Machine Learning on Optimizing Algorithms on Tree Decomposition

Zhiyuan Shen, Yifei Wang, Nuo Zheng

Abstract—NP problems are one of the most famous problem in the theoretical computer science aspect. The exponential time complexity makes it unsolvable if the data is large. However, many methods are found to reduce the problem scale of the origin problem, including tree-decomposition. Different tree-decomposition cost different run times. However, we can fastly generate tree-decompositions [2], that means, if choose the best tree-decomposition, the run time of program can be reduce a lot. Our goal is to do the further optimization based on the tree-decompositions.

✦

1 INTRODUCTION

This paper aims to use the machine learning to predict the run time of 3-colorability on a graph and its tree-decomposition. Thus we can generate multiple tree-decomposition, and find the fastest one then solve it.

This project chooses the 3-colorability as an example because it is a relatively easy NP-complete problem. This paper may first introducing some background information and give a fully view of how the project built.

This project conducts 3 parts, including data generating part, machine learning part and final test part. All 3 parts will be fully explained below analyses.

In conclusion, the paper will show the findings of optimization through machine learning on the tree-decompositions and try to explain the reasons behind such result.

2 RELATED WORKS

From the basic algorithm, the most traditional algorithm is to enumerate the colors of all nodes. After that, graph-coloring problem is determined to be the NP-complete problem. Rather than solve the problem straightforward, academic circle tries to reduce the scale of the problem. Tree-decomposition is an effective method[6] to reduce the scale of origin problem.

Then, the focus is shifted to the problems on the tree-decompositions. Improved tree-decomposition methods are found [2][3][5] to form a more standard and solvable structures.

Next, machine learning was introduced into this problem [1]. Machine learning is used to predict the run time of tree-decomposition structure and help problem-solver to dicide to solve which structure.

3 BACKGROUND INFORMATION

This section will introduce 3 basci concept of the project, 3-colorability, Tree-decomposition, and machine-learning.

3.1 3-colorability

A coloring of a graph refers to a proper vertex coloring, namely a labeling of the graph's vertices with colors such that no two vertices sharing the same edge have the same color.

A coloring using at most k colors is called a proper k-coloring. Thus if $k \leq 3$, we call the graph is 3-colorable.

The same, if a graph $G = (V, E)$ is 3 colorable. There should exist one example that $\forall v \in V, color(v) \in \{R, B, G\}$, and $\forall e = (u, v) \in E, color(u) \neq color(v)$.

3.2 Tree Decomposition

A *tree decomposition* of a graph $G = (V, E)$ is a tree T, where

1) Each vertex i of T is labeled by a subset $B_i \subset V$ of vertices G, referred to as a "bag".
2) Each edge of G is in a subgraph induced by at least one of the B_i
3) $\forall u \in V$, The subtree of T consisting of all "bags"containing u is connected.

A rooted *nice tree decomposition* (T, B) of a graph $G = (V, E)$ is *nice* if each node x of T is of one of the following types:

1) leaf node: x has no children and $|B_x| = 1$.
2) introduce node: x has a unique child y and $B_x = B_y \cup \{v\}$ for some $v \in V \backslash B_y$.
3) forget node: x has a unique child y and $B_x = B_y \backslash \{v\}$ for some $v \in B_y$.
4) join node: x has exactly 2 children y, z and $B_x = B_y = B_z$

We can convert a given tree decomposition of width t and $O(n)$ nodes into a nice tree decomposition of width t and $O(tn)$ nodes in time $t^{O(1)}n$[5].

3.3 Machine Learning and Predictions

Hence, we may select the features from the tree-decompositions and predict the run time of our algorithms on such structure.

The detailed information of features will be introduced in the later part.

4 DATA-SET GENERATING

This project generate all the data in Linux, Ubuntu 20.04 environment. The code could be found at https://github.com/jerry3128/Stats102-Final-Project-Linux.

In total, we generate 2000 graphs and 100 tree decompositions for each graph used to train our model. The cloud server location is 124.223.218.55.

4.1 Graph Generating

The undirected graph is generated based on uniformly randomly select the edge from the complete graph. In this way, it is convenient to control some features like the number of edges.

4.2 Tree Decomposition Generating

In this part, the implemention refers to hypertree decompositions(https://github.com/mabseher/htd), and [2]. Thus the tree-decomposition algorithm for this project is actually a combined algorithm.

In the process of generating tree-decomposition, we may find a maximum matching of the graph, and we can adjust the tree-decomposition after we done the algorithm, thus we can randomize the order to node to get different tree decomposition and add slight variables to the tree-decomposition to generate mutiple of them.

4.3 Problem Solver

To solve the 3-colorability on a graph with its tree-decomposition, we may use ASP (Answer Set program) principles [4].

Originally, we have the full set of answers. For 3-colorability, that means we can color every node into every color. When we gradually link the edges, new restrictions, such as the color of two nodes in a link cannot be the same, will be formed. Then, we delete the answers in the set that do not satisfy the new restrictions.

In the end, the remained answers are acceptable answers to origin problem.

To implement this algorithm, we can use back-track searching through C++ to stimulate the process. More information could be found in the code.

4.4 CSV File Generating

Another C++ program is written to generate csv file and deal with these features.

5 MACHINE LEARNING

In this section, we will introduce the features we select and how we implement the machine learning.

Apparently, compared to the other features, the *Tree Width* has much more impact on the run time. Thus we straightly group the data by tree width, and then for each group we do the polynomial linear regression.

Here is the features we may use:

1) Average Bag Size: Average of size of bags on tree decomposition.
2) Decomposition Overhead Ratio: Sum of bag size divided by size of graph.
3) Average Depth: Average depth of tree.
4) Node percentage: Introduce/Forget/Join/Leaf's Percentage of tree (sum of them is 100%).
5) Sum of Join node distance: Find all pairs of Join nodes and plus their distances.
6) Branching Factor: Average number of childs (exclude leaf nodes).
7) Bag Adjacency Factor: For all pair of nodes in a bag, that the adjacencent pairs divided by the total pairs.
8) Bag Connectedness Factor: For all pair of nodes in a bag, the reachable pairs divided by the total pairs.
9) Bag Neighborhood Coverage Factor: For all bags B_x, average of nodes that belong to a bag B_x or are adjacent to a node belongs to B_x

After training, we use cross validation to validate each model and select the polynomial linear regression.

We select the best models to participate in the final test.

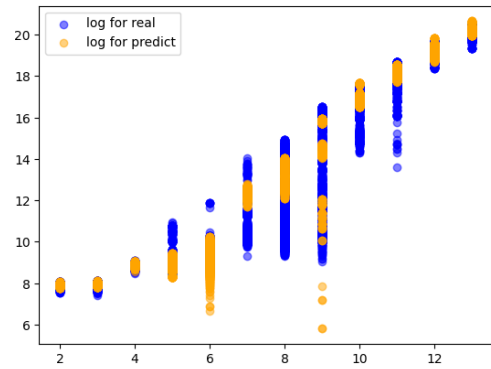Model: For each tree width, we use Linear-Regression to predict the runtime.

Fig. 1. separate tree-width, straight polynomial linear regression for run time

Also we compare this model with the other models by the square of the difference between the predictions and real value, and we call this.

The models for the ln of run time all has sum of square of difference over 10^6, while straightly linear regression the run time only over 10^5.

Then, we see that, if we combine all data, the model shows obvious inaccurate in the area that tree width is little.

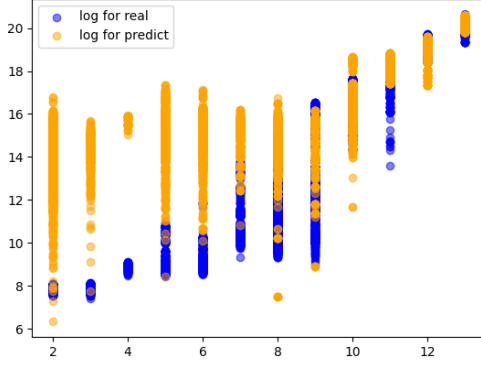Thus we may choose the separate tree width model as the predictor.

Fig. 2. combine tree-width, straight polynomial linear regression for run time

6 CONCLUSION

In this section, we compare efficiency of the

1) *predictor 1: machine learning predictor.*
2) *predictor 2: theory time complexity predictor.*
3) *predictor 3: average running time.*

to solve new generated graphs.

We will analysis the result and provide the further project focus.
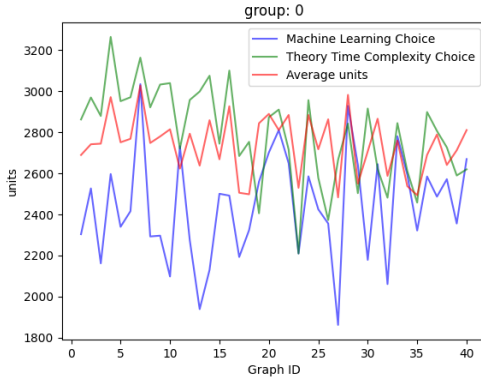

Fig. 3. compare the run time

The *predictor 1* shows a very high performance averagely. That only cost 87.4% run time of *predictor 2* and 91.0% run time of *predictor 3*.

However, here is 2% for this predictor that get huge differences, that the predicted run time is 2 times larger than real run time. Also, we found that the optimization is not so well when the tree width is large, and the efficiency smaller than related works.

We thought that, still the predictor has not sufficient data to do learning, especially in the large tree width area. Also, we found that our tree decomposition algorithm is a kind of combined algorithm, so that it is not uniformly random and will have a higher probability to already optimized tree decomposition.

Thus we may improve our project by constantly adding data and improve the basic algorithm parts.

REFERENCES

[1] M. Abseher, N. Musliu, and S. Woltran. Improving the efficiency of dynamic programming on tree decompositions via machine learning. *Journal of Artificial Intelligence Research*, 58:829–858, 04 2017.

[2] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

[3] H. L. Bodlaender and T. Kloks. Better algorithms for the pathwidth and treewidth of graphs. In J. L. Albert, B. Monien, and M. R. Artalejo, editors, *Automata, Languages and Programming*, pages 544–555, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[4] S. Gutin, G. Szeider. Parameterized and exact computation: 8th international symposium. In M. G. R. Niedermeier, editor, *Automata, Languages and Programming*. Springer Berlin Heidelberg, 2013.

[5] K. Kangas, M. Koivisto, and S. Salonen. A Faster Tree-Decomposition Based Algorithm for Counting Linear Extensions. In C. Paul and M. Pilipczuk, editors, *13th International Symposium on Parameterized and Exact Computation (IPEC 2018)*, volume 115 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:13, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[6] J. Matoušek and R. Thomas. Algorithms finding tree-decompositions of graphs. *Journal of Algorithms*, 12(1):1–22, 1991.