13th International Conference on Applications of Statistics and Probability in Civil Engineering, ICASP13
Seoul, South Korea, May 26-30, 2019

Importance of Graph Topology Properties on SAT Solver Efficiency

Kyle Shepherd
PhD Student, Dept. of Civil Engineering, Rice University, Houston, United States

ABSTRACT: This report investigates how graph topological properties affect the performance of #SAT solvers. A general framework for representing #SAT problems as graphs while retaining information about the polarity of the variables is presented. From this representation, a selection of topological properties were extracted from a set of 2100 cubic graphs representing a vertex cover problem. Statistical summary features were computed from these properties and used to train a logistic regression model to predict whether classical #SAT solvers or newer TensorOrder #SAT solvers should be used. When compared to a model that just uses the number of nodes as the input feature, a model using topological properties selectes the wrong model 20% less often. It was determined that graphs with stable clusters as determined by the Markov Cluster Algorithm (MCL), graphs with small rings, and graphs with tree-like structure as measured by steady state diffusion, are determined to be most suitible for the TensorOrder #SAT solver.

1. Introduction

1.1. *Objectives*

The objective of this work is to investigate the graph topological properties that influence the performance of solvers that solve the #SAT problem, also known as the model counting problem. These topological properties will be used to build a simple model selection algorithm to evaluate the most important graph properties for model selection.

1.2. *Justification*

The model counting problem falls into the category of #P-Complete. Which means problems in general cannot be solved faster than exponential time. However, efficient #SAT solvers exist in practice because many classes of problems can be solved faster than exponential time. When these solvers are considered in aggregate and the fastest solver is used for each case, they are known as the Virtual Best Solver (VBS). The drawback is that it is difficult to know which solver will be fastest on a particular problem a priori.

We know that some solvers use the DPLL algorithm, which is a recursive decomposition algorithm. The efficiency of this algorithm is highly dependent on the structure of the problem. Other solvers have their complexity parameterized by the treewidth of the SAT problem. This dependence on structure implies that SAT problems can be represented by graphs, and implies that the properties of this graph representation can affect the efficiency of these solvers.

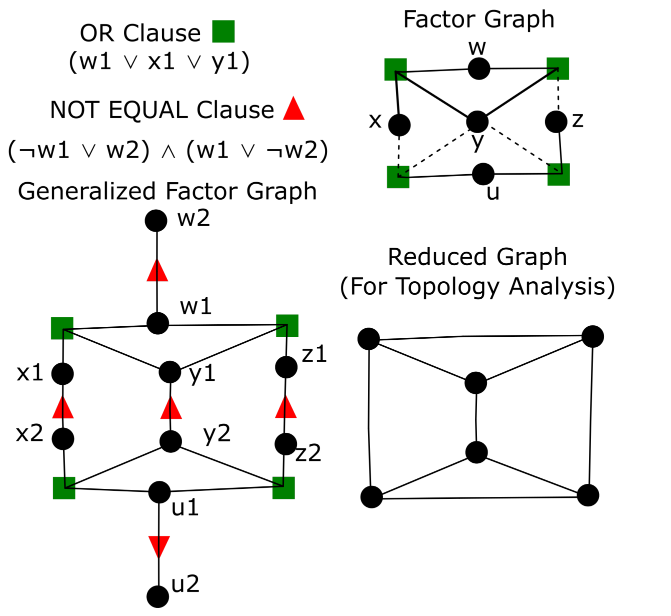
1.3. *Background*

SAT problems have been represented as graphs in the past. These are known as Factor Graphs, or Variable-Clause Graphs. This is a bipartite graph where one set of nodes are the clauses, one set of nodes are the variables, and the edges indicate which variables are in which clauses. However, when these graphs are drawn, dotted lines are used for negative literals and solid lines are for positive literals. This means the polarity of the variables does not affect the topology of factor graphs. However, we know that the polarity of variables can affect the performance of #SAT solvers, so this information must be recorded in the graph.

The performance of #SAT solvers on a benchmark collection of 1091 weighted model counting problems plus 2100 cubic vertex cover problems has been collected by Dudek et al. (2019). From

Figure 1: Generalized Factor Graph: The factor graph representation of the Φ SAT problem. The expansion to a generalized factor graph is shown, and the simplification removing all degree 1 and 2 nodes is shown

$\phi = (w \lor x \lor y) \land (w \lor y \lor \neg z) \land (\neg x \lor \neg y) \land (\neg y \lor z)$



this data, we know which solvers performed best on each problem. From this data, a model selection algorithm can be trained to measure features of the #SAT problem and then select the best #SAT solver to use. While prior work has been done on model selection by Nudelman et al. (2004), they used only basic features of the graph (degree distribution and one type of clustering), and used graph representations that did not capture the variable polarity.

1.4. Guidance to the Reader

The first step is to represent the #SAT benchmark problems as graphs. I will generalize the concept of Factor Graphs by adding virtual variables and virtual nodes. Each positive and negative literal will be represented by separate variables. For example, $\neg x$ and x becomes x_1 and x_2. These new variables will be connected by a new clause that enforces the fact that new variables must take different values.

The next step is to simplify the graphs. #SAT solvers can solve clauses with 1 or 2 variables in linear time, so I will iteratively remove all 1 degree nodes from the graph, and replace all 2 degree nodes with 1 spanning edge. This will leave a “core” network with nodes of degree 3 or higher. This kind of preprocessing is done for #SAT solver competitions. In combination with the previous step, this means each variable node in the original Factor Graph is split into two nodes connected by an edge, to represent the effect of positive and negative literals.

With the constructed graphs, a battery of graph topological properties will be extracted. For this report, only the features for the cubic graphs will be extracted and analyzed. The larger graphs in the weighted model counting caused some scaling problems with the feature extraction algorithms. Graph properties come in 3 varieties. Scalars such as diameter or max degree, vectors such as, eigenvalues, Shortest Path Ring statistics (SP Ring), or nodal betweenness, and matrices such as mean first passage time or shortest path length.

The goal is to feed these properties as features into a simple classification model to predict which #SAT solver is the fastest on each #SAT problem. Specifically, this model will predict if classical solvers (cachet, miniC2D, and d4) are faster than the TensorOrder solver by Dudek et al. (2019). A logistic regression model will be used. Scalars can be used directly in the classification model. For vector and matrix values, statistical scalar measures such as mean, median, variance, min, max, and entropy will be used.

The accuracy of the models will be evaluated, and the importance of the features will be investigated.

2. Results and Discussion

2.1. Feature Calculation

Table 1 displays the calculated properties and the calculated features. SP Ring statistics uses an algorithm from Franzblau (1991). It finds cycles in a graph with no chords, no shortcuts from one end of the cycle to the other. The Markov Cluster Algorithm (MCL) was developed by van Dongen (2000) and for this project I used the implementation by GuyAllard (2018). It finds graph clusters by alternating matrix multiplication (expansion) and matrix normalization (inflation). By setting the expansion coefficient to 3, and taking 20 evenly spaced samples of the inflation parameter from 1 to 6, the number of clusters can be obtained, and their sensitivity to the inflation parameter can be obtained. The Fiedler Vector Ratio is the ratio of positive values to negative values in the Fiedler Vector, the eigenvector associated with the second smallest eigenvalue. The Edge Cut of Spectral Clustering is the number of edges connecting the two groups as defined by the Fiedler Vector.

Steady State Diffusion is calculated by setting up a diffusion problem on the graph. All nodes receive a source of 1 substance, and a node i is designated as the "sink" node. The steady state substance values of the nodes are calculated, and a matrix of steady state values are obtained. In addition, the maximum and minimum flow into the sink for each node i is calculated, and the variance of the steady state values for each node i is calculated.

$SSD_{i,j}$ = steady state substance value of node j if node i is the sink

$SSmax_i$ = maximum substance value of the neighbors of node i

$SSmin_i$ = minimum substance value of the neighbors of node i

$SSvar_i$ = variance of steady state substance values if node i is the sink

2.2. *Benchmark Data Curation*

Some data curation was performed. Out of the original 2100 benchmarks, only 1768 of the benchmarks were solved by any #SAT solver. Therefore, this analysis will only consider the solved benchmarks. In addition, there was no attempt to differentiate between the three different classical solvers, the solve time of the TensorOrder method was compared to the best classical solver. This could hide some of the advantages of the TensorOrder method, if the classical methods cover for each other's weaknesses. However, in most cases miniC2D was the superior solver so most comparisons were between TensorOrder and miniC2D.

Table 1: List of Computed Features

Topology Properties Used	Summary Statistics Used
# of Nodes	Mean
# of Edges	Geometric Mean
Shortest Path	Harmonic Mean
Betweeness	Median
Mean First Passage Time	Lower 16th percentile
SP Ring Statistics	Upper 16th percentile
Node Centered Ring Statistics	Variance
Markov Cluster Algorithm	Skewness
Eigenvalues	Kurtosis
Fiedler Vector Ratio	Minimim
Edge Cut of Spectral Clustering	Maximum
Cosine Similarity	Entropy
Pearson Similarity	
Steady State Diffusion	

2.3. *Feature Average Comparison*

The first step to take is to determine if the average of any of the features is different between benchmarks solved fastest by the classical methods, and solved fasted by the tensor method. Table 2 shows the 8 most significant features.

It appears that Betweenness and Steady State Diffusion features are different between these two cases. The small values in the mean Pearson similarity feature suggests floating point errors are the cause of its apparent importance. Shortest path and mean first passing time also appear on this list. However, we know from Dudek et al. (2019) that the tensor methods are faster for larger networks, so the correlation with node count was checked. It appears that mean first passing time and Steady State Diffusion variance are independent from the node count and are possible candidates for the difference between classical methods and tensor methods.

Table 2: Features with large differences between benchmarks solved fast by classical methods, and solved fast by tensor methods. The correlation with the node count is shown.

Feature	Classical	Tensor	Corr
skewness MFPT	-0.033	-0.186	-0.04

skewness	-0.069	-0.308	-0.41
Between variance	6875	28710	0.88
Between kurt	1.512	4.762	0.50
SSflow Min mean	-4.505e-18	1.919e-18	0.28
PearSim kurt	-0.255	0.066	0.30
Between kurt	-0.37	-0.031	0.96
Shortest Path variance	1.473e-05	1.542e-06	-0.17
SSflow Var			

2.4. *Control Logistic Regression Model*

To measure the effect of these features in predicting what #SAT solver to use, a logistic regression model is trained. The model that is being fit is shown below.

$$p(\text{Tensor is faster}) = \frac{1}{1+e^{-(\beta_0+\beta_1 * x_1+\beta_2 * x_2 \dots)}}$$

First, a control model is trained using just the node count as a predictor for which #SAT solver to use. Table 3 shows the resulting parameters of this model. Overall, it has a low error rate, only predicting 7.35% of the benchmarks wrong. Generally, the model has higher confidence when predicting that the tensor method will be faster than the classical methods.

Table 3: Logistic regression model results (node feature only).

Parameter	Value
β_0	1.731
β_1	5.077
Wrong Prediction Rate	0.0735
Average confidence when predicting classical is faster	0.86
Average confidence when predicting tensor is faster	0.91

2.5. *Full Logistic Regression Model*

Next, a model will be trained using all of the features that were calculated. To ensure the features are equally considered, the features are standardized by subtracting by the mean and dividing by the standard deviation. Table 4 shows the resulting parameters of this full model. With all of the features, the model is more accurate, only making the wrong #SAT solver recommendation 5.825% of the time. In addition, when this model is providing its predictions, it is more confident in its results.

The features with the most influence include the upper 16th percentile SP Ring statistics, entropy of SP ring statistics, upper 16th percentile steady state diffusion values, lower 16th percentile MCL group count, and skewness of steady state diffusion value.

Table 4: Logistic regression model results (Fully Featured).

Parameter	Value
maximum magnitude β	1.277
Wrong Prediction Rate	0.05825
Average confidence when predicting classical is faster	0.90
Average confidence when predicting tensor is faster	0.93

2.6. *Principal Component Logistic Regression Model*

However, the coefficients of the logistic fit can be sensitive to variables that are correlated with each other. Therefore, to remove this correlation effect, a logistic regression model will be fit using principal components. The 10 most significant principal components were extracted from the normalized features, and then used to train the model.

Table 5 shows the resulting parameters of the model. While this model performs slightly worse than the fully featured model, this model can be used to extract the importance of each variable.

Table 5: Logistic regression model results (Principal Components).

Parameter	Value
maximum magnitude β	-0.656
Wrong Prediction Rate	0.0690
Average confidence when predicting classical is faster	0.88
Average confidence when predicting tensor is faster	0.92

The principal component with the largest influence in the model is the 1st component, and it explains 55% of the feature variance. The coefficients of this principal component can be inspected, and compared to the coefficient of the node count feature. Inspecting the coefficients of the 1st principal component vector, it seems in general the mean values of the features have the highest prominence, and have the same magnitude as the node count feature. Therefore, as the features increase along the direction of this principal component, the node count increases along with the means of the features. It seems this component captures the variation of solver speed with node count.

The principal component with the second largest influence in the model is the 8th component, and it explains 1.4% of the data variance. The node count of this component has much less influence, so values in this principal component can increase without also increasing the node count. Influential features appear to be minimum of ring statistics, variance of steady state diffusion variance, kurtosis of node centered SP Ring statistics, entropy of ring statistics, skewness of node centered SP Ring statistics, skewness of steady state diffusion maximums, kurtosis of steady state diffusion maximums, and variance of SP Ring statistics.

2.7. *Feature Importance*

One final check that can be performed is to sum up the influence of each feature across the model and the principal components, and then control for correlation with the node count.

$Influence_{feature} = \sum_{components}(\text{Model Weight}) *$ $(\text{Principal Component Weight})$
$Influence_{\text{Node Count Controlled}} =$
$Influence_{feature} - Influence_{\text{Node Count}} *$
$(\text{slope of linear fit with the feature})$

When this operation is performed, the following features in Table 6 are identified as the most important when controlled for node count. Features from the MCL group count appear frequently as important. The features from the steady state diffusion also appear frequently. Ring statistics also appears.

Table 6: Influential Features (controlled for node count).

Parameter	Influence
min MCL groups	0.132
harmonic mean MCL groups	0.130
variance MCL groups	-0.128
lower 16th percentile MCL groups	0.126
min Ring Statistics	-0.112
kurtosis steady state diffusion minimum	0.112
Datarange MCL groups	-0.107
kurtosis steady state diffusion maximum	0.106
variance steady state diffusion variance	-0.103
min steady state diffusion maximum	0.085

3. Conclusion

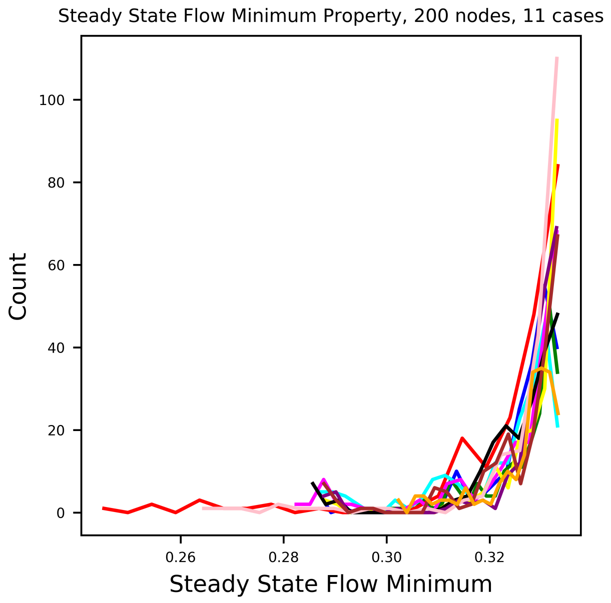
3.1. Results Summary

Overall, the size of the #SAT problem is the most important feature in deciding whether to use a classical #SAT solver or the TensorOrder #SAT solver. A model decision algorithm just using node count chooses the correct #SAT solver 93% percent of the time.

From Table 6, some conclusions about the desired properties for graphs being solved by the TensorOrder #SAT solver can be drawn. The graph needs a high minimum and average of clusters found by the MCL method, but the negative coefficient on the MCL data range implies that the number of groups found by MCL as the inflation parameter varies cannot vary too much. These results suggests that graphs need a large number of stable, tight clusters to be easily solved by the TensorOrder method.

In addition, the graph needs a high kurtosis for the minimum and maximum values of neighbor nodes found by the steady state diffusion method. This means when a node is a sink, it receives the majority of its flow from one neighbor. Examples

Figure 2: Steady State Flow Minimum Typical Distribution: 11 samples of 200 node graphs and their Steady State Flow Minimum are displayed. This distribution has a peak, its average, towards the right. There is also a bubble of activity in the middle and left, leading to heavy tail behavior.



of this property distribution are seen in Figure 2. When the kurtosis is high, it implies that the small density lump seen on the left is larger than normal, a heavier tailed distribution. Graphs solved faster by TensorOrder have heavier tails, more nodes getting most of its flow from one neighbor.

A node getting most of its flow from one neighbor implies some sort of topologic bottleneck between that node and the other nodes. This is similar to nodes on a tree, where there are very few paths to other nodes. Therefore, this property implies a tree like structure in the graph, which implies a small treewidth of the graph, enabling faster computation by the TensorOrder method which is bounded by the treewidth.

A low variance of steady state diffusion variance is desired for the TensorOrder #SAT solver. The steady state diffusion variance is the variance in the steady substance values for a given sink node. If this variance measure does not change if the sink node changes, that implies that each node is equally impactful on the network. This implies each node has a similar neighborhood, implying the presence of clusters.

Finally, the minimum value obtained by the ring statistics should be small for the TensorOrder #SAT solver. This implies that the graph should have triangles and squares, small cycles.

3.2. *Future Work*

This work can be expanded upon in multiple directions. Immediately, the weighted model counting benchmark cases can have their features measured. However, due to their size some of the topology properties cannot be extracted efficiently. The SP Rings algorithm and the mean first passage time algorithm do not scale well, and will likely have to be heavily modified, (such as capping the size of measured SP rings) or removed from the list of considered properties. The preliminary results from these cubic graphs suggests that these two properties are not crucial for predicting #SAT solver performance because they correlate with the size of the problem. Therefore, not too much information is lost by removing these features.

This work can be verified by producing artificial graphs that correspond to the proposed TensorOrder friendly properties discussed in the results summary, such as clusterability. Predictions for the best solver can be calculated ahead of time on the generated tests, and these predictions can be tested by measuring the running time of classical #SAT solvers and the TensorOrder #SAT solver.

In addition, more summary statistics can be extracted from the topology properties. The original plan in this report was to calculate features that rely on the probability mass distribution (PDF) of the values. These features included entropy and parameters of best fit distributions. However, developing general purpose code to fit distributions to arbitrary data was outside the scope of this report.

Some work has been performed on creating a smooth PDF of the data distribution, so properties such as entropy can be measured and so the tails of the distribution can be modeled. A smooth PDF also allows for better normalization of the data so the fitting models can be more accurate. One method includes fitting M-Splines and I-Splines to the data, and another method involves fitting piecewise monotonic functions (such as generalized logistic functions and monotonic cubic functions) to the data. However, as with the distribution fitting code, developing general purpose code to create smooth PDFs was outside the scope of this report.

The unusual distribution seen in the steady state distribution minimum flow needs to be investigated. The lumpy nature of the distribution implies a discrete set of network geometries, such as bottlenecks, that could be characterized and then measured. Overall, the underlying structures giving rise to the values calculated in the steady state diffusion method, and in other methods, needs to be investigated, so the behavior of different #SAT solvers can be analyzed.

4. REFERENCES

Dudek, J. M., Dueñas-Osorio, L., and Vardi, M. Y. (2019). "Efficient contraction of large tensor networks for weighted model counting through graph decompositions.

Franzblau, D. S. (1991). "Computation of ring statistics for network models of solids." *Phys. Rev. B*, 44, 4925–4930.

GuyAllard (2018). "Markov clustering in python, <https://github.com/GuyAllard/markov_clustering>.

Nudelman, E., Leyton-Brown, K., Hoos, H. H., Devkar, A., and Shoham, Y. (2004). "Understanding random sat: Beyond the clauses-to-variables ratio." *Principles and Practice of Constraint Programming – CP 2004*, M. Wallace, ed., Berlin, Heidelberg, Springer Berlin Heidelberg, 438–452.

van Dongen, S. (2000). "Graph clustering by flow simulation." Ph.D. thesis, University of Utrecht, hello world, <https://micans.org/mcl/index.html>.