A Study on Directed Feedback Vertex Set (DFVS)

Soumyajit Karmakar

April 2022

Abstract

A Feedback Vertex Set (FVS) for an undirected graph $G(V, E)$, is a set of vertices $S \subseteq V$, such that $G \setminus S$ does not have any cycles. In this report, we study different algorithms (parameterized and exact) for the Feedback Vertex Set problem for Directed Graphs (DFVS). The parameterized version of DFVS is defined as follows: given a directed graph G and a parameter k, our task is to either return a feedback vertex set whose size is bounded by k, or report that no such set exists. This problem can be shown to be in the NP-complete class, while its unparameterized version, where the problem is, given a directed graph G either return a feedback vertex set or report no such set is possible, falls in the class of NP-Hard. The current state-of-the-art algorithm for finding the exact solution to the parameterized DFVS problem has a running time complexity of $4^k!n^{O(1)}$, where n is the number of nodes in the graph [1]. The main contribution of this report is implementation and analysis of the aforementioned algorithm.

1 Introduction

The problem of DFVS appears in a variety of applications. One of the applications is resolving deadlocks in software such as operating systems and database systems [1]. A deadlock is a scenario where no member of some group of entities can proceed because each waits for another member to take action. This problem can be modeled as a directed graph with each vertex representing the entities and each edge representing the dependencies,

a deadlock in this case would form a cycle in the graph. Other important application is found in testing circuits [1], where the vertices represent gates and the directed edges represent wires that connect the gates. Recognizing feedback sets here help in reducing the hardware overhead required in testing the circuit.

A Graph is a mathematical structure which is used to store information about connections or relations, called edges, among entities, called vertices. Depending on the nature of these connections graphs can be of two types Directed or Undirected. In Undirected Graphs information can flow in both directions of an edge while in Directed Graphs all edges have a specified direction, and flow of information is only allowed in that direction only.

2 Definitions and Notations

2.1 Graph Theoretic Definitions

Let G be a directed graph with vertex set $V(G)$ and edge set $E(G)$. In this report. We will use E and $E(G)$, and similarly V and $V(G)$, interchangeably depending on the context. A *path* in a graph G of length k, denoted by P is defined as $E(P) = \{v_i, v_{i+1}, 1 \leq i \leq k-1, v_i \in V\}$. A *cycle* in a graph G of length k, denoted by C is defined as $E(C) = \{(v_i, v_{i+1}), 1 \leq i \leq k-1, v_i \in V\} \cup (v_k, v_1)$.

Let a directed graph be represented as $G = (V, E)$, where V is the set of vertices and E is the set of edges in the graph. Let an edge be represented as $e = [u, v]$, The edge e is called an outgoing edge of the vertex u, and an incoming edge of the vertex v.

For a vertex set S, we say that an edge goes out from S if the edge goes out from a vertex in the set S and comes into a vertex not in S. Similarly, we say an edge comes into S if the edge goes out from a vertex not in S and comes into a vertex in S.

For a vertex set $V' \subseteq V$ in the directed graph $G = (V, E)$, the graph induced by the vertices of V' is denoted by $G[V']$. Therefore, $G[V \setminus V']$ denotes the graph induced by the vertices of $V(G) \setminus V'$. Similarly, $G[V \setminus w]$ denotes the graph induced by the vertices of $V(G) \setminus w$. A set $F \subseteq V(G)$ in the directed graph G is a DFVS if the graph $G[V \setminus F]$ is a directed acyclic graph (DAG).

2.2 Algorithmic Definitions

The NP class of problems include all the problems whose solution can be obtained in polynomial time using a nondeterministic Turing machine, but a solution to such problem can be verified in polynomial time by a deterministic Turing machine.

A problem L is in NP-Complete class if L is in NP class and every problem in NP is reducible to L in polynomial time. A problem A is reducible to problem B, if an algorithm for solving problem B can also be used as a subroutine to solve problem A. A problem L is in NP-Hard class if L if every problem in NP is reducible to L in polynomial time and it is not in the NP class.

The problem of DFVS can be reduced from the well-known Vertex Cover problem by Karp [2], thus it falls in the class of NP-Hard problems, the decision problem of DFVS falls in the NP-Complete class.

3 Literature Survey

Since DFVS is NP-Hard, the question of existence of a parameterized algorithm remained open for a few decades. Chen et al. proposed a fixed parameterized algorithm for DFVS [1] and thus solved the aforementioned open question. The algorithm proposed by Chen et al. has a running time of $O(4^{\mathrm{k}}k!n^{\mathrm{O(1)}})$.

The algorithm for the *DFVS* problem is obtained through careful development of algorithms for a series of problems. The first important problem is the *Skew Separator* problem. Here, given a directed graph G, a parameter k and collection of "Source Sets" $[S_1, S_2, ..., S_{\mathrm{l}}]$ and "Sink Sets" $[T_1, T_2, ..., T_{\mathrm{l}}]$, such that (1) all the sets $S_1, ..., S_{\mathrm{l}}, T_1, ..., T_{\mathrm{l}}$ are pairwise disjoint, (2) for each $i, 1 \leq i \leq l-1$, there is not edge coming into the set S_{i}, for each $j, 1 \leq j \leq l$, there is no edge going out from the set T_{j}, we need to either find a set of vertices, called a Skew Separator X, of at most k vertices such that for any pair of indices i and j satisfying $l \geq i \geq j \geq 1$, there is no path from S_{i} to T_{j} in the graph $G[V \setminus X]$ or report that no such set exists. The authors discussed an algorithm which solves the problem in time $O(4^{\mathrm{k}}kn^3)$.

Using the above algorithm they go on to solve another problem called the *D*irected *A*cyclic *G*raph *B*ipartition *F*VS problem. A bipartition refers to two independent sets, U and V such that every edge (u, v) either connects a

vertex from U to V or a vertex from V to U. In this problem given a directed graph G and a DAG-bipartition (D_1, D_2) for G and a parameter k, we need to either find a D_1-FVS with size bounded by k for the graph G, or report that no such set exists. A vertex subset F in the graph G is a D_1-FVS if F is an FVS for G and $F \subseteq D1$. The algorithm for this problem has a running time of $O(4^k kn^3 h!)$, where h is the number of vertices in the set $D2$.

Using the above results the authors first solved a more restricted version of the original problem, the DFVS Reduction problem. Here, given a directed graph G, a parameter k and a FVS set of size $k+1$, our task is to construct an FVS of size bounded by k for the graph G or report that no such FVS exists. The algorithm for this problem has a run time of $O(n^3 4^k k^3 k!)$.

The original $DFVS$ problem can be solved using the above algorithm by applying the Iterative Compression Method, as proposed by Reed et al. [3], this method has been used for solving the FVS problem on undirected graphs by Dehne et al [4]. Here we start with a trivial FVS solution on a subset of the graph and slowly add other vertices to it, at each step trying to find a new FVS bounded by k or reporting that no such set exists, until we have covered the entire graph. The time complexity for this algorithm is $O(n^4 4^k k^3 k!)$.

The relationship between the DFVS problem and multi-cut problems has been studied in the research of approximation algorithms for the Feedback Vertex Set problem by Even at el. [5]. This paper introduces approximation algorithms for the FVS problem in directed graphs. This approximation algorithm can be used with the algorithm proposed by [1] to further improve the performance.

4 Implementation Details

The entire project is done in `Python`. The graph structure is implemented using the `Networkx` library.

The source code for the entire project along with the report can be found here.

The following are the utility functions which are used while implementing the main algorithms:

checkEdge(G, S, T) →bool This function takes a directed graph G and two vertex sets S and T as input and returns if there exists an edge

between them. We may also pass individual vertices in place of the vertex sets to check if there is any edge between them.

findMinCut(G, S, T) →list(int) This function takes a directed graph and two vertex sets, source set S and sink set T as input and returns a list of nodes that make up the Minimum Cut for these two sets. A minimum cut is defined as a set of vertices (integers) which when removed disconnect the Source Set and the Sink Set.

checkForW(G, allS, allT) →int This function takes a directed graph G and two sets of vertex sets, all the source sets $allS$ (ie., $[S_1, S_2, ..., S_l]$) and all the sink sets $allT$ (ie. $[T_1, T_2, ..., T_l]$) as input and returns a vertex (integer) w, such that (1) w is not in any of the source sets or sink sets, (2) there exists an edge from S_l to w and (3) there exists an edge from w to any of the sink sets. If such vertex does not exist then it returns -1 (default error value).

getExtendedVertex(G, allS, allT) →int This function takes a directed graph G and two sets of vertex sets, all the source sets $allS$ (ie., $[S_1, S_2, ..., S_l]$) and all the sink sets $allT$ (ie. $[T_1, T_2, ..., T_l]$) as input and returns an Extended Vertex u0 (integer) for S_l. An extended vertex, u_0, has the following properties: (1) u_0 is not in any of the source sets or sink sets, (2) there exists an edge from S_l to u_0 and (3) there does not exist any edge from u_0 to any of the sink sets. If such vertex does not exist then it returns -1 (default error value).

checkFVS(G, X) →list(int) This function takes a directed graph and a vertex set, X, and return a list of all cycles in G after removing the vertex set X from it. This function is primarily used to check if X is a FVS for the graph G, since if it is then a null set is obtained.

The following are the core functions which are used while implementing the main algorithm:

SMC(G, S, T, k) →list(int) This function takes a directed graph G, two sets of vertex sets, all the source sets S (ie., $[S_1, S_2, ..., S_l]$) and all the sink sets T (ie. $[T_1, T_2, ..., T_l]$) and a parameter k as input and returns either the skew separator set, X, whose size is bounded by k, or the default error value of -1, if no such set is possible.

DBF(G, D1, D2, k) →list(int) This function takes a directed graph G, two vertex sets D_1 and D_2, where (D_1, D_2) is a DAG-bipartition of G and a parameter k as input and returns a FVS X such that $X \subseteq D1$ and size of X is bounded by k. It returns the default error value of -1 if no such set exists.

DFVSR(G, F, k) →list(int) This function takes a directed graph G, a FVS set F of size $k+1$ and a parameter k as input and return a FVS set for G whose size is bounded by k or the default error value -1 is a FVS set of size k is not possible for graph G.

DFVS(G, k) →list(int) This function takes a directed graph G and a parameter k as input. This function first considers a trivial FVS for a subset of G and iteratively runs the $DFVSR()$ function while adding other vertices of G which are not included in the current subset. Finally it returns either a FVS set whose size is bound by k or the default error value of -1 if no such set exists, thus solving the original problem.

5 Conclusion and Future Work

In this project we have analyzed and implemented the foundation of the algorithm to find the exact solution of the parameterized DFVS problem, with the runtime of $O(n^4 4^k k^3 k!)$, which is significantly faster than the previous exponential runtime algorithms. This algorithm can be further improved by using the approximation algorithm as discussed in [5] to get the initial FVS for a subset and then iteratively adding more nodes to it. In future, we plan to implement this idea and analyse the results.

One of the motivations for taking up this project was the PACE challenge 2022. The Parameterized Algorithms and Computational Experiments Challenge (PACE) was conceived in Fall 2015 to deepen the relationship between parameterized algorithms and practice. It is an organisation that hosts algorithmic problems each year for everyone around the world to compete. This year the challenge is to find a FVS set for a directed graph. We plan on improving on the foundation we have built in this project in order to participate this year.

References

[1] J. Chen, Y. Liu, S. Lu, B. O'sullivan, and I. Razgon, "A fixed-parameter algorithm for the directed feedback vertex set problem," in *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pp. 177–186, 2008.

[2] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*, pp. 85–103, Springer, 1972.

[3] B. Reed, K. Smith, and A. Vetta, "Finding odd cycle transversals," *Operations Research Letters*, vol. 32, no. 4, pp. 299–301, 2004.

[4] F. Dehne, M. Fellows, M. Langston, F. Rosamond, and K. Stevens, "An o (2o (k) n3) fpt algorithm for the undirected feedback vertex set problem," *Theory of Computing Systems*, vol. 41, no. 3, pp. 479–492, 2007.

[5] G. Even, B. Schieber, M. Sudan, *et al.*, "Approximating minimum feedback sets and multicuts in directed graphs," *Algorithmica*, vol. 20, no. 2, pp. 151–174, 1998.