Parameterized Algorithms for Classical Problems on Graphs : Theory and Experiments

A Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Sriharsha Gaddipati
(111601005)

under the guidance of

Krithika Ramaswamy

INDIAN INSTITUTE
OF TECHNOLOGY
PALAKKAD

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**Parameterized Algorithms for Classical Problems on Graphs : Theory and Experiments**" is a bonafide work of **Sriharsha Gaddipati** (**Roll No. 111601005**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Palakkad under my supervision and that it has not been submitted elsewhere for a degree.*

Krithika Ramaswamy
Assistant/Associate Professor
Department of Computer Science & Engineering
Indian Institute of Technology Palakkad

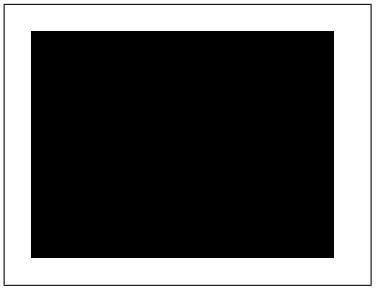
Acknowledgements

I would like to thank Krithika Ramaswamy for mentoring and helping me in successfully doing this project.

Contents

List of Figures

List of Tables



Chapter 1

Vertex Cover

Definition : Vertex cover of a graph is a set of vertices such that each edge of the graph is incident to at least one vertex of the set.

1.1 Branching Algorithm

Instance : A graph G on n vertices, m edges and integer k.
Question : Does G have a vertex cover of size atmost k?
Parameter : k

Reduction Rule 1 : Delete isolated vertex v

Resulting instance : (G-v, k)

Theorem: 1 *(G,k) is an yes-instance iff (G-v,k) is an yes-instance*

Proof:

(G,k) is an yes-instance if (G-v,k) is an yes-instance

Adding a isolated vertex v to graph G-v does not change the size of vertex cover as it does not have any edge with other vertices.

(G-v,k) is an yes-instance if (G,k) is an yes-instance

Removing a isolated vertex v to graph G does not change the size of vertex cover as it does not have any edge with other vertices.

Reduction Rule 2 : Delete high degree vertices($\geq$k)

Resulting instance : (G-v, k-1)

Theorem: 2 *(G,k) is an yes-instance iff (G-v,k-1) is an yes-instance*

Proof:

(G-v,k-1) is an yes-instance if (G,k) is an yes-instance

If G contains a vertex v of degree more than k, then v should be in every vertex cover of size at most k. So we can remove vertex v and decrease the parameter by 1.

(G,k) is an yes-instance if (G-v,k-1) is an yes-instance

If we add a vertex of degree $\geq$ k to graph, then either this vertex should be in solution or all the neighbours of v which are greater than k should be in solution. The later case is not possible. So this vertex should definitely in vertex cover. So adding a vertex of degree $\geq$ k leads to increase of parameter by 1.

Reduction Rule 3: Add neighbour of degree-1 vertices to solution

Degree-1 vertex will only have an edge with one other vertex. Either it or its neighbour needs to be in solution. Adding the neighbour to solution will be more useful as it may cover some more edges in the graph.

Reduction Rule 4 : For degree-2 vertices

Case 1 :

Theorem: 3 *If neighbours of degree-2 vertex has an edge between them, then the resulting instance is (G', k-2)*

Proof:

Let the degree-2 vertex be v and its neighbour be x and y.

Any vertex cover has at least 2 from v,x,y.

If v is included, then v covers 2 edges and to cover edge between x and y we require one of them.

If v is not included, then both of its neighbours should be included.

Resulting instance : (G-{v,x,y}, k-2)

Case 2 :

Theorem: 4 *If neighbours of degree-2 vertex does not have an edge between them, then the resulting instance is (G', k-1)*

Proof:

(G',k-1) is an yes-instance if (G,k) is an yes-instance

Let the degree-2 vertex be v and its neighbour be x and y. Let z be a vertex having edges with neighbours of x and y apart from v.

Either v is in minimum vertex cover or x and y are in minimum vertex cover.

If v is in minimum vertex cover, then both x and y will not be in minimum vertex cover. As we are not including x and y, all the neighbours of x and y will be in minimum vertex cover. So we can delete this vertex from the graph and reduce the parameter by 1.

If x and y are in minimum vertex cover, then it will cover the edges of v. So now we need to cover only neighbours of x and y apart from v. To cover these edges we can remove x and y from the graph and add vertex z

(G,k) is an yes-instance if (G',k-1) is an yes-instance

Either z in T or z is not in T.

If z is not in minimum vertex cover, then neighbours of x and y will be in solution and edges of vertex needs to be covered. So include v to solution, this will increase the parameter by only 1.

If z is in minimum vertex cover, then we can delete this vertex and include x and y to the solution. As x and y will cover all the neighbours of z and also v.

Now, let the final reduced instance be (G,k) and G has a minimum degree-3 vertex.

Find a vertex v $\in$ V(G) of maximum degree in G.

Either v or N(v) in the vertex cover.

2 branches of (G,k) are (G-v, k-1) and (G-N(v), k-$|N(V)|$).

If the reduced instance graph is empty and still k$\geq$0, then we return as yes-instance. If the k<0, then we return as no-instance.

Apply preprocessing reduction rules to the new reduced instance and continue the same process.

Running time = (number of nodes in the search tree) * (time taken at each node)

Time taken at each node is in order of n.

Number of nodes in the search tree = 2l-1, where l is the number of leaves.

So Running time is bounded by the number of leaves in the tree.

Let T(k) denote the no. of leaves in the tree rooted at instance with parameter k

$$T(K) = \left\{ \begin{array}{ll} T(k-1) + T(k-3) & if k \geq 3 \\ 1 & otherwise \end{array} \right\}$$

Total running time of the algorithm is $O(n^3 + 1.465^k * k^3)$

Chapter 2

Feedback Vertex Set

Definition: Feedback Vertex Set of a graph is a set of vertices that has at least one vertex of every cycle

2.1 Branching Algorithm

Instance : An undirected graph G and an integer k
Question : Does there exist a feedback vertex set of G of size at most k?
Parameter : k

Reduction Rule 1: Delete isolated vertex v

Resulting instance : (G-v, k)

Theorem: 5 *(G,k) is an yes-instance iff (G-v,k) is an yes-instance*

Proof:

(G,k) is an yes-instance if (G-v,k) is an yes-instance

Adding a isolated vertex v to graph G-v does not change the size of feedback vertex set as it does not from any cycle.

(G-v,k) is an yes-instance if (G,k) is an yes-instance

Removing a isolated vertex v to graph G does not change the size of vertex cover as it won't be in any cycle of graph.

Reduction Rule 2: Delete degree-1 vertex

Resulting instance : (G-v, k)

Theorem: 6 *(G,k) is an yes-instance iff (G-v,k) is an yes-instance*

Proof:

(G-v,k) is an yes-instance if (G,k) is an yes-instance

Removing a degree-1 vertex does not change the size of feedback vertex set as it won't be part of any cycle

(G,k) is an yes-instance if (G-v,k) is an yes-instance

Adding a degree-1 vertex does not change the size of feedback vertex set as it won't be part of any cycle

Reduction Rule 3: If there is a loop at a vertex v, delete v from the graph and reduce the parameter by 1

Reduction Rule 4: If there is an edge with multiplicity > 2, reduce it to 2

Reduction Rule 5: Short circuit degree-2 vertices

Reduction instance : (G',k)

Theorem: 7 *(G,k) is an yes-instance iff (G',k) is an yes-instance*

Proof:

(G,k) is an yes-instance iff (G',k) is an yes-instance

Let v be the degree-2 vertex and x,y be its neighbours.

Introducing an vertex v between x and y (v-x edge and v-y edge), does not change the size of the feedback vertex set.

(G',k) is an yes-instance iff (G,k) is an yes-instance

Any cycle through v also goes through x and y. So removing vertex v does not change the size of the feedback vertex set.

Now after applying these reduction rules, graph contains minimum degree-3.

Theorem: 8 *If G is a graph with minimum degree $\geq$ 3, then every FVS of size $\leq$ k contains at least one vertex from the top 3k vertices after sorting the vertices according to their degree.*

Proof:

Let $v_1, v_2, ..., v_n$ be the descending order of vertices according to their degree.

$$v_h = \{v_1, v_2, ..., v_{3k}\ \}$$

Let us assume that there is a feedback vertex set X of size at most k such that $X \cap v_h = \emptyset$.

Graph F = G-X is a forest

Number of edges in F is atmost $|V(G)| - |X| - 1$.

Every edge of E(G)-E(F) is incident to a vertex of X.

$$\Sigma_{v \in X} d(v) + |V(G)| - |X| - 1 \geq |E(G)|$$

$$\Sigma_{v \in X}(d(v) - 1) \geq |E(G)| - |V(G)| + 1$$

For every v $\in$ X, d(v) is at most the minimum of vertex degrees from v_h

$$\Sigma_{i=1}^{3k}(d(v_i) - 1) \geq 3 * (\Sigma_{v \in X}(d(v_) - 1) \geq 3 * (|E(G)| - |V(G)| + 1)$$

$$\Sigma_{i>3k}(d(v_i) - 1) \geq \Sigma_{v \in x}(d(v) - 1)$$

$$\Sigma_{i=1}^{n}(d(v_i) - 1) \geq 4 * (|E(G)| - |V(G)| + 1)$$

$$\Sigma_{i=1}^{n}(d(v_i)) = 2|E(G)|$$

$$2|E(G)| - |V(G)| \geq 4 * (|E(G)| - |V(G)| + 1)$$

$$2|E(G)| < 3|V(G)|$$

This contradicts the fact that every vertex of G is of minimum degree 3.

From these 3k vertices, at least one will be in solution. So we have 3k branches each branch choosing one vertex from the v_h.

Reduced instance: (G-v, k-1)

If the reduced instance graph is empty and still k≥0, then we return as yes-instance. If the k<0, then we return as no-instance.

Total running time of the algorithm is $O^*((3k)^k)$

2.2 Iterative Compression Algorithm

Iterative compression is an algorithmic technique for the design of fixed-parameter tractable algorithms, in which one element (such as a vertex of a graph) is added to the problem in each step, and a small solution for the problem prior to the addition is used to help find a small solution to the problem after the step.

1. Start with a subgraph induced by a vertex set S of size k, and a solution X that equals S itself.

2. While S ≠ V, perform the following steps:

   - Let v be any vertex of V-S, and add v to S
   - Test whether the (k + 1)-vertex solution Y = X ∪ {v} to S can be compressed to a k-vertex solution.
   - If it cannot be compressed, abort the algorithm: the input graph has no k-vertex solution.
   - Otherwise, set X to the new compressed solution and continue the loop.

Instance: Given a graph G and k+1 size solution S
Question: Does there exist a feedback vertex set of G of size at most k?
Parameter : k

Let the solution set of size atmost k be R.

Divide the set s into 2 parts. One part which contains the vertices from R and other part X which does contain vertices from R.

Total there are 2^{k+1} choices of S $\cap$ R

Let r number of vertices be in X.

As k+1-r are already in solution, now our aim is to find a set of at most r-1 vertices from Y (i.e. G-S).

Reduction rule 1: Delete all the vertices of degree at most 1 in G

Reduction rule 2: If there exists any vertex v in Y such that G(X $\cup$ v) contains a cycle, then include this vertex in the solution, remove this vertex and decrease the parameter by 1.

Reduction rule 3: If there is a vertex v in Y of degree 2 in G such that at least one neighbor of v in G is from Y, then delete this vertex and make its neighbors adjacent.

Since Y is a forest, Y has a vertex v of degree at most 1.

v has ≥ 2 neighbours in Y, otherwise above reduction rules would have been applied. And also those two neighbours will be in different components of Y, otherwise above reduction rule would have been applied.

Here we have two possibilities. Either v is in solution or v is not in solution. Branch into these possibilities.

If v is in solution, then the reduced instance is (X,Y-$\{v\}$,r-2)

If v is not in solution, then the reduced instance is (X $\cup\{v\}$,Y-$\{v\}$,r-1)

In one branch, r decreases by 1 and in another branch connected components of x decrease by 1.

Let I = r + #C(X), where #C(X) is the number of connected components of X.

$$I \leq k+1+k \leq 2k+1$$

Let T(I) denote the number of leaves in the tree rooted at instance with measure I.

$$T(I) = \left\{ \begin{array}{ll} 2T(I-1) & if I \geq 1 \\ 1 & otherwise \end{array} \right\}$$

Total running time for disjoint compression algorithm is $O^*(2^{2k})$

If there exists an algorithm solving Disjoint Feedback Vertex Set in time g(k)n , then there exists an algorithm solving Feedback Vertex Set in Compression in time $\Sigma_{i=0}^{k}\binom{k+1}{i}g(k-i)n^{O(1)}$

If g(k) = 4^k, then the running time of the algorithm is $O^*(5^k)$

2.3 Randomized Algorithm

Instance: Given a graph G and a integer k
Question: Does there exist a feedback vertex set of G of size at most k?
Parameter : k

Reduction Rule 1: Delete isolated vertices

Reduction Rule 2: Delete degree-1 vertices

Reduction Rule 3: If there is a loop at a vertex v, delete v from the graph and reduce the parameter by 1

Reduction Rule 4: If there is an edge with multiplicity > 2, reduce it to 2

Reduction Rule 5: Short circuit degree-2 vertices

After applying the reduction rules, graph contains minimum degree 3.

Theorem: 9 *If G is graph with minimum degree ≥ 3, then number of edges incident to any feedback vertex set S is* $> |E(G)|/2$

Proof:

Let feedback vertex set of G be S and H = G-S

Let E(X) be edges incident to S, E(G) be edges in graph G, E(H) be edges in subgraph H, E(H,S) be edges between H and S.

We need to prove that |E(X)| > |E(G |/2

$|E(X)| > |E(G)|/2$ is equivalent to $|E(X)| > |V(H)|$

$2|E(X)| > |E(G)|$

$2|E(X)| > |E(X)| + |E(H)|$

$|E(X)| > |E(H)|$

$|E(X)| > |V(H)|$

Let $V_{\leq 1}$, V_2 and $V_{\geq 3}$ denote the set of vertices in V(H) such that they have degree at most 1, exactly 2, and at least 3 in H respectively.

$|E(X)| \geq |E(H, S)|$

$|E(X)| \geq 2|V_{\leq 1}| + |V_2|$

$|E(X)| > |V_{\leq 1}| + |V_2| + |V_{\geq 3}|$

$|E(X)| > |V(H)|$

Feedback Vertex Set can be solved in randomized polynomial time, with success probability at least 4^{-k}

Run the following algorithm 4^k times. If none of the executions return yes, then declare that (G,k) is a no-instance. Other wise, declare that (G,k) is an yes-instance.

Initialize the solution set to empty. Then apply the above mentioned reduction rules to the graph. Now graph will have minimum degree 3. Pick a edge uniformly at random from the edges in the reduced instance. Now, pick a endpoint from the edge picked previously uniformly at random. Add this endpoint to the solution set and delete that from the reduced instance. And continue the process till our parameter is positive. Now check if the solution set obtained from the algorithm is a feedback vertex set or not. If yes, we return it as solution.

Total running time of the algorithm is $O^*(4^k)$

Proof:

If (G,k) is a no-instance then algorithm always outputs no.

Suppose (G,k) is an yes-instance and F is a $\leq$ k feedback vertex set.

Let H=G-F

Pr (an edge in E(F) $\cup$ E(H,F) is chosen) $> 1/2$

Pr (a vertex from F is chosen) $> 1/2 * 1/2 = 1/4$

Pr(S=F) $> (1/4)$k

Pr(Algorithm says no) $< (1 - (1/4)^k)^{(4)^k} \leq 1/e$

Pr(Algorithm says yes) $> 1 - 1/e \geq 1/2$

Chapter 3

Tree Decomposition

Definition : A tree decomposition of a graph G is a pair (T,B) where T is a tree and $B : V(T) \rightarrow 2^{V(G)}$ satisfies the following

- For each vertex v in G, there is a node x in V(T) such that v is in B(x)
- For each edge e={u, v} in G, there is a node x in V(T) such that u are v are in B(x)
- For each vertex v in G, the set {x ∈ V(T) : v ∈ B(x) } induces a connected graph

3.1 Simple Tree Decomposition

Definition : A simple tree decomposition (T,B) is a tree decomposition in which there is no pair of distinct nodes x and y in T such that B(x) ⊆ B(y)

3.1.1 Compute a simple tree decomposition

Question : Given a graph G and a tree decomposition pair (T,B) where T is a tree and B $B : V(T) \rightarrow 2^{V(G)}$, find a simple tree decomposition (T',B') of G

First, we have to verify whether the given tree decomposition is a tree decomposition or not.

1. To verify the first rule of tree decomposition, check for every vertex in graph if there is a bag that contains the vertex. If its not true for every vertex, then it is not a tree decomposition.

2. To verify the second rule of tree decomposition, check for each edge in graph if there is a bag that contains both the end points of the edge. If its not true for every edge, then it is not a tree decomposition.

3. To verify the third rule of tree decomposition, for every vertex v, we do the following checking. First we mark all the bags that contains v. Then we will start a modified breadth first search from any arbitrary node, which will only explore the children which contains v and mark these vertices as visited. Otherwise we ignore the vertex. Once the search is finished, if the visited vertices are same as the vertices which we marked initially then we say that the rule 3 is followed.

If it is a tree decomposition, we now further verify if its a simple tree decomposition or not.

1. To verify the rule of simple tree decomposition, check if there are any bags such that one is the subset of the another.

If there are no such bags, then we say that the given tree decomposition is a simple tree decomposition Otherwise we remove a node from the tree which has the bag that is subset of other bag and add the edges between the children of removed vertex and the superset node.

Total running time of the algorithm is $O(n^{\alpha})$

3.2 Optimal Tree Decomposition

Width of a tree decomposition T = w(T) = max { $|B(x)|$: x ∈ V(T)} - 1

Treewidth of G, tw(G) = min {w(T) : T is a tree decomposition of G}

Definition : An optimal tree decomposition of a graph G is a tree decomposition of G of width tw(G)

3.2.1 Compute an optimal tree decomposition

Question : Given a graph G, compute an optimal tree decomposition of G

To find the optimal tree decomposition, we enumerate on the all the possible tree decomposition pairs and find the minimum width.

We start the size of the tree T from 1, and increase till V(G). For a particular size n of the tree, a tree consists of n-1 edges. We need to pick n-1 edges which forms a tree from the n*(n-1)/2 edges of the complete graph consisting of n vertices. So total atmost $\binom{(n*(n-1)/2)}{n-1}$ number of tree combinations are possible.

For a fixed tree, we have at most $2^{(n^2)}$ choices of B.

After choosing a pair of (T,B), we verify whether it is a tree decomposition. If yes, then we will update minimum treewidth if the new width value is minimum.
Total running time of the algorithm is $2^{O(n^2)}$

3.3 Dominating set

Definition : A dominating set in Graph G is a set S of vertices such that N(S)=V(G) or every vertex not in S is adjacent to at least one member of S.

3.3.1 Dominating set on Trees

Instance : A tree T and an integer K Question : Does there exist a dominating set of T of size at most K?

To solve this problem we use bottom up approach of dynamic programming. We solve the subparts of the problem and use them to get the final solution.

Root T at an arbitrary vertex. Let that root vertex be **root**

For a vertex v, let T_v denote the subtree of T rooted at v

- Let $\Gamma(v)$ denote the minimum possible size of a dominating set in Tv
- Let $\Lambda(v)$ denote the minimum possible size of a dominating set in T_v that dominates every vertex in $T_v - v$
- Let $\Delta(v)$ denote the minimum possible size of a dominating set in T_v that contains v

$\Gamma(root)$ is the final solution which we need. To get this value, we start solving for the $\Gamma(v)$, $\Lambda(v)$, $\Delta(v)$ values from the leaf nodes, which will be used further as we go till root of the tree.

If v is a leaf, then

$$\Gamma(v) = 1$$

$$\Lambda(v) = 0$$

$$\Delta(v) = 1$$

If v has $v_1, v_2, ..., v_q$ as its children, then

$$\Delta(v) = 1 + \Lambda(v_1) + \Lambda(v_2) + ... + \Lambda(v_q)$$

$\Delta(v)$ definitely contain v in dominating set, so we need not cover children of v. But we should cover all the grand children of v.

$$\Lambda(v) = \min\{\ \Gamma(v_1) + \Gamma(v_2) + ... + \Gamma(v_q),\ 1 + \Lambda(v_1) + \Lambda(v_2) + ... + \Lambda(v_q)\ \}$$

$\Lambda(v)$ may or may not contain v in dominating set. If we include v in solution, then we need to cover all the grand children of v. If we dont include v in solution, then we should cover all the children and grand children of v.

$$\Gamma(v) = \min\{\ 1 + \Lambda(v_1) + \Lambda(v_2) + ... + \Lambda(v_q),\ \min\{\Delta(v_i) + \Sigma_{i \neq j}\Gamma(v_j) : i \in [q]\ \}\ \}$$

Total running time of the algorithm is O(n).

Chapter 4

Conclusion and Future Work

- To find a appropriate kernel using 2-expansion lemma for feedback vertex set.
- Analyzing the performance when instead of doing iterative compression algorithm directly to the input instance, find a good kernel of the instance and then apply iterative compression algorithm.
- Minimum dominating set problem can be parameterized by the treewidth. Vertex cover of a graph is a dominating set but it may not be the minimum. Finding if it can be parameterized by the vertex cover size.
- Implement a good approximation algorithm for feedback vertex set.

References

[1] *Parameterized Algorithms by Cygan et al. and Kernelization by Fomin et al.*