Randomized Minimum Vertex Cover On Undirected Graphs

Vasco Regal

Abstract **–Following previous analysis of minimum vertex optimization algorithms, this paper showcases yet another implementation, with a randomized approach. This means the algorithm will, with a certain degree of randomness, compute an approximate solution to the problem. To better understand the performance of the procedure, its results are put against the previously developed solutions.**

Resumo **–Em seguimento das análises previamente efetuadas em algoritmos de cobertura mínima de vértices, neste artigo é apresentada uma outra implementação, recorrendo a elementos aleatórios. Por outras palavras, este algoritmo vai, de acordo com um certo nível de aleatoriedade, calcular uma solução aproximada. Para melhor compreender a sua performance, os resultados são comparados com as soluções já desenvolvidas.**

Keywords **–graphs, optimization, min-vertex-cover, computational-analysis, randomized**

Palavras chave **–grafos, otimização, cobertura-minima, analise-computacional, randomizado**

I. The Minimum Vertex Cover

A. Definition

Given an undirected graph, *G = (V, E)* and *V'* a subset of *V*, *V'* is a vertex cover if:

$$\forall\ u,v \in E \implies u \in V' \vee v \in V'$$

In other words, a subset of the graph's vertexes in which all edges of the graph are incident in at least one of the elements of the subset. The vertex cover of minimum size is called a **minimum vertex cover**.

II. Previous Implementations

Before jumping into the focus of this paper, the aforementioned implementations are quickly summarized in this section.

A. Exhaustive

The exhaustive algorithm calculates all possible combinations of vertexes. Its an NP-complete algorithm.

B. Greedy

As for the greedy approach, a score is calculated for each vertex to form a list of best candidates as they are added to the final solution. The time complexity for this is $O(|E| * |V|)$

III. Randomized Algorithm

The randomness of this algorithm is based upon the selection of which vertex to add next to the solution. The procedure has a defined number of iterations, **MAX_ITER**. In each iteration, a random minimum vertex cover is generated, according to 2, starting with size=1 up to —V—. The size threshold is increased every time the program fails to compute a solution with said length and —V— iterations were conducted with the current size. Figure 1 presents the pseudo code for this.

Algorithm 1 Randomized Minimum Vertex Cover Approximation

```
1:  procedure MinimumVertexCover(V, E, MAX_ITERS
2:      passed_solutions ← []
3:      best ← NIL
4:      size ← 1
5:      while iter < MAX_ITER do
6:          solution ← RandomMVC(size)
7:          if solution == NIL then
8:              if size_iterations == |V| then
9:                  size+ = 1
10:             end if
11:         else
12:             AddToPassedSolutions(solution)
13:             if |solution| < |best| then
14:                 best ← solution
15:             end if
16:         end if
17:     end while
18:     return best
19: end procedure
```

IV. Implementation

As for implementation, a new class, **RandomizedSolver**, also implementing the Solver abstract class from previous development, was created. For this class, the solve method accepts an extra parameter, the number of iterations. The Problem and Solution classes had no major changes.

A. Analysis

A.1 Running experiments

The same CLI was used, with a little adaptation. Now the **-s** flag accept either (**E**)xhaustive, (**G**)reedy and (**R**)andom (first letter only) to specify the algorithm to use. Also, the **-i** flag specifies the MAX_ITER for the rand solver.

Algorithm 2 Generate Random Minimum Vertex Cover

```
procedure RandMVC(V, size)
    starting_vertex ← Random(V)
    solution ← [starting_vertex]
    size ← 1
    while |solution| < size do
        next ← RandomNeighborVertex(soltuion)
        if next == NIL then
            if IsVertexCover(solution) then
                return solution
            else
                return NIL
            end if
        end if
        AddToSolution(next)
    end while
    return solution
end procedure
```

Having this interface setup, with a simple shell script, 20 runs were made with the randomized subclass, starting at 5 vertexes. All runs were exported to files.

A.2 Analysing runs

With the results stored in files, the analysis was conducted with the aid of **matplotlib** to generate plots, **pandas** module and other data science related python packages. The results of said study is presented below.

V. RESULTS

For the experimentation, the runner script was used with following parameters:

- **Starting Vertexes** = 5
- **Max Vertexes** = 20
- **Edges** = 50% of max edges.
- **Seed** = 97636

A. Runs

The generated files from the random computations contain the data on table I (exhaustive).

B. Time Complexity

The time complexity was evaluated against the other solutions, plotting the execution time as V grows on 1. Evaluating the curves, this algorithm proves to have slightly better times than the greedy approach for V ¿ 15. It is however, and as expected, an exponential curve.
This data was fit to an expression, present on 2

C. Approximation error

With all the data collected, the approximation error was calculated, in relation to the values of the exhaustive runs, since, as a NP-complete problem, holds the best possible solution. Based on table II, the average error of each solution is around **16.23** %.

V	E	time(s)	solution
5	5	0.02294	2
6	7	0.06657	4
7	10	0.11726	5
8	14	0.14067	5
9	18	0.23153	6
10	22	0.48149	7
11	27	0.68489	8
12	33	0.99199	10
13	39	1.28048	10
14	45	1.30272	11
15	52	2.27233	12
16	60	2.90738	13
17	68	3.13390	13
18	76	3.46962	15
19	85	4.67807	16
20	95	6.08068	16

TABLE I
RANDOM SEARCH RESULTS

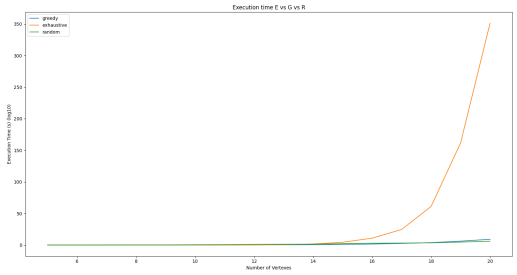

Fig. 1
EXECUTION TIME COMPARISON

$$t(V) = 0.13354 * e^(0.19272) * V) + -0.40307$$

Fig. 2
RANDOM ALGORITHM EXPONENTIAL FIT

This implementation proved, on the average case, to be slightly more accurate than the greedy one.

VI. CONCLUSION

The random approach proved to have not just better accuracy than the greedy implementation but also better times, in the average case.

REFERENCES

[1] Yongfei Zhang et al., An Efficient Heuristic Algorithm for Solving Connected Vertex Cover Problem, Haipeng Peng, https://www.hindawi.com/journals/mpe/2018/3935804/

[2] Wikipedia, *NP-completeness*, publicly available online, https://en.wikipedia.org/wiki/NP-completeness.

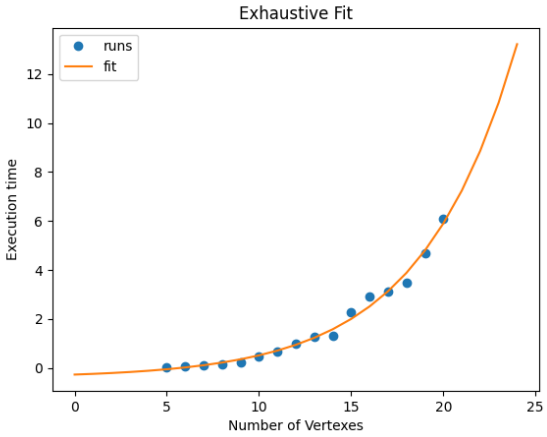

Fig. 3
Random fit vs actual data

V	exhaustive	random	absolute error
5	2	3	0.50
6	3	4	0.33
7	4	5	0.25
8	5	5	0.00
9	5	6	0.20
10	6	7	0.17
11	7	8	0.14
12	8	10	0.25
13	8	10	0.25
14	10	11	0.10
15	10	12	0.20
16	11	13	0.18
17	12	13	0.08
18	13	15	0.15
19	14	16	0.14
20	14	16	0.14

TABLE II
Approximation error