# Efficient implementation of Vertex Cover (Exact) - PACE 2019

**P. R. Vaidyanathan**
Department of Computer Science, Indian Institute of Technology Gandhinagar
pr.vaidyanathan@iitgn.ac.in

**Chinmay Sonar**
Department of Computer Science, Indian Institute of Technology Gandhinagar
sonar.chinmay@iitgn.ac.in

## Abstract

The challenge for this iteration of PACE was to implement a Vertex Cover Solver, and evaluate the implementation on provided real-world instances. We have developed a Vertex Cover Solver(7) in Python using the `networkx` module(4) for handling and manipulating graphs.

**2012 ACM Subject Classification** Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Fixed parameter tractability; Mathematics of computing → Graph algorithms

**Keywords and phrases** Vertex cover, Reduction rules

**Supplement Material** github.com/aditya95sriram/python-vc, doi:10.5281/zenodo.3236977

## 1 Introduction

Vertex cover has a rich literature and is one of the first problems to be analyzed in the parameterized setting (2), (5). Several attempts have been made to evaluate the performance of heuristics on difference classes of graphs (1).

Our overall approach is to first break the graph into its connected components and then get a good estimate for the range of the size of the optimal Vertex Cover and finally binary search over this range applying the basic vertex branching FPT algorithm at each probe of the binary search. Further we apply reduction rules in between each of the aforementioned steps so as to minimize the computational load on the branching algorithm by aggressively pruning the graph beforehand.

## 2 Notations

We denote an instance of vertex cover problem as $\mathcal{I} = (G, k)$. Here, $G = (V, E)$ is a graph with vertex set $V$ and edge set $E$, and $k$ is the bound on the size of the vertex cover.

For a vertex $v$, $G - v$ denotes the graph with vertex set $V \setminus v$, and edges $E \setminus e_v$ where $e_v$ is the set of edges incident on $v$. Similarly, we define $G - S$ for $S \subseteq V$.

## 3 Reduction Rules

In this section, we thoroughly discuss the reduction rules we employed in our solver, and provide citations to the original sources of these rules. We majorly rely on commonly known reduction rules from theoretical sources and from available online implementations. In some cases, we extend these rules to cover further empirically relevant cases. We now describe the reduction rules from our solver.

Our first three reduction rules are from Parameterized Algorithms textbook (3):

▷ **Reduction rule 1.** If $G$ contains isolated vertex $v$, we remove $v$ from $G$. The new instance is $(G - v, k)$.

▷ **Reduction rule 2.** If $G$ contains degree one vertex $v$, we remove $v$ from $G$, and add $N(v)$ to VC. The new instance is $(G - (\{v\} \cup N(v)), k - 1)$.

▷ **Reduction rule 3.** If there is vertex $v$ with degree at least $k + 1$, then we add $v$ to the cover and decrement the parameter $k$ by 1. The new instance is $(G - v, k - 1)$.

We added the following reduction rule which eliminates *all* degree two vertices in $G$ from the publicly available implementation of VC in SageMath (6), more specifically from here.

▷ **Reduction rule 4.** If $G$ contains vertex $v$ with degree two, we either delete $v$ and add $N(v)$ to the VC or 'fold' it to form a higher degree vertex depending on whether the two neighbors of $v$ have an edge between them. The parameter of new instance is $k - 2$.

Apart from the rules above, we add the following rule which handles one type of degree three vertices.

▷ **Reduction rule 5.** If $G$ contains vertex $v$ with degree three, and $v$ belongs to a four-clique in $G$, we remove $v$ from G, and add $N(v)$ to VC. The new instance is $(G - (\{v\} \cup N(v)), k - 3)$.

Note that, apart from Reduction rule 3 every other reduction rule is parameter independent (i.e. the information regarding the budget $k$ is not necessary)

## 4 The Overall Approach

**Algorithm 1:** Vertex Cover overall approach

**Input:** A graph $G := (V, E)$
**Output:** Size of optimal VC, and optimal VC.

```
VC ← {}                                        ▷ initializing final vertex cover
Apply parameter independent reduction rules (1, 2, 4, 5)
for each component C in G do
    tempVC ← {}                                ▷ temporary vertex cover variable
    if C is a clique then
        Add all but one vertices of C to tempVC
    else
        tempVC ← minVertexCover(C)             ▷ invoke procedure minVertexCover
    end
    VC ← VC ∪ tempVC                           ▷ add tempVC to final vertex cover
end
return VC
```

**Algorithm 2:** procedure `minVertexCover`

**Input:** A connected component $C$ of $G$
**Output:** Optimal vertex cover

*Apply parameter independent reduction rules (1, 2, 4, 5)*
**if** *$C$ does not contain any edges* **then**
  **return** $\{\}$
**end**
▷ `find upper and lower bounds on vc`
`lpOpt` ← `halfIntLPVC(`$C$`)` ▷ `find half integral LP solution`
`low` ← `lpOpt` ▷ `set optimal LP solution as the lower bound`
`high` ← `2*lpOpt` ▷ `set preliminary upper bound`
▷ `compute greedy vertex cover by successively adding high-degree vertices to the cover`
`greedyVC` ← `greedyVertexCover(`$C$`)`
`high` ← `min(high, |greedyVC|)` ▷ `refine upper bound`
*Binary search for size of optimal cover in* `[low, high]`*, applying the standard vertex branching at each probe*

## 5 Employed and Proposed Heuristics

**Heuristics Employed:**

- Obtaining upperbound on VC using greedy VC
- LP solution (rounding) – Once the dual graph is created, we try several rounding strategies based on the outcome of maximum matching algorithm in bipartite graphs to force the solution as far from trivial all half solution.
- Handling certain degree 3 vertices

**Proposed heuristics during implementation:**

- Treewidth based approach for VC.
- Running two different solutions using time division schemes.
- Pruning branching at several places at once and continue with the branch which gives maximum information.

## 6 Implementation details

The implementation is made publicly available as a Git repository hosted on GitHub[Link]. This project has been developed and tested using `NetworkX 1.11`, `NumPy 1.11.3` and `Python 3.5.2`.

## Bibliography

1 Eric Angel, Romain Campigotto, and Christian Laforest. Implementation and comparison of heuristics for the vertex cover problem on huge graphs. In *International Symposium on Experimental Algorithms*, pages 39–50, 2012.

2 Jianer Chen, Iyad A Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, pages 3736–3756, 2010.

3 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015.
4 Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008.
5 Daniel Lokshtanov, NS Narayanaswamy, Venkatesh Raman, MS Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Transactions on Algorithms (TALG)*, 11(2):15, 2014.
6 The Sage Developers. *SageMath, the Sage Mathematics Software System*, 2019. https://www.sagemath.org.
7 P. R. Vaidyanathan and Chinmay Sonar. Python VC, June 2019. doi:10.5281/zenodo.3236977.