Constraint Programming for an Efficient and Flexible Block Modeling Solver

Alex Lucía Mattenet,[1] Ian Davidson,[2] Siegfried Nijssen,[1] Pierre Schaus[1]
[1]UCLouvain, ICTEAM, Belgium, [2]Computer Science Department, University of California at Davis, USA
{alex.mattenet, siegfried.nijssen, pierre.schaus}@uclouvain.be,
davidson@cs.ucdavis.edu

Abstract

Constraint Programming (CP) is a powerful paradigm for solving combinatorial problems. In CP, the user creates a model by declaring variables with their domains and expresses the constraints that need to be satisfied in any solution. The solver is then in charge of finding feasible solutions—a value in the domain of each variable that satisfies all the constraints. The discovery of solutions is done by exploring a search tree that is pruned by the constraints in charge of removing impossible values. The CP framework has the advantage of exposing a rich high-level declarative constraint language for modeling, as well as efficient purpose-specific filtering algorithms that can be reused in many problems. In this work, we harness this flexibility and efficiency for the Block Modeling problem. It is a variant of the graph clustering problem that has been used extensively in many domains including social science, spatio-temporal data analysis and even medical imaging. 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. We show its use in the analysis of real datasets.

This is an extended abstract of an earlier publication at CP2019 (Mattenet et al. 2019).

Introduction

Block modeling is a problem that originates from the analysis of social networks. The core problem is to take a graph and divide its vertices into k clusters, in such a way that vertices in the same cluster have the same pattern of ties to other vertices. These clusters and the interactions between them summarize the graph and give insight into its large-scale structure.

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 clusters stacked column wise. Here $F_{i,j} \in \{0, 1\}$ and M is a $k \times k$ image matrix showing the interaction between clusters. The objective function is to minimize the reconstruction error $||X - FMF^t||$.

This block modeling formulation has the advantage of identifying structural equivalence (Lorrain and White 1971): 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.

An example of block model with 3 clusters for a toy graph with 9 vertices is given in Figure 1. The three clusters are $c_1 = \{a, h, g\}$, $c_2 = \{b, i, f\}$, and $c_3 = \{c, d, e\}$. The adjacency matrix of the graph is shown with the columns and rows reordered to group the vertices in their respective cluster. The cost of this block model is 9. The nine entries where X differs from FMF^t are in red italics. The image matrix M corresponds to an *image graph* of three vertices (one per cluster), which gives a simplified overview of the large-scale structure of the base graph.

As this block model formulation is very generic, it is useful to incorporate additional constraints such as bounds on the cluster size, constraints on the structure of the image graph M (forcing it to be a tree, a ring graph,...), constraints on the composition of the clusters, and more. This is to combine strong semantic knowledge (the constraints) along with empirical evidence (the graph). The constraints exclude unwanted or useless models from the search space.

Existing approaches for solving this problem with additional constraints focus on one or two constraint at a time (Wang et al. 2011; Bai et al. 2017; Ganji et al. 2018a; Bai, Qian, and Davidson 2018) and are not compatible. It is impossible to use all of multiple constraints at the same time because they use incompatible solving methods, and they are either not usable or not scalable without the predefined constraints.

In this paper, we propose a Constraint Programming approach for the block modeling problem, with a dedicated global constraint. This constraint and its implementation within a CP framework makes that our approach is inherently composable with any of the others constraints implemented in the CP solver. Furthermore, the CP solver can be used both to find exact solutions, and to find approximations using Large Neighborhood Search. We compare our CP approach with existing exact methods and local search methods, and show that it outperforms the state of the art.

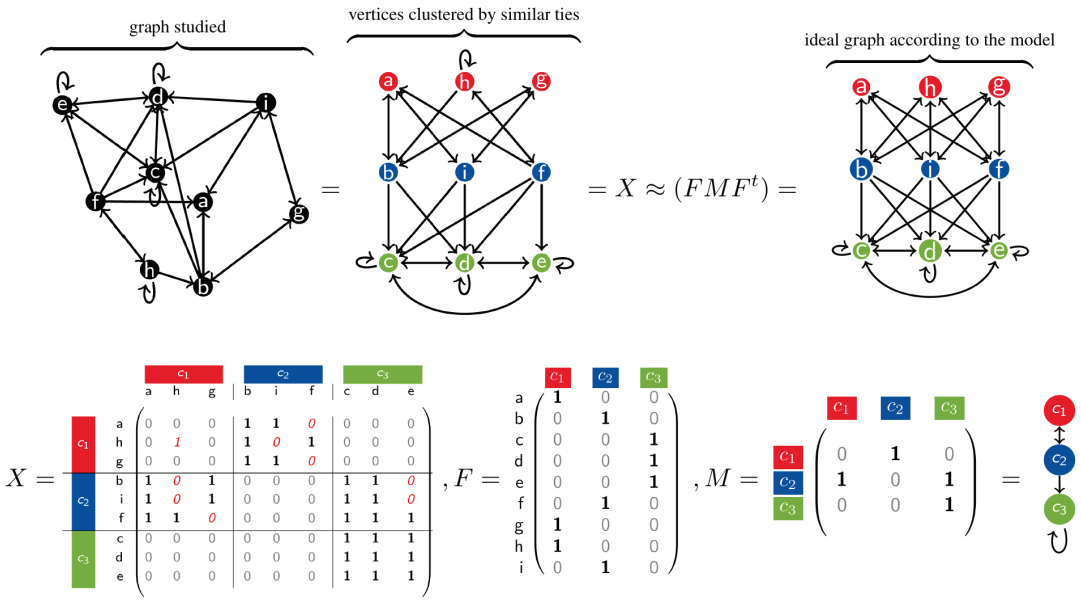

Figure 1: Illustration of a block model for a small graph. The adjacency matrix X has been reordered to group the vertices in their respective clusters. The entries where the X differs from the model are in red italics. The cost of this block model is 9.

Constraint Programming Approach

Constraint Programming is a powerful paradigm for solving combinatorial search problems. It has been successfully applied to many domains, such as scheduling, planning, vehicle routing, and bioinformatics (Rossi, van Beek, and Walsh 2006). The basic idea of CP is that the user gives a description of the problem in terms of variables and constraints between them, and a solver will search for suitable solutions. We refer to (L. Michel 2019) for more information on CP and CP systems.

In recent years, there has been a trend for using CP in data mining (Bessiere et al. 2016; Aoga, Guns, and Schaus 2016; Dao et al. 2018). Many data mining and AI problems are fundamentally discrete optimization. However, these discrete optimization problems are typically relaxed to continuous optimization problems for mathematical convenience, as is often the case for the block modeling problem. The continuous relaxation means compromise in terms of i) loss of interpretability and ii) inability to enforce semantic constraints. CP formulations allow us to model the problem correctly, as well as giving us access to efficient generic solvers and heuristics, and ease of modification and extension of the model, with a treasure-trove of existing additional constraints which can be added or removed to fit the specific problem and domain-specific expertise.

There is work on incorporating CP as a component inside existing iterative block modeling algorithms (Ganji et al. 2018b), but to our knowledge there exists no efficient solver for this problem developed entirely within the CP framework. In our work, we fill this gap by introducing a CP model for the Block Modeling problem, with a dedicated constraint and filtering algorithm.

```
1  class BMCP(graph: Digraph, val k: Int) extends CPModel {
2    val n = graph.n // number of nodes in the graph
3    val X = graph.adjacencyMatrix
4    // CP variables
5    val C = Array.fill(n)(CPIntVar(0 until k))
6    val M = Array.fill(k,k)(CPIntVar(0, 1))
7    val cost = Array.fill(k,k)(CPIntVar(0 to n*n))
8    val totalCost = CPIntVar(0 to n*n)
9    // CP constraints
10   for(c <- 0 until k) add(atLeast(1, C, c))
11   add(sum(cost.flatten, totalCost))
12   add(blockModelCost(X,M,C,cost,totalCost))
13 }
```

Listing 1: Implementation of the model for OscaR

CP Model for Block Modeling

The description (the *CP model*) for the block modeling problem is given in Listing 1. The code is written for OscaR (Team 2012), an open-source Scala CP library. The constructor for the class takes two parameters: a directed graph and the number of clusters k. With those, it instanciates the necessary variables and constraints, and registers them to OscaR's CP solver. Not included in this example is the code defining the search heuristics and starting the solver.

There are four groups of variables in our model. Each variable has a domain, which is the set of values it can take. They are defined as follows:

Line 5, C**:** n cluster variables, with $\mathsf{C}_i = c$ if vertex i is in cluster c. This is a different formulation of the F matrix;

Line 6, M**:** $k \times k$ image matrix variables, with $\mathsf{M}_{cd} = 0$ if the image graph has no arc from cluster c to cluster d, and 1 if it is has such interactions;

Line 7, cost**:** $k \times k$ variables, with cost_{cd} is the number of errors in block $c, d : \#\{(i,j) \mid X_{ij} \neq \mathsf{M}_{cd}, \mathsf{C}_i = c$, and $\mathsf{C}_j = d\}$ the number of entries with rows in cluster c and columns in cluster d where the value in X do not match M_{cd};

Line 8, totalCost**:** a variable representing $||X - FMF^t||$.

CP systems all come with a library of common constraints, such as $\texttt{sum}(\texttt{x}, \texttt{s})$ ensuring that $\sum_i \texttt{x}_i = \texttt{s}$, $\texttt{atLeast}(\texttt{n}, \texttt{x}, \texttt{v})$ ensuring that at least n variables in x have value v, etc. These are general constraints that appear in a wide range of problems and can be composed together. In line 10, we use atLeast constraints to ensure that there is at least one vertex in each cluster, and in line 11, we use a sum constraint to ensure that the total cost of the solution and the individual cost of every block stays consistent.

Other constraints are implemented for a domain-specific task. For examples, there are constraints for vehicle routing, for planning, for scheduling, and more. These constraints are less generic than the ones mentioned previously, but their filtering algorithm is much more efficient than the composition of multiple simple constraints.

In this work, we implemented a domain-specific constraint for the block modeling problem, blockModelCost. It is called on line 12. It filters the values of the different variables along the search, ensuring that the cost and totalCost variables stay consistent with their definiton with respect to X, C and M. Specifically, it ensures $\sum_{i=1}^{n}\sum_{j=1}^{n} |X_{ij} - M_{C_iC_j}| \leq \mathsf{totalCost}$ and $\sum_{i=1}^{n}\sum_{j=1}^{n}(C_i = c) \cdot (C_j = d) \cdot |X_{ij} - M_{cd}| \leq \mathsf{cost}_{cd}\ \forall c, d$.

The intuition for blockModelCost's filtering algorithm is that we keep track of some key values depending on the current partial assignment of vertices into clusters, and use these values to calculate a lower bound on the cost of the best block model possible with this partial assignment. These bounds are used to prune inconsistent values. The complete details and the pseudocode of the algorithm are given in the full paper, along with a discussion of its complexity.

Optimizing the Search Procedure

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 on the space of possible assignments of values to the C and M variables. At every node of the search, the solver will call a dedicated algorithm for each relevant constraint. These are called the constraints' *filtering algorithms*. They prune *inconsistent* values from the search space, i.e. values from the domain of their associated variables which cannot lead to a valid solution. If a variable has all its values pruned from its domain, then no valid solution can be found along this path of the search, and the solver backtracks. Once the solver finds a first solution with $\mathsf{totalCost} = x$, it will continue the search with the additional constraint that $\mathsf{totalCost} < x$, repeating this until the optimal solution has been found.

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) (Shaw 1998) 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 (Gay et al. 2015; Hebrard and Siala 2017; Michel and Hentenryck 2012). 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.

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 and 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, which are discussed in the full paper. Here we will only present one. It is clear that as long as the clusters stay the same, their labels can be changed. This kind of symmetry can be broken with a dynamic symmetry-breaking scheme: 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 $m = \max\{v \mid \exists i : \mathsf{C}_i = \{v\}\}$. This way we avoid exploring symmetrical states, without harming the value heuristics.

Evaluation and Results

In the full paper, we compare the performance of our CP approach to the current state of the art for exact solving of the Block Modeling problem, which uses a MIP model (Dabkowski, Fan, and Breiger 2016). We measure the run time to completion on well-studied datasets from the literature, and show that our approach is order of magnitudes better.

We also compare the performance of a popular local search algorithm for Block Modeling bundled in the Pajek software (Batagelj et al. 2004) with a Large Neighborhood Search using our CP model. We measure the evolution of the objective function with respect to time on synthetic datasets with up to 200 vertices (Pajek's limit). For all instances over 50 vertices, the LNS method outperformed Pajek's search.

Finally, we explore the scalability the LNS method on synthetic graphs of up to 7000 vertices with known blockmodel structure, and we illustrate possible uses of additional constraints (e.g. requiring that each cluster form a connected graph) with applications to the analysis of global human migration.

Conclusion and Further Work

In the full paper, we 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. 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 (Doreian, Batagelj, and Ferligoj 2005). The search could be accelerated by considering more complex symmetry-breaking techniques, notably breaking symmetries on automorphisms of X and M, and by considering more advanced variable ordering schemes.

Further work could also be done to study the effectiveness of the model with the types of additional constraints mentioned in the introduction. For each of these variants, one could compare different variable and value heuristics, and evaluate the performance compared to dedicated solvers. Finally, one could study the expansion of the CP model to the RESCAL setting (Krompaß et al. 2013) — similar to block modeling with multigraphs.

Acknowledgments

Alex Mattenet is supported by a FRIA grant.

References

Aoga, J. O.; Guns, T.; and Schaus, P. 2016. An efficient algorithm for mining frequent sequence with constraint programming. In *ECML-PKDD*. Springer.

Bai, Z.; Walker, P.; Tschiffely, A.; Wang, F.; and Davidson, I. 2017. Unsupervised network discovery for brain imaging data. In *SIGKDD*. ACM.

Bai, Z.; Qian, B.; and Davidson, I. 2018. Discovering models from structural and behavioral brain imaging data. In *SIGKDD*. ACM.

Batagelj, V.; Mrvar, A.; Ferligoj, A.; and Doreian, P. 2004. Generalized blockmodeling with pajek. *Metodoloski zvezki* 1(2).

Bessiere, C.; De Raedt, L.; Kotthoff, L.; Nijssen, S.; O'Sullivan, B.; and Pedreschi, D. 2016. *Data Mining and Constraint Programming - Foundations of a Cross-Disciplinary Approach*.

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

Dao, T.-B.-H.; Kuo, C.-T.; Ravi, S.; Vrain, C.; and Davidson, I. 2018. Descriptive clustering: Ilp and cp formulations with applications. In *IJCAI*. AAAI Press.

Doreian, P.; Batagelj, V.; and Ferligoj, A. 2005. *Generalized Blockmodeling*. Cambridge University Press.

Ganji, M.; Chan, J.; Stuckey, P.; Bailey, J.; Leckie, C.; Ramamohanarao, K.; and Davidson, I. 2018a. Image Constrained Blockmodelling: A Constraint Programming Approach. In *2018 SIAM International Conference on Data Mining*, Proceedings. Society for Industrial and Applied Mathematics.

Ganji, M.; Chan, J.; Stuckey, P. J.; Bailey, J.; Leckie, C.; Ramamohanarao, K.; and Park, L. 2018b. Semi-supervised blockmodelling with pairwise guidance. In *ECML-PKDD*. Springer.

Gay, S.; Hartert, R.; Lecoutre, C.; and Schaus, P. 2015. Conflict ordering search for scheduling problems. In *CP*. Springer International Publishing.

Hebrard, E., and Siala, M. 2017. Explanation-based weighted degree. In *CPAIOR*.

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

L. Michel, P. Schaus, P. V. H. 2019. Minicp: A lightweight solver for constraint programming.

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

Mattenet, A.; Davidson, I.; Nijssen, S.; and Schaus, P. 2019. Generic constraint-based block modeling using constraint programming. In *CP*.

Michel, L., and Hentenryck, P. V. 2012. Activity-based search for black-box constraint programming solvers. In *CPAIOR 2012, Nantes, France, May 28 - June1, 2012. Proceedings*.

Rossi, F.; van Beek, P.; and Walsh, T. 2006. *Handbook of constraint programming*. Amsterdam: Elsevier.

Shaw, P. 1998. Using constraint programming and local search methods to solve vehicle routing problems. In *CP*. Springer.

Team, O. 2012. OscaR: Scala in OR. Available from https://bitbucket.org/oscarlib/oscar.

Van Hentenryck, P., and Michel, L. 2008. The Steel Mill Slab Design Problem Revisited. In *CPAIOR*, LNCS. Springer Berlin Heidelberg.

Wang, F.; Li, T.; Wang, X.; Zhu, S.; and Ding, C. 2011. Community discovery using nonnegative matrix factorization. *Data Mining and Knowledge Discovery* 22(3).