NOVA SCHOOL OF SCIENCE & TECHNOLOGY

Design of Algorithms for Optimization Problems

2020/2021

Report Phase 1:

Minimum Dominating Set

Teacher:

Margarida Mamede

Authors:

Alexandru Caramida nº 45604

Gonçalo Mateus nº 51927

Min. Dominating Set

Given an undirected graph G=(V, E) we want to compute a minimum size dominating set, a subset $S \subseteq V$ of its vertices such that for all vertices $v \in V$, either $v \in S$ or a neighbor u of v is in S.
Computing a dominating set of minimal size is NP-hard.

Greedy Algorithm

Initially, the dominant set D is empty and with each iteration of the algorithm, vertex $v \in V$ is added to D until D becomes a dominant set. The vertex selected to belong to D is chosen on the condition that it covers the maximum number of vertices not covered in the previous iteration. In case of tie the vertex to be added is chosen at random among them.

Algorithm: greedy(G)
Input: an undirected graph G
Output: size of the dominating set D

D = ∅
for every $v_i \in G$
    $weight_i$ = 1 + d(v_i)
    $covered_i$ = false

do
    v = chooseVertex (weight)
    if v != −1
        add v to D
        adjustWeights (G, weight, covered, v)
until v = −1

return D.size

Method: chooseVertex (weight)
Input: the weight vector
Output: a vertex of G which covers the maximum number of vertices not yet covered

M = max $weight_{i, 1 \le i \le n}$
if M = 0
    return −1
else
    S = $\{v_i | weight_i = M\}$
    randomly return an element of S

Method: adjustWeights(G, weight, covered, v_i)
Input: the graph G, the weight vector, the covered vector, the index of the v_i

$weight_i$ = 0
for every v_j neighbour of v_i that $weight_j > 0$
    if !$covered_i$
        $weight_j$- -
    if !$covered_j$
        $covered_j$= true
        $weight_j$- -
        for every v_k neighbor of vj
            if $weight_k > 0$
                $weight_k$ - -
$covered_i$= true

For a graph G with v vertices and e edges, a call to choose a vertex is at most ***O***(v). The total time spent in adjustWeights is ***O***(e). If d is the size of the dominant set found, the total complexity of the algorithm is in ***O***$(v*d + e) \le$ ***O***(v^2).

Time complexity: ***O***(v^2) (1)
Space complexity: ***O***(v^2)
Approximation ratio: ***O***(log(Δ)), where Δ is the maximum degree of a vertex (2)

Linear Programming Algorithm

The linear programming algorithm consists of solving the following linear optimization problem:

$$Min\ F = \sum xi$$

$xi + \sum variables\ of\ adjacent\ vertices >= 1$for all xi

xi >= 0 for all xi

We then count the number of variables with a value that exceeds 1/ (d+1), where d stands for the biggest degree a vertice has in the graph. The result is the size of an approximation of the minimum set cover.
The approximation ratio is d+1.

Test Instances

Social Network Samples
This is a set of anonymised social network samples from
https://davidchalupa.github.io/research/data/social.html
pokec_500.col
pokec_2000.col
pokec_10000.col
pokec_20000.col
pokec_50000.col
gplus_500.col
gplus_2000.col
gplus_10000.col
gplus_20000.col
gplus_50000.col

This is a set of Graph Coloring Instances from
https://mat.gsia.cmu.edu/COLOR/instances.html
anna.col
homer.col
david.col
huck.col

Both test instances can also be found in reference (2) page. 18-19

Results

Results sampled from 5 runs per test.
All times are in seconds.

Run time

Graph	Greedy		Linear Programming	
Samples Pokec	avg	standard deviation	avg	standard deviation
pokec 500	0.00596	0.00900	0.02885	0.03216
pokec 2000	0.00554	0.00220	0.07357	0.04831
pokec 10000	0.05289	0.00259	44.55567	0.48140
pokec 20000	0.20214	0.00127	623.65685	4.09649
pokec 50000	1.60451	0.01724	-	-
Samples Google+				
gplus 500	0.00033	0	0.01080	0.00136
gplus 2000	0.00415	0.00004	0.04592	0.00348
gplus 10000	0.09373	0.00031	1.76016	0.13252
gplus 20000	0.37071	0.00056	146.37404	1.03912
gplus 50000	2.75927	0.00752	-	-
DIMACS Graphs				
anna	0.00007	0	0.00241	0.00032
homer	0.00108	0.00021	0.01243	0.00074
david	0.00008	0.00005	0.00153	0.00011
huck	0.00005	0	0.00161	0.00017

Dominating Set

Graph	Greedy			Linear Programming
Samples Pokec	min	max	avg	
pokec 500, γ=16	16	16	16	16
pokec 2000, γ=75	75	75	75	75
pokec 10000, γ=413	413	414	413.60	413
pokec 20000, γ=921	925	928	926.20	921
pokec 50000, γ≥2706	2768	2781	2774	-
Samples Google+				
gplus 500, γ=42	42	42	42	42
gplus 2000, γ=170	175	179	176.80	172
gplus 10000, γ=861	894	902	896.40	870
gplus 20000, γ≥1716	1806	1813	1809.40	1718
gplus 50000, γ≥4566	4835	4847	4840.40	-
DIMACS Graphs				
anna, γ=12	12	12	12	12
homer, γ=96	96	96	96	96
david, γ=2	2	2	2	2
huck, γ=9	9	9	9	9

Analysis

We tried to use various types of tests, both big and small, for a better comparison between the algorithms. Looking at both algorithms in terms of compute time it's clear that the Greedy is by far the fastest one, we can also observe that the Linear Programming algorithm compute time increase, with more complex graphs, is superior to the Greedy algorithm.
In terms of Dominating Set results the two algorithms are close until more complex graphs are used, at a certain graph size the Linear Programming algorithm returns better results. Examples of this observation are the graphs with 10000 vertices or more and gplus 2000. The reason we can have different results in each iteration for the Greedy algorithm is because the cases of a tie between the vertices with more cover of others, and in this case the vertex is randomly selected between the tied ones.
We weren't able to produce results with the Linear Programming algorithm for the graphs with 50000 vertices because the compute times were too big and we decided to stop the computation.
The Greedy algorithm is much faster but for bigger graphs has worse results, on the other hand the Linear Approximation algorithm is much slower but gives results very close to the optimum. So if we want the smallest Dominating set and time is not a concern the Linear Programming algorithm is better but if time is a priority then the Greedy algorithm is recommended.
In conclusion, the one to use depends on the goal and complexity of the problem at hand.

References

1. Algorithm and time complexity
   Bouali Zakariae - Comparaison d'algorithmes pour le problème d'ensemble dominant minimum dans un graphe, thesis.pdf *page. 58-62*
   https://github.com/JavaZakariae/MinDominatingSet
2. Approximation ratio
   David Chalupa - An Order-based Algorithm for MinimumDominating Set with Application in GraphMining, *page. 4-5*.
   https://arxiv.org/pdf/1705.00318.pdf