# Feedback vertex set using Edge Density and REmove Redundant (FEDRER): A heuristic solver for finding a feedback vertex set in a directed graph

**Aman Jain**
Department of Computer Science and Engineering, IIIT Guwahati, India

**Sachin Agarwal**
Department of Computer Science and Engineering, IIIT Guwahati, India

**Nimish Agrawal**
Department of Computer Science and Engineering, IIIT Guwahati, India

**Soumyajit Karmakar**
Department of Computer Science and Engineering, IIIT Guwahati, India

**Srinibas Swain**
Department of Computer Science and Engineering, IIIT Guwahati, India

**Abstract**

Feedback vertex set using Edge Density and REmove Redundant (FEDRER) is a novel heuristic solver for finding a feedback vertex set in a directed graph (DFVS). The solver can be accessed through this **link**[1]. In FEDRER, we first remove all the redundant vertices, that is, the vertices which can never be a part of any optimal DFVS via various known reduction techniques. In the process, we also obtain some vertices which belong to one of the optimal DFVS. Then we decompose the reduced graph into several strongly connected components and compute the union of the DFVSs (say $D$) of each of the components respectively. Finally, we remove vertices from $D$, such that $D$ is still a DFVS of $G$.

**Keywords and phrases** Divide and Conquer, Depth First Search, Edge Density and Redundancy Removal

## 1 Definitions

We used the following definitions and notations in FEDRER. Let $G(V, E)$ be a directed graph. If $(u, v) \in E(G)$, then $u$ is a *predecessor* of $v$ and $v$ is a *successor* of $u$. An edge $(u, v)$ is called *PIE-edge* iff both $(u, v)$ and $(v, u) \in E(G)$. The *neighbour* of a vertex $v \in V(G)$, denoted as $N(v)$ is defined as $\{u : (v, u)$ is a PIE-edge$\}$. A vertex $u \in V(G)$ is a *PIE-vertex* if all edges incident on $u$ are PIE-edges. The *edge density* of $u \in V(G)$ is the product of its in-degree and out-degree. A vertex $v \in V(G)$ is a *critical node* if it has the maximum edge density among all the vertices of $G$. A *critical density sequence* $S$ is a sequence of vertices of $G$ arranged in the descending order of their respective edge densities. A *critical node list* $L \subseteq V(G)$ of order $k$ is the list of first $k$ vertices from $S$.

A subgraph $H \subseteq G$ is a *strongly connected component* (SCC) if there exists a path between any two vertices of $H$. Let $H_1, H_2$ be two SCCs of $G$. An edge $(u, v) \in E(G)$ is called an *acyclic edge*, if $u \in V(H_1)$ and $v \in V(H_2)$, that is, $(u, v)$ is not a part of any cycle in $G$. We use *d-clique* to denote a clique in a directed graph. A vertex $v \in V(G)$ is a *core vertex* if $v$

[1] DOI for the code is available at: zenodo.

is a PIE-vertex and $N(v) \cup \{v\}$ forms a d-clique. A cycle $C_1 =< v_1, v_2, \cdots, v_n, v_1 >$ is said to be *covered* by a cycle $C_2 =< u_1, u_2, \cdots, u_k, u_1 >$ if each vertex $v_i, i = 1, \cdots, n$, in $C_1$ is contained in $C_2$. A cycle is *minimal* if it does not cover any other cycle. Given an edge ($u$, $v$), if ($u$, $v$) is a *PIE-edge*, then the vertex $u$ is called *PIE-predecessor* of the vertex $v$ and the vertex $v$ is called *PIE-successor* of the vertex $u$. Otherwise, the vertex $u$ is called *non-PIE predecessor* of the vertex $v$ and the vertex $v$ is called *non-PIE successor* of the vertex $u$. An edge $(u, v)$ is a *dominated edge* if one of the following condition holds:

- Set of non-PIE predecessor of $u$ is a subset of all predecessor of $v$.
- Set of non-PIE successor of $v$ is a subset of all successors of $u$.

## 2 FEDRER

We use greedy, divide and conquer strategies in FEDRER. Our approach is divided into three major phases:

- Size Reduction
- Divide and Conquer
- Remove Redundant

### 2.1 Size Reduction

We use a collection of data reduction rules from [3, 4] to reduce the size of the input graph. The reductions used in FEDRER is implemented in three phases, namely: *Levy and Low, PIE reduction and core reduction.* If $|E(G)| \leq 10^6$ we additionally use *dome reduction* defined in [4]. To be self-contained, we now give a brief description of the aforementioned reduction rules. Refer to [3, 4] for a more thorough discussion on these reductions, including implementation details. The first five most straight forward reductions are described by Levy and Low in [3]:

**LOOP**$(v)$ If $v$ has a loop, it must be a member of any DFVS of $G$. Thus, $G$ is transformed into $G - v$ as for any DFVS (say) $D'$ of $G - v$ the set $D' \cup v$ is a DFVS of $G$.

**IN0**$(v)$ If $v$ is loop-free and has indegree 0, it cannot be a part of any cycle. Hence, $G$ is transformed into $G - v$ since both digraphs have the same minimal DFVSs.

**OUT0**$(v)$ If $v$ is loop-free and has outdegree 0, $G$ is transformed into $G - v$ with the same argumentation as in IN0(v).

**IN1**$(v)$ If $v$ is loop-free and has indegree 1, it has a unique predecessor $u$. Merge $v$ into $u$ as a single vertex.

**OUT1**$(v)$ If $v$ is loop-free and has outdegree 1, it has a unique successor $u$. Merge $v$ into $u$ as a single vertex.

Another reduction rule that decreases the cardinality of V is presented in [4].

**CORE**$(v)$ If $v$ is a core vertex, then $N(v)$ is part of an optimal DFVS. Hence, $G$ is transformed into $G - N(v)$ as any optimal DFVS $D'$ of $G - N(v)$ yields the optimal DFVS $D' \cup N(v)$ of $G$.

There are reductions presented in [4] which cause a decrease of $E(G)$ only.

**PIE reduction** For the sake of clarity, the set of *PIE-edges* in $G$ are called *PIE*. According to this reduction rule, the *acyclic edges* in the $G - PIE$ can be removed safely.

**DOME reduction** This rule states that all cycles that contain a *dominated edge* are not *minimal* and thus, the *dominated edges* can be safely removed from $G$.

## 2.2 Divide and Conquer

After applying the reduction rules described in Section 2.1, we then apply the divide and conquer strategy represented in the following steps:

- step 1: $DFVS = \{\}$
- step 2: Decompose the reduced graph into a dis-joint union of strongly connected components using Kosaraju's algorithm [1]. Let $S_l$ be the list of SCCs after the decomposition.
- step 3: For each SCC in $S_l$:
  - step 4: we remove a list of critical nodes $L$ from the SCC.
  - step 5: $DFVS = DFVS \cup L$
- step 6: Apply the reductions of Section 2.1 on each SCC of $S_l$.
- step 7: Apply steps $2 - 7$ on each SCC of $S_l$, obtained in Step 6, until each SCC is of order 1.
- step 8: Return $DFVS$

## 2.3 Remove Redundant

In this phase, we try to optimise the DFVS formed in Section 2.2. The optimisation is achieved via two steps:

- **DFVS refinement:** Since FEDRER is a heuristic solver, so sometimes the $DFVS$ obtained from Section 2.2 may not be a minimum DFVS. Therefore, we randomly add $k$ vertices (based on the size of the input graph) from $V(G) - DFVS$ to $DFVS$. We then reduce the size of the $DFVS$ in the DFVS reduction step.
- **DFVS reduction:** Let $DFVS$ be the set obtained from the DFVS refinement step. We then remove the redundant vertices from $DFVS$ using Algorithm $B$ of [2].

We repeatedly apply the DFVS refinement and DFVS reduction steps on $DFVS$ until we exhaust the ETL (end of time limit). We return the $DFVS$ with minimum cardinality obtained in this process.

## References

1 Paul B Callahan and S Rao Kosaraju. A decomposition of multidimensional point sets with applications to $k$-nearest-neighbors and $n$-body potential fields. *Journal of the ACM (JACM)*, 42(1):67–90, 1995.

2 Berend Hasselman. An efficient method for detecting redundant feedback vertices. Technical Report CPB Discussion Paper 29, CPB Netherlands Bureau for Economic Policy Analysis, 2004.

3 Hanoch Levy and David W Low. A contraction algorithm for finding small cycle cutsets. *Journal of algorithms*, 9(4):470–493, 1988.

4 Hen-Ming Lin and Jing-Yang Jou. On computing the minimum feedback vertex set of a directed graph by contraction operations. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 19(3):295–307, 2000.