Report on Treewidth lower bound of Industrial SAT instances

Paul Suganthan G C

One of the most important structural parameters of graphs is treewidth, a measure for the “tree-likeness” and thus in many cases an indicator for the hardness of problem instances. The smaller the treewidth, the closer the graph is to a tree and the more efficiently the underlying instance often can be solved.

**Definition of Tree width :**

Treewidth intuitively is a measure of how tree-like a specific graph is. A tree decomposition for a graph \( G = (V,E) \) is a pair \( ((X_i | i \in I), T=(I,F)) \) where \( X_i \) is a bag of vertices representing a node of the tree decomposition and \( T \) is the tree. To be a valid decomposition, the following properties must hold:

1. \( \cup X_i = V \) (covers)
2. For \( (u,v) \in E \) there exists \( X_i \) such that \( u,v \in X_i \)
3. For all \( v \in V \), the set \( \{i \in I | v \in X_i \} \) is connected in \( T \)

It is known that computing the treewidth of a graph is NP-Complete. So we go in for finding the bounds for treewidth.

**SAT to Graph :**

We convert the SAT instance to a graphical model and then try to find the tree width lower bound for the resulting graph. We consider three types of graphical models – primal graph, incidence graph and the dual graph.

![Graphs](image)

*Figure 1: Graphs associated with the CNF formula \( F = \{C_1, \ldots, C_5\} \) with \( C_1 = \{u, \neg v, \neg y\}, C_2 = \{\neg u, z\}, C_3 = \{v, \neg w\}, C_4 = \{w, \neg x\}, C_5 = \{x, y, \neg z\}; \) the primal graph \( G(F) \), the dual graph \( G^d(F) \), and the incidence graph \( G^*(F) \).*
**SAT to Primal**

The primal graph has as vertices the variables of the given formula, two variables are joined by an edge if they occur together in a clause.

**SAT to Dual**

Symmetrically, the dual graph has as vertices the clauses of the formula, two clauses are joined by an edge if they share a variable.

**SAT to Incidence**

Finally, the incidence graph is a bipartite graph where one vertex class consists of the clauses of the given formula, and the other consists of the variables; a clause and a variable are joined by an edge if the variable occurs in the clause.

Depending on whether we consider the treewidth of the primal, dual, or incidence graph of the given CNF formula, we speak of the primal, dual, or incidence treewidth of the formula, respectively.

Dual graphs have received less attention compared to primal and incidence graphs due to their structure. So we try to exploit the primal and incidence treewidth.

We considered various approaches to determine the treewidth lowerbound,

**MMD+ Algorithm for Computing Lower bound of treewidth :**

**Input :** Generate graph \( G = (V, E) \) from the input DIMACS CNF file

**Output :** Lowerbound \( lowerbound \) on the treewidth of \( G \)

While \( G \) is not empty graph do

Find \( v \in V \) of minimum degree \( d \)

\( lowerbound = \max(lowerbound, d) \)

Contract \( v \) to neighbor \( w \) (select neighbor \( w \) that shares the smallest number of neighbours with \( v \))

Done

return \( lowerbound \)

**Correctness of the MMD+ Algorithm :**

**Lemma 1 :**

Let \( G = (V, E) \) be a graph, and \( W \subseteq V \) be a set of vertices. The treewidth of \( G[W] \) is at most the treewidth of \( G \).
**Proof:**

Let T be a tree decomposition of graph G, of width k. Now if we remove edges or vertices from the input graph, the size of each bag in tree decomposition T will only decrease. Thus an increase in the size of a bag is not possible. Thus the treewidth of G[W] is atmost the treewidth of G.

**Lemma 2:**

Let H = (W, F) be a minor of G = (V, E). Then the treewidth of H is atmost the treewidth of G.

**Proof:**

Similar to the deletion of edges or vertices, performing contraction also doesn’t increase treewidth. Let T be a tree decomposition of graph G, of width k. Now if we contract the edge (v,w) to vertex x, we build a tree decomposition of the new graph by replacing each occurrence of v or w in a bag by x. Thus it is not possible for the size of a bag to increase. Thus treewidth of H is atmost the treewidth of G.

**Why do we select the Min-degree vertex?**

Since finding treewidth of a graph is NP Complete, we resolve for computing the lower and upper bounds for treewidth.

**Lowerbound:**

The trivial lowerbound of treewidth is the minimum degree of the graph. Over the next iterations we try to improve this lowerbound.

Minimum degree (G) \(\leq\) Treewidth (G)

If G has treewidth k, then it has a vertex of degree at most k (consider the first vertex of an elimination order).

We know that all tree decompositions can be created by chordalization/elimination. During chordalization, an eliminated vertex creates a clique between itself and all of its neighbours later in the ordering, which corresponds to a bag of vertices in the tree decomposition. This means that there if there is a clique of size k then the treewidth is at least k - 1.

Now consider the first vertex, v, that is eliminated.

\[d=\text{degree}(v) \geq m, \text{ where } m \text{ is the minimum degree.}\]
If we eliminate this vertex then there is a clique of size $d + 1 \geq m + 1$, and the treewidth is at least $d + 1 - 1 = d \geq m$, since there is a clique of that size in the graph. Because any tree decomposition can be represented by elimination, one vertex must be eliminated initially.

So, the minimum degree is a simple lower bound on the treewidth. It can be improved using the following observation:

the treewidth of a minor of $G$ is at most the treewidth of $G$

**Implementation:**

We used Boost Graph Library (C++) to implement the algorithm. So iterating through all vertices to find the Min-degree vertex during each iteration is not efficient. So we use a Heap data structure to find the Min-degree vertex

**Selection of Min-degree vertex:**

We maintain a Min Heap of the vertices of the graph based on the degree. So at any time, the root of the Heap yields the Min-degree vertex in $O(1)$ time.

**Improving the lowerbound:**

At each iteration, we have a lot of vertices with same min-degree, so we have to select the best min-degree vertex to perform contraction.

**Heuristic for resolving ties among min-degree vertices:**

Check the top three positions in the Heap

If more than one vertex $v$ has same degree as root among the top three positions then

value = Minimum_number_of_shared_neighbours ($v$)

If (value < best_value)

Min_degree_vertex = $v$

Done
**Sample Results for Primal graph:**

<table>
<thead>
<tr>
<th>Instance Name</th>
<th>Number of variables</th>
<th>Number of clauses</th>
<th>Maximum clause size</th>
<th>Lowerbound</th>
</tr>
</thead>
<tbody>
<tr>
<td>eq.atree.braun.8.unsat</td>
<td>684</td>
<td>2300</td>
<td>15</td>
<td>24</td>
</tr>
<tr>
<td>AProVE09-06</td>
<td>77262</td>
<td>263137</td>
<td>74</td>
<td>131</td>
</tr>
<tr>
<td>manol-pipe-c7bidw_i</td>
<td>131004</td>
<td>387841</td>
<td>3</td>
<td>205</td>
</tr>
<tr>
<td>schup-12s-motst-2-k315</td>
<td>507145</td>
<td>1601920</td>
<td>316</td>
<td>403</td>
</tr>
</tbody>
</table>

**Inferences on results of primal graphs:**

The results show that treewidth parameter of primal graphs is not a good indicator for the hardness of a SAT problem. The treewidth is a good measure for dynamic programming type algorithms. However the behavior of modern SAT solvers is far different from that. Our results show a few easy instances, that exhibit considerably high lower bound on treewidth.

**Sample Results for Incidence graph:**

<table>
<thead>
<tr>
<th>Instance Name</th>
<th>Number of variables</th>
<th>Number of clauses</th>
<th>Maximum clause size</th>
<th>Lowerbound</th>
</tr>
</thead>
<tbody>
<tr>
<td>eq.atree.braun.8.unsat</td>
<td>684</td>
<td>2300</td>
<td>15</td>
<td>22</td>
</tr>
<tr>
<td>AProVE09-06</td>
<td>77262</td>
<td>263137</td>
<td>74</td>
<td>94</td>
</tr>
</tbody>
</table>

**Inferences on results of incidence graphs:**

The running time for incidence graphs is much larger than that of primal graphs due to large number of vertices. Also the lower bound values obtained doesn’t show significant difference from the bounds of primal graphs. Thus incidence graphs also doesn’t clearly indicate the hardness of a SAT problem.

**Lower bound based on Maximum clique:**

It is well known that treewidth of a graph does not increase under taking minors. So the treewidth of a minor H of G is a lowerbound for the treewidth of G,

\[ Tw(H) \leq th(G) \]

Since every graph induced by a clique Q in G is a minor of G,

\[ Tw(G) \geq IQI - 1 \]
The best bound derived this way is for the maximum clique. Hence the maximum clique number of a graph minus one is a lower bound for the treewidth.

**Finding a Maximum Clique**:

Finding the maximum clique is an NP Complete problem. A well known class of graphs where the maximum clique problem is polynomially solvable, is the class of perfect graphs. A graph $G$ is perfect if every induced subgraph of $G$ has the property that the size of its maximum clique equals the minimum number of independent sets needed to cover all the vertices (commonly called a coloring).

For eg, weakly triangulated graphs are perfect graphs.

**Heuristics**:

**Best in Heuristic**:

Best in heuristic constructs a maximal clique by repeatedly adding in a vertex that has the largest degree among candidate vertices.

**Worst out Heuristic**:

Worst out heuristic can start with the whole vertex set $V$. It will repeatedly remove a vertex out of $V$ until $V$ becomes a clique. This sounds good for computing the treewidth lowerbound as we can start with the input graph and keep on removing vertices until we get a clique.

**Local search Heuristics**:

Let us denote $S_G$ to be the space consisting of all the maximal cliques of $G$. What a sequential greedy heuristic does is to find one point in $S_G$, hoping it is (close to) the optimal point. This suggests us a possible way to improve our approximation solutions, namely, expand the search in $S_G$. For example, once we find a point $x \in S_G$, we can search its neighbors to improve $x$. This leads to the class of the local search heuristics. In local search heuristics, the more neighbours of $x$ we search, there is a greater chance of finding a better solution.

**Randomized Heuristics**:

In this method, we find a random point in $S_G$ and then expand the search. The selection of a random point may be based on several parameters.

**Subgraph based approach**:

It is based on the fact that the maximum clique $C$ of subgraph $G'$ of $G$, is also a clique of $G$. This method finds a subgraph $G'$ of $G$ such that the maximum clique of $G'$ can be found in polynomial time. Then it uses the maximum clique of $G'$ as an approximation solution.
Recently, **Tabu search and Neural Networks** have also been used to find an approximate solution for the maximum clique problem.

**Is Maximum clique bound better than MMD(Maximum minimum degree) method:**

Since the maximum clique of $G$ is a subgraph,

$$\text{MMD}(G) \geq \text{Maximum clique size} - 1$$

Thus Minimum degree algorithm often gives better bounds than Maximum clique based bound.

**Lower bound using Maximum clique minor**

Here we follow a heuristic similar to MMD+ and generate minors during each iteration. We terminate when we get a maximum clique minor. But we can't be sure that contracting the minimum degree vertex will definitely yield a maximum clique minor.

**Finding a maximum clique minor:**

**Chromatic number:**

The smallest number of colors needed to color a graph $G$ is called its **chromatic number**, $\chi(G)$.

**Bounds on the chromatic number**

Assigning distinct colors to distinct vertices always yields a proper coloring, so

$$1 \leq \chi(G) \leq n.$$

The only graphs that can be 1-colored are edgeless graphs and the complete graph $K_n$ of $n$ vertices requires $\chi(K_n) = n$ colors. In an optimal coloring there must be at least one of the graph's $m$ edges between every pair of color classes, so

$$\chi(G) \left( \chi(G) - 1 \right) \leq 2m.$$

If $G$ contains a clique of size $k$, then at least $k$ colors are needed to color that clique; in other words, the chromatic number is at least the clique number:

$$\chi(G) \geq \omega(G).$$

Graphs with large cliques have high chromatic number, but the opposite is not true.

**Hadwiger conjecture**

It states that, if all proper colorings of an undirected graph $G$ use $k$ or more colors, then one can find $k$ disjoint connected subgraphs of $G$ such that each subgraph is connected by an edge
to each other subgraph. Contracting the edges within each of these subgraphs so that each subgraph collapses to a single supervertex produces a complete graph $K_k$ on $k$ vertices as a minor of $G$.

The Hadwiger number $h(G)$ of a graph $G$ is the size $k$ of the largest complete graph $K_k$ that is a minor of $G$ (or equivalently can be obtained by contracting edges of $G$). It is also known as the contraction clique number of $G$. The Hadwiger conjecture can be stated in the simple algebraic form $\chi(G) \leq h(G)$ where $\chi(G)$ denotes the chromatic number of $G$.

Determining the Hadwiger number of a given graph is NP-complete but fixed-parameter tractable: there is an algorithm for finding the largest clique minor in an amount of time that depends only polynomially on the size of the graph, but exponentially in $h(G)$. Additionally, polynomial time algorithms can approximate the Hadwiger number significantly more accurately than the best polynomial-time approximation (assuming P\(\neq\)NP) to the size of the largest clique in a graph.

Lowerbound based on SAT Solving:

A lot of powerful SAT solvers are currently being developed and a lot of research is being done into making SAT solving faster. We can encode the decision problem whether a given graph $G = (V, E)$ has treewidth at most $k$ to an instance $F$ of the propositional satisfiability problem (SAT) such that $G$ has treewidth less than or equal to $k$ if and only if $F$ is satisfiable.

According to the Encoding specified in “Encoding Treewidth into SAT – Marko Samer and Helmut Veith”, SAT instances consisting of $O(k |V|^2)$ variables and $O(|V|^3)$ clauses are generated from the graph $G = (V, E)$.

But for Industrial SAT instances, the number of variables and clauses increases hugely such that it is infeasible to perform computation.

References:

[1] Robert Mateescu. Treewidth in Industrial SAT Benchmarks


