezki **1**(2), 455 (2004)
6. Bessiere, C., Hebrard, E., Katsirelos, G., Walsh, T.: Reasoning about connectivity constraints. In: Twenty-Fourth International Joint Conference on Artificial Intelligence (2015)
7. Bessiere, C., Van Hentenryck, P.: To be or not to be... a global constraint. In: International conference on principles and practice of constraint programming. pp. 789–794. Springer (2003)
8. Brusco, M.J., Steinley, D.: 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 (Dec 2009). https://doi.org/10.1016/j.jmp.2009.08.003
9. Dabkowski, M., Fan, N., Breiger, R.: Exploratory blockmodeling for one-mode, unsigned, deterministic networks using integer programming and structural equivalence. Social Networks **47**, 93–106 (Oct 2016). https://doi.org/10.1016/j.socnet.2016.05.005
10. Dooms, G.: The CP(Graph) computation domain in constraint programming. Ph.D. thesis, UCL - Université Catholique de Louvain (2006), https://dial.uclouvain.be/pr/boreal/object/boreal:107275
11. Doreian, P., Albert, L.H.: Partitioning Political Actor Networks: Some Quantitative Tools for Analyzing Qualitative Networks. Journal of Quantitative Anthropology **1**, 279–291 (1989), https://www.ifip.com/Partitioning_Political_Actor.html
12. Doreian, P., Batagelj, V., Ferligoj, A.: Generalized Blockmodeling. Cambridge University Press (2005)
13. Drabek, T., Tamminga, H., Kilijanek, T., Adams, C.: Managing multi-organizational emergency responses. Boulder: University of Colorado, Institute of Behavioral Science (1981)
14. Fine, G.A.: With the Boys: Little League Baseball and Preadolescent Culture. University of Chicago Press (Mar 1987), google-Books-ID: 2qWgZPuNjEYC
15. Ganji, M., Chan, J., Stuckey, P., Bailey, J., Leckie, C., Ramamohanarao, K., Davidson, I.: Image Constrained Blockmodelling: A Constraint Programming Approach. In: Proceedings of the 2018 SIAM International Conference on Data Mining, pp. 19–27. Proceedings, Society for Industrial and Applied Mathematics (May 2018). https://doi.org/10.1137/1.9781611975321.3
16. Ganji, M., Chan, J., Stuckey, P.J., Bailey, J., Leckie, C., Ramamohanarao, K., Park, L.: Semi-supervised blockmodelling with pairwise guidance. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 158–174. Springer (2018)
17. Gay, S., Hartert, R., Lecoutre, C., Schaus, P.: Conflict ordering search for scheduling problems. In: Principles and Practice of Constraint Programming. pp. 140–148. Springer International Publishing (2015)

18. Gurobi Optimization, L.: Gurobi optimizer reference manual (2018), http://www.gurobi.com
19. Hebrard, E., Siala, M.: 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 (2017). https://doi.org/10.1007/978-3-319-59776-8_13
20. van Hoeve, W.J., Katriel, I.: Global constraints. In: Foundations of Artificial Intelligence, vol. 2, pp. 169–208. Elsevier (2006)
21. Krompaß, D., Nickel, M., Jiang, X., Tresp, V.: Non-negative tensor factorization with rescal. In: Tensor Methods for Machine Learning, ECML workshop. (2013)
22. Lorrain, F., White, H.C.: Structural equivalence of individuals in social networks. The Journal of Mathematical Sociology **1**(1), 49–80 (Jan 1971). https://doi.org/10.1080/0022250X.1971.9989788
23. Michel, L., Hentenryck, P.V.: 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 (2012). https://doi.org/10.1007/978-3-642-29828-8_15
24. OscaR Team: OscaR: Scala in OR (2012), available from `https://bitbucket.org/oscarlib/oscar`
25. Prosser, P., Unsworth, C.: A connectivity constraint using bridges. In: ECAI. pp. 707–708 (2006)
26. le Clément de Saint-Marcq, V., Schaus, P., Solnon, C., Lecoutre, C.: Sparse-sets for domain implementation. In: The 19th International Conference on Principles and Practice of Constraint Programming, Uppsala, Sweden, September 16 – 20, 2013 (2013), https://dial.uclouvain.be/pr/boreal/object/boreal:135574
27. Schulte, C.: 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 (1999)
28. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: International conference on principles and practice of constraint programming. pp. 417–431. Springer (1998)
29. Van Hentenryck, P., Michel, L.: 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. pp. 377–381. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2008)
30. Wang, F., Li, T., Wang, X., Zhu, S., Ding, C.: Community discovery using nonnegative matrix factorization. Data Mining and Knowledge Discovery **22**(3), 493–521 (May 2011). https://doi.org/10.1007/s10618-010-0181-y
31. Wasserman, S., Faust, K.: Social network analysis: methods and applications. No. 8 in Structural analysis in the social sciences, Cambridge university press, Cambridge, 4th print. with corr edn. (2018)
32. Zampelli, S., Deville, Y., Dupont, P.: Symmetry Breaking in Subgraph Pattern Matching. In: Frédéric Benhamou,Narendra Jussien, Barry O'Sullivan ; ”Trends in Constraint Programming”- p. 203-218 (ISBN : 978-1-905209-97-2) (2006), https://dial.uclouvain.be/pr/boreal/object/boreal:85004