An Efficient Algorithm for the k-Dominating Set Problem on Very Large-Scale Networks

Minh Hai Nguyen[1,2], Minh Hoàng Hà[1], Dinh Thai Hoang[2], Diep N. Nguyen[2], Eryk Dutkiewicz[2], and The Trung Tran[3]

[1] ORLab, VNU University of Engineering and Technology, Hanoi, Vietnam
[2] Faculty of Engineering and Information Technology, University of Technology Sydney, Australia
[3] FPT Technology Research Institute, Hanoi, Vietnam

The minimum dominating set problem (MDSP) aims to construct the minimum-size subset $D \subset V$ of a graph $G = (V, E)$ such that every vertex has at least one neighbor in D. The problem is proved to be NP-hard [4]. In a recent industrial application, we encountered a more general variant of MDSP that extends the neighborhood relationship as follows: a vertex is a k-neighbor of another if there exists a linking path through no more than k edges between them. This problem is called the minimum k-dominating set problem (MkDSP) and the dominating set is denoted as D_k. The MkDSP can be used to model applications in social networks [1] and design of wireless sensor networks [2]. In our case, a telecommunication company uses the problem model to supervise a large social network up to 17 millions nodes via a dominating subset in which k is set to 3.

Unlike MDSP that has been well investigated, the only work that addressed the large-scale MkDSP was published by [1]. In this work, the MkDSP is converted to the classical MDSP by connecting all non-adjacent pairs of vertices whose distance is no more than k edges. The converted MDSP is then solved by a greedy algorithm that works as follows. First, vertex v is added into the set D_k, where v is the most covering vertex. Then, all vertices in the set of k-neighbors of v denoted by $\mathcal{N}(k, v)$ are marked as covered. The same procedure is then repeated until all the vertices are covered. The algorithm, called *Campan*, could solve instances of up to 36,000 vertices and 200,000 edges [1]. However, it fails to provide any solution on larger instances because computing and storing k-neighbor sets of all vertices are very expensive.

The telecommunication company currently uses a simple greedy algorithm whose basic idea is to sort the vertices in decreasing order of degree. We then check each vertex in the obtained list. If the considering vertex v is uncovered, it is added to D_k and the vertices in $\mathcal{N}(k, v)$ become covered. Our experiments show that this algorithm, called HEU_1, is faster but provides solutions that are often worse than *Campan*.

Our main contribution is to propose an algorithm that yields better solutions at the expense of reasonably longer computational time than *Campan*. More specially, unlike *Campan*, our algorithm can handle very large real-world networks. The algorithm, denoted as HEU_2, includes three phases: preprocessing, solution construction, and post-optimization. In the first phase, we remove the connected components whose radius is less than $k+1$. The construction phase is similar to HEU_1 except that if the considering vertex v is covered but itself covers more than θ uncovered vertices, then v is added to D_k. We repeat this process with different integer values of θ from 0 to 4, and select the best result. In the post-optimization phase, we reduce the size of D_k by two techniques.

First, the vertices in D_k are divided into disjoint subsets; each contains about 20,000 vertices with the degree less than 1000, e.g. if there are 45,000 vertices in D_k that have degree less than 1000 then they are divided in to three subsets which have 20,000, 20,000 and 5000 vertices respectively. For each set B, we define set $\bar{B} = \cup_{v \in B} N(v, 1)$ and X, the set of all vertices covered by B, but not by $D_k \setminus B$. A Mixed Integer Programming (MIP) model is used to find a better solution B' which replaces B in D_k. The second technique is removing redundant vertices. A vertex $v \in D_k$ is redundant if there exists a subset $U \subset D_k \setminus \{v\}$, such that $\mathcal{N}(k, v) \subset \cup_{u \in U} \mathcal{N}(k, u)$.

Experiments are performed on a computer with Intel Core i7-8750h 2.2 Ghz and 24 GB RAM. Three algorithms are implemented in *Python* using *IBM CPLEX* 12.8.0 whenever we need to solve the MIP formulations. The summarized results are shown in Table 1. The first ten instances are from the Network Data Repository source [3]. The last two instances are taken from the data of the telecommunication company mentioned above. The values of k are set to 1 and 3. The results clearly demonstrate the performance of our proposed algorithm. It outperforms the current algorithm used by the company (HEU_1) in terms of solution quality and provides better solutions than $Campan$ on 10 over 12 instances. More specially, it can handle 13 very large instances that $Campan$ cannot (results marked "-" in $Campan$ columns).

			$k = 1$						$k = 3$					
			HEU_1		$Campan$		HEU_2		HEU_1		$Campan$		HEU_2	
Instances	$\|V\|$	$\|E\|$	**Sol**	**Time (s)**	**Sol**	**Time (s)**	**Sol**	**Time (s)**	**Sol**	**Time (s)**	**Sol**	**Time (s)**	**Sol**	**Time (s)**
ca-GrQc	4k	13k	1210	0.00	803	0.15	**776**	1.38	251	0.01	120	0.35	**102**	2.71
ca-HepPh	11k	118k	2961	0.01	1730	1.54	**1662**	6.49	430	0.02	138	14.63	**117**	53.76
ca-AstroPh	18k	197k	3911	0.02	2175	1.79	**2055**	15.22	438	0.06	122	75.60	**106**	203.18
ca-CondMat	21k	91k	5053	0.04	3104	4.20	**2990**	21.35	898	0.02	302	5.82	**266**	63.16
email-enron-large	34k	181k	12283	0.10	2005	4.48	**1972**	37.71	724	0.14	-	-	**92**	203.72
soc-BlogCatalog	89k	2093k	49433	0.72	**4896**	26.89	4915	1839.26	87	0.06	-	-	**15**	1616.70
soc-delicious	536k	1366k	215261	19.07	**56066**	1464.84	56600	5679.63	14806	2.44	-	-	**1505**	1695.77
soc-flixster	2523k	7919k	1452450	999	-	-	**91543**	27374.44	20996	29.71	-	-	**313**	3333.45
hugebubbles	2680k	2161k	1213638	2087.83	-	-	**1169394**	7498.20	843077	649.47	-	-	**688817**	17221.76
soc-livejournal	4033k	27933k	1538044	2689.72	-	-	**930632**	75185.96	211894	394.98	-	-	**83710**	42600.51
soc-tc-0	17642k	33397k	6263241	64228.04	-	-	**29278**	26740.42	6337	55.57	-	-	**5158**	5200.1448
soc-tc-1	16819k	26086k	4129393	19109.00	-	-	**38303**	38644.65	12807	78.3	-	-	**10905**	5481.59

Table 1: Comparisons among three algorithms.

References

1. Campan, A., Truta, T.M., Beckerich, M.: Approximation algorithms for d-hop dominating set problem. In: 12th International Conference on Data Mining. pp. 86–91 (2016)
2. Rieck, M., Pai, S., Dhar, S.: Distributed routing algorithms for wireless ad hoc networks using d-hop connected dominating sets. Computer Networks **47**, 785–799 (04 2005)
3. Rossi, R., Ahmed, N.: The network data repository with interactive graph analytics and visualization (02 2015)
4. Wang, Y., Cai, S., Chen, J., Yin, M.: A fast local search algorithm for minimum weight dominating set problem on massive graphs. In: Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI). pp. 1514–1522 (07 2018)