Nguyen *et al. Comput Soc Netw* (2020) 7:4
https://doi.org/10.1186/s40649-020-00078-5

Computational Social Networks

RESEARCH Open Access

Solving the *k*-dominating set problem on very large-scale networks

Minh Hai Nguyen[1,3], Minh Hoàng Hà[1,2*], Diep N. Nguyen[3] and The Trung Tran[4]

*Correspondence:
hoang.haminh@phenikaa-uni.edu.vn
[2] ORLab, Faculty of Computer Science, Phenikaa University, Hanoi, Vietnam
Full list of author information is available at the end of the article

Abstract

The well-known minimum dominating set problem (MDSP) aims to construct the minimum-size subset of vertices in a graph such that every other vertex has at least one neighbor in the subset. In this article, we study a general version of the problem that extends the neighborhood relationship: two vertices are called neighbors of each other if there exists a path through no more than *k* edges between them. The problem called "minimum *k*-dominating set problem" (M*k*DSP) becomes the classical dominating set problem if *k* is 1 and has important applications in monitoring large-scale social networks. We propose an efficient heuristic algorithm that can handle real-world instances with up to 17 million vertices and 33 million edges. This is the first time such large graphs are solved for the minimum *k*-dominating set problem.

Keywords: *k*-dominating set problem, Social networks, Large-scale networks, Heuristic method

Introduction

Problem context and definition

The well-known minimum dominating set problem (MDSP) deals with determining the smallest dominating set of a given graph $G = (V, E)$. The dominating set is a subset of the vertex set V such that each vertex in V is a member of the dominating set or is adjacent to a member of the dominating set. The applications of MDSPs are quite rich. The problems can be used in the study of social networks [1–3], design of wireless sensor networks [4], protein interaction networks [5, 6] and covering codes [7].

In a recent industrial application, the authors have been confronted with a more general variant of the MDSP which received, until now, only limited attention in the academic literature. We take the viewpoint of a company that runs a very large social network in which users can be modeled as nodes and the relationship among users can be modeled as edges. One of the important tasks of the company is monitoring all the activities (conversations, interactions, etc.) of the network users to detect anomalies such as cheating or spreading fake news. With millions of users, it is impossible to observe all users in the network. A potential solution is to construct a subset of users that can represent key properties of the network. The typical dominating set could be a good candidate. But in the case of social network scale, it is still too expensive to construct a dominating set because the size of

the dominating set could be large. Therefore, we need to consider the general version of dominating set named k-dominating set D_k which is defined as following: each vertex either belongs to the D_k or is connected to at least one member of D_k through a path of no more than k edges. The classical minimum dominating set corresponds to a special case when $k = 1$. For value $k > 1$, the cardinality of k-dominating set is less than that of 1-dominating set: $|D_k| \leq |D_1|$, the monitoring cost of the network is therefore reduced.

It should be noted that the value of k should be selected carefully. If k is too large, the users in the resulting dominating set cannot be the representatives for the original graph. But if k is too small, the monitoring cost would be very high due to the large size of the dominating set. In our application, k is in general set to 3. Figure 1 illustrates the solutions of the MkDSP (the k-dominating sets including the black nodes) in the cases of $k = 1$ and $k = 3$.

In this paper, we aim to construct the minimum k dominating set of a graph. The problem is called the minimum k-dominating set problem (abbreviated as MkDSP for short). Its application in determining a good approximation of large-scale social networks can be also found in [8]. The variant which requires vertices in k-dominating set to be connected can be used to form the backbone of an ad hoc wireless network as mentioned in [9, 10].

The MDSP is proved NP-complete [8], thus the MkDSP is clearly NP-hard because it reduces to the classical MDSP when $k = 1$. For further reading, we present a number of notations in the follows. If u is a vertex in the k-dominating set, and v is connected to u through a path with no more than k edges, we say u k-dominates (or covers) v or v is k-dominated (or covered) by u. In context without ambiguity we could remove the prefix k for short. We call a vertex of dominating set as a k-dominating vertex or dominating vertex for short. A vertex is a k-covered or k-dominated vertex if it is covered by a dominating vertex. The problem can be modeled as the following mixed-integer linear programming (MILP) model,

$$\text{Minimize} \sum_{v \in V} z_v, \tag{1}$$

$$\text{subject to} \sum_{v \in \mathcal{N}(u,k)} z_v \geq 1, \forall u \in V, \tag{2}$$

$$z_v \in \{0, 1\}, \forall v \in V. \tag{3}$$

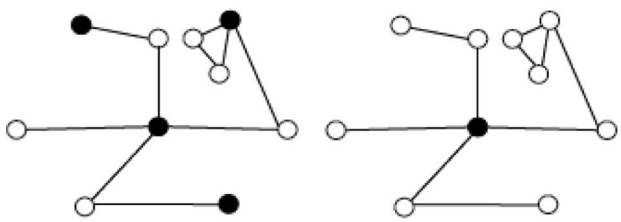

Fig. 1 The dominating set (black vertices) for $k = 1$ (left) and $k = 3$ (right)

where z_v is the binary variable representing whether the vertex v belongs to the k-dominating set, i.e., $z_v = 1$ if and only if $v \in D_k$. The objective (1) is to minimize the number of vertices in D_k while constraints (2) assure that each vertex u must be covered by at least one dominating vertex. Here, $\mathcal{N}(u,k)$ denotes the set of vertices that can cover u, i.e., the vertices connect to u through a path with no more than k edges. The cardinality of $\mathcal{N}(u,k)$ plays an important role in the investigation of complexities of the algorithms presented in the next sections. In general, we denote $n_k = |\mathcal{N}(u,k)|$; its value can be estimated on average by $n_k = (\overline{d}^{k+1} - 1)/(\overline{d} - 1) \approx \mathcal{O}(\overline{d}^k)$, where $\overline{d}$ is the average degree of vertices in the graph and is equal to $2|E|/|V|$. The optimal algorithm to compute $\mathcal{N}(u,k)$ is the breadth first search algorithm which has the complexity of $\mathcal{O}(n_k)$.

It can be seen that both the number of binary variables and the number of constraints in the MILP model above are equal to the size of vertex set V. This is a very large number of graphs arising in the context of social networks. Modeling and solving such a big formulation appears to be an impossible task for current MILP tools and computing capacity.

Literature review

Literature has attempted to deal with the MSDP. The most efficient exact method for the problem and other variants is presented in [11] where a branch-and-reduce method is developed. Although this method can provide an optimal solution, it handles only small-size instances defined on graphs with a few hundred vertices in acceptable running time. Several efforts are spent to design approximation algorithms. Grandoni [12] proposes an algorithm in $\mathcal{O}(1.9053^n)$ while Rooij and Bodlaender [11] propose algorithm in $\mathcal{O}(1.4969^n)$ time and polynomial space.

The MDSP can also be tackled by existing approaches proposed to solve its variants. The most popular variant of the problem deals with a weight associated with each vertex of the graph, called the minimum weight dominating set (MWDS) problem (Ugurlu et al. [13]). The objective function seeks to minimize the total weight, without regarding the cardinality of the dominating set. The best metaheuristic in terms of solution quality for the MWDS is recently introduced by [14]. It is a hybrid metaheuristic combining a tabu search with an integer programming solver. The MILP solver is used to solve subproblems in which only a part of the decision variables, selected relative to the search history, are left free. The authors also introduce an adaptive penalty to promote the exploration of infeasible solutions during the search, enhance the algorithm with perturbations and node elimination procedures, and exploit richer neighborhood classes. The performance of the method is investigated on small- and medium-size instances with up to 4000 vertices. For massive graphs, Wang et al. [15] develop a local search algorithm called FastMWDS. Two new ideas are used in FastMWDS. First, a new fast construction procedure with four reduction rules is proposed. This procedure includes three parts: reducing, constructing, and shrinking. After this construction procedure, the size of massive graphs is reduced. Second, a new configuration checking with multiple values is designed, which can exploit the relevant information of the current solution.

Relating to MkDSP problem, a number of variants of this problem have been proposed and studied. As most of the related problems studied in the literature are in the context of wireless networks, in works, the dominating set is usually required to be connected.

The problem can be solved in polynomial time on several restricted graphs such as distance-hereditary graphs [16], HT-graphs [17], and graphs with bounded treewidth [18]. The hardness and approximation algorithms are introduced in [19, 20]. Two approximation algorithms are also developed to solve the minimum 2-connected k-dominating set problem in [9]. The first one is a greedy algorithm using an ear decomposition of 2-connected graphs. The second one is a three-phase algorithm that can be used to handle disk graphs only. Rieck et al. [10] propose a distributed algorithm to provide approximate solutions. The algorithm is tested on a small graph with only several hundred vertices.

To the best of our knowledge, the only work that proposes efficient algorithms to solve the MkDSP in the context of a large social network has been recently published by Campan et al. [8]. The MkDSP is first converted to the classical minimum dominating set problem by adding edge connecting vertices that are not adjacent but have distance not exceeding k. The MkDSP can now be solved by directly applying one of three greedy algorithms that work for the MDSP. The performance of algorithms is tested on medium-size real social networks with up to 36,000 vertices and 200,000 edges. However, as shown in the experimental section, the method proposed in [8] cannot provide any solution for the instances with millions of vertices and edges in acceptable running time.

Problem challenges and contributions

One of the challenges to solve the MkDSP is to determine the domination relation between pairs of vertices. In general, this often leads to a procedure that we call k-neighbor search to compute the set $\mathcal{N}(u,k)$ for vertex u, which is very expensive on massive graphs with $k > 1$. As a consequence, approaches proposed in the literature that pre-compute the dominating set of every vertex are infeasible in the context of massive graphs. For example, the method proposed in [14] uses a decomposition method to tackle the MWDS and solves multiple sub-problems, each corresponds to an MILP and then uses several local search operators. To speed up the local search procedure, the set $\mathcal{N}(u,k)$ for every vertex u in the graph has to be pre-computed. Multiple MILP programs and the pre-computation of $\mathcal{N}(u,k)$ make the algorithm perform slowly in the case of very large-scale graphs. Similarly to the algorithm proposed in [15], even though it can handle large-scale instances in the context of social networks but it works only in the case where $k = 1$. Applying this algorithm to solve our problem with $k > 1$ is not practical because when k increases, the algorithm gets stuck as it has to iteratively compute the set $\mathcal{N}(u,k)$ for every vertex u.

The MkDSP can be converted to a typical dominating set problem by inserting additional edges to the graph G that joint two non-adjacent vertices if the number of edges on the path among them is not greater than k. This polynomial complexity conversion procedure allows using any efficient algorithm proposed for the 1-dominating set problem to solve k-dominating set problem. The idea is proposed in [15]. However, inserting edges increases significantly the degree of vertices in the graph, leading to tedious performance of the method in terms of computational speed when tackling large-scale social networks with millions of vertices as shown in the experiments in "Experimental results" section.

In this paper, we consider the MkDSP in the context of social networks. Our main contribution is an algorithm that can efficiently solve the MkDSP. The novel features of our method are (i) a prepossessing phase that reduces the graph's size; (ii) a construction phase with different greedy algorithms; and (iii) a post-optimization phase that removes redundant vertices. In all phases, we also use techniques to reduce the number of times to compute k-neighbor set of vertices which is very expensive on graphs arisen in social networks.

We have investigated the performance of our method on different sets of graphs which are classified mainly by their size of vertex set. A graph is labeled as a large size category if it has more than 100 thousand vertices, while the small one has less than 10 thousand vertices, the remaining cases are of medium size. The obtained results show the performance of our method. It outperforms the algorithm currently used by the company mentioned above in terms of solution quality. It can also handle real large-scale instances with up to 17 million vertices that the algorithm proposed in [8] could not. Finally, it is worth noting that an extended abstract of this paper is published in [21]. In the current work, we describe in more details the main sections including literature review, heuristic method, and experimental results. In particular, we add an additional section to show the hardness of the problem and carry out more experiments to analyze the performance of the methods.

Solution methods

In this section, we describe in detail an efficient algorithm for large-scale MkDSP problems. Our heuristic consists of three phases: pre-processing phase to reduce the graph size, construction phase to build a k-dominating set that will be reduced in the post-optimization phase by removing redundant vertices.

Pre-processing phase

As mentioned above, the first phase of our algorithms is reducing the size of the original graph. We extend the reduction rules in [15] to k-dominating set by finding structures that we call k-isolated clusters. A k-isolated cluster is a connected component whose vertices are k-dominated by a single vertex. If there exists a vertex $v \in V$ such that $|\mathcal{N}(v,k)| = |\mathcal{N}(v,k+1)|$, set $\mathcal{N}(v,k)$ is a k-isolated cluster associated with v. We can remove the vertices belonging to this k-isolated cluster from G and add vertex v to the k-dominating set. Algorithm 1 describes our reduction rule on small- and medium-size graphs. To estimate the complexity of Algorithm 1, it is easy to see that the FOR loop in Line 2 has $|V|$ steps and in each step, $k+1$ and k neighbors have been calculated. Therefore, the complexity of Algorithm 1 in the worst case is $\mathcal{O}(|V|n_{k+1})$.

Algorithm 1 Remove the isolated clusters on small and medium-size graphs

Input: Graph $G = (V, E)$ with $|V| < 100,000$
Output: Reduced graph G'

1: $\mathcal{R} \leftarrow \emptyset$
2: **for** $v \in V(G)$ **do**
3: **if** $|\mathcal{N}(v,k)| = |\mathcal{N}(v,k+1)|$ **then**
4: $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{N}(v,k)$
5: **end if**
6: **end for**
7: $G' \leftarrow$ reduced graph of G on $V(G) \setminus \mathcal{R}$
8: **return** G'

Algorithm 1 does not work on large-size graphs due to the expensive cost of k-neighbor search $\mathcal{N}(v,k)$. As a sequence, on massive graphs with more than 100,000 vertices, we implement a modified version of Algorithm 1 that is shown in Algorithm 2. The idea is based on the observation that, if $|\mathcal{N}(v,k)| \neq |\mathcal{N}(v,k+1)|$, it is highly possible that $\mathcal{N}(u,k)$ would not be an isolated cluster for every $u \in \mathcal{N}(v,k+1)$. We could thus ignore the isolated clusters checking on $\mathcal{N}(u,k)$. In Algorithm 2, for each vertex v, the variable $f[v]$ is set to *False* if $\mathcal{N}(v,k)$ has a high probability of not being an isolated cluster. If a vertex is marked *False*, it is not checked through the condition in Line 7, to avoid computing k-neighbor searches. The complexity of Algorithm 2 is $\mathcal{O}(|V|n_{k+1}/n_k)$. More precisely, the FOR loop in Line 5 repeats $|V|$ times and there are $|V|/n_k$ vertices that we need to compute their $(k+1)$-neighbor set, which runs in $\mathcal{O}(n_{k+1})$.

Algorithm 2 Remove the isolated clusters on large-scale graphs

Input: Graph $G = (V, E)$ with $|V| \geq 100,000$
Output: Reduced graph G'

1: $\mathcal{R} \leftarrow \emptyset$
2: **for** $v \in V(G)$ **do**
3: $f[v] \leftarrow True$
4: **end for**
5: **for** $v \in V(G)$ **do**
6: **if** $f[v] = True$ **then**
7: **if** $|\mathcal{N}(v,k)| = |\mathcal{N}(v,k+1)|$ **then**
8: $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{N}(v,k)$
9: **end if**
10: **for** $u \in \mathcal{N}(v,k)$ **do**
11: $f[u] \leftarrow False$
12: **end for**
13: **end if**
14: **end for**
15: $G' \leftarrow$ reduced graph of G on $V(G) \setminus \mathcal{R}$
16: **return** G'

k-dominating set construction phase

To begin this subsection, we introduce the greedy heuristic that is currently used by our partner mentioned in the first section. The idea is originated from the observation that the higher degree vertex would tend to dominate more vertices. Thus, the vertices in the graph are first rearranged in descending order of their degree and then consecutively consider each vertex in the received list. If the considering vertex v is uncovered, it is added to the k-dominating set D_k and all members of the k-neighbor

set $\mathcal{N}(v,k)$ is marked as covered. This greedy heuristic is denoted as HEU_1 and is shown in Algorithm 3.

The complexity of HEU_1 is $\mathcal{O}(|V|\log(|V|) + |D_k|n_k)$. First, sorting the vertices in Lines 2–3 costs $\mathcal{O}(|V|\log(|V|))$. In the FOR loop in Lines 6–13, there are $|D_k|$ times a vertex is added to D_k. And at each addition operation, we need to compute $\mathcal{N}(v,k)$, which runs in $\mathcal{O}(n_k)$ (FOR loop in Lines 9–10). Therefore, the complexity of FOR loop in Lines 6–13 is $\mathcal{O}(|V| + |D_k|n_k)$.

Algorithm 3 Greedy heuristic 1 (HEU_1)

Input: Graph $G = (V, E)$
Output: k-dominating set D_k
1: sort vertices in $V(G)$ in descending order of degree
2: **for** $v \in V(G)$ **do**
3: $isCovered[v] \leftarrow False$
4: **end for**
5: $D_k \leftarrow \emptyset$
6: **for** $v \in V(G)$ **do**
7: **if** $isCovered[v] = False$ **then**
8: $D_k \leftarrow D_k \cup \{v\}$
9: **for** $u \in \mathcal{N}(v,k)$ **do**
10: $isCovered[u] \leftarrow True$
11: **end for**
12: **end if**
13: **end for**
14: **return** D_k

The heuristic HEU_1 is fast and can handle very large-scale instances but such a simple greedy algorithm cannot provide high-quality solutions. To search for better solutions, we now present the second greedy algorithm called HEU_2 whose pseudo-code is provided in Algorithm 4. This algorithm is different from the first one in the way to treat covered vertices. In HEU_1, covered vertices are never added to the dominating set while in HEU_2, they can be still added if some conditions are satisfied. In Algorithm 4, $\mathcal{N}'(k,v)$ denotes the set of uncovered vertices in $\mathcal{N}(k,v)$. Line 10 in Algorithm 4 indicates that if the vertex v is uncovered or the number of uncovered vertices in $\mathcal{N}(k,v)$ is greater than a pre-defined parameter θ, vertex v will be selected as a dominating vertex. In practice, the operations from Line 6 to Line 16 of HEU_2 are quite time-consuming. While HEU_1 has to compute the k-neighbor sets for a number of vertices that is equal to the size of dominating set, the operations of Lines 6–16 in HEU_2 have to compute the k-neighbor sets for every vertex in the graph. To speed up the process, we limit the running time for the operations 6–16 by the conditions in Line 7 using the parameter t_{loop}. Here, t_{6-16} is the running time of the FOR loop 6–16. If t_{loop} is set to a large value, the running time of the algorithm could be very high due to the computation of k-neighborhood sets of all vertices on Line 10. However, another observation is that once the running time t_{6-16} exceeds t_{loop}, HEU_1 will be applied on the remaining unexplored vertices. That means if t_{loop} is set to a too small value, HEU_2 would behave almost like HEU_1, possibly leading to low-quality solutions. Therefore, the parameter t_{loop} should be neither too large nor too small. It should be neither less than $t_{\min}$ seconds nor greater than $t_{\max}$ seconds, and is computed as $t_{\text{loop}} = \max\left(t_{\min}, t_{\max}.{}^{|V|}/_{N}\right)$ (seconds) where N is approximately the number of vertices in the largest instances. We select the values of $t_{\min}, t_{\max}$, and N mainly by experiments. In experiments, we set

$t_{\min} = 400$, $t_{\max} = 950$, and $N = 17,000,000$. If the running time of FOR loop at Line 6 excesses t_{loop} and there are still uncovered vertices (Line 17), HEU_2 applies the same strategy as in HEU_1 for uncovered vertices (Lines 17–18).

The complexity of Algorithm HEU_2 is $\mathcal{O}(|V|\log(|V|) + |V|n_k)$. The sorting operation in Line 1 runs in $\mathcal{O}(|V|\log(|V|))$. The FOR loop in Lines 6–16 runs $|V|$ times. Each time, if the considering vertex v is covered its k-neighbor set will be computed; otherwise, the uncovered subset $\mathcal{N}'(v,k)$ of $\mathcal{N}(v,k)$ will be computed. The computations of $\mathcal{N}(v,k)$ and $\mathcal{N}'(v,k)$ have the same complexity as $\mathcal{O}(n_k)$. Therefore, the main operation is to construct a k-neighbor set with the complexity of $\mathcal{O}(n_k)$ on average.

Algorithm 4 Greedy algorithm 2 (HEU_2)

Input: Graph $G = (V, E)$, parameter θ
Output: k-dominating set D_k^θ

1: sort vertices in $V(G)$ in descending order of degree
2: **for** $v \in V(G)$ **do**
3: $isCovered[v] \leftarrow False$
4: **end for**
5: $D_k^\theta \leftarrow \emptyset$
6: **for** $v \in V(G)$ **do**
7: **if** (running time t_{6-16} is reached to t_{loop} or every vertex is covered) **then**
8: break
9: **end if**
10: **if** $isCovered[v] = False$ or $|\mathcal{N}'(k,v)| \geq \theta$ **then**
11: $D_k^\theta \leftarrow D_k^\theta \cup \{v\}$
12: **for** $u \in \mathcal{N}(k,v)$ **do**
13: $isCovered[u] \leftarrow True$
14: **end for**
15: **end if**
16: **end for**
17: **if** $\exists\, v \in V(G)$ such that $isCovered[v] = False$ **then**
18: apply algorithm HEU_1 for the uncovered vertices
19: **end if**
20: **return** D_k^θ

Experiments show that the performance of the algorithm HEU_2 heavily depends on the value of θ. An interesting fact is that HEU_2 behaves similarly as HEU_1 if θ and t_{loop} are set to very large numbers. If the value of θ is large enough, HEU_2 provides the same solutions as HEU_1, but it is more time-consuming (due to the computation of $\mathcal{N}'(v,k)$ in Line 10). Therefore, to get better solutions, we decide to execute HEU_2 with several small integer values of θ from 0 to 4 and choose the best one.

Post-optimization phase

The k-dominating set D_k obtained from algorithm HEU_2 can contain redundant vertices that can be removed while the remaining vertices still k-dominate the graph. We implement a procedure named *greedy redundant removal* to remove such redundant vertices. The algorithm is shown in Algorithm 5.

Algorithm 5 Greedy redundant removal

Input: Graph $G = (V, E)$, a k-dominating set D_k
Output: a reduced k-dominating set D_k

1: sort vertices in D_k in ascending order of size of their k-neighbour set
2: $k_1 = \lfloor 1/2(k+1) \rfloor$
3: $k_2 = k - k_1$
4: **for** $v \in D_k$ **do**
5: $\mathcal{S} \leftarrow TRUE$
6: **for** u in $\mathcal{N}(v, k)$ **do**
7: $\mathcal{T} \leftarrow FALSE$
8: compute $\mathcal{N}(u, k_1)$
9: **for** $w \in D_k \setminus \{v\}$ **do**
10: **if** $\mathcal{N}(w, k_2) \cap \mathcal{N}(u, k_1) \neq \emptyset$ **then**
11: $\mathcal{T} \leftarrow TRUE$
12: break
13: **end if**
14: **end for**
15: **if** $\mathcal{T} = FALSE$ **then**
16: $\mathcal{S} \leftarrow FALSE$
17: break
18: **end if**
19: **end for**
20: **if** $\mathcal{S} = TRUE$ **then**
21: $D_k = D_k \setminus \{v\}$
22: **end if**
23: **end for**
24: **return** D_k

The FOR loop in Lines 4–23 in Algorithm 5 considers every dominating vertex $v \in D_k$ to check if it is redundant. The variable $\mathcal{S}$ gets *TRUE* value if v is redundant and *FALSE* otherwise. If v is not redundant, there exists a vertex u in $\mathcal{N}(v, k)$ such that u is not covered by any vertex w in $D_k \setminus \{v\}$. Instead of computing $\mathcal{N}(u, k)$ and checking whether $w \in \mathcal{N}(u, k)$, which are very expensive on large-scale instances, we verify if $\mathcal{N}(u, k_1)$ and $\mathcal{N}(w, k_2)$ are not disjoint. Here, k_1 and k_2 are positive integers such that $k_1 + k_2 = k$.

The sorting operation in Line 1 runs in $\mathcal{O}(|D_k| \log(|D_k|))$. The FOR loop in Lines 6–19 repeats for $|D_k|$ times. The FOR loop in Lines 9–14 operates n_k iterations in the worst case. Inside this loop, there is a k_1-neighbor set construction $\mathcal{N}(u, k_1)$ in Line 8. To verify the condition in Line 10, we sort the element of the small-size set and perform binary search of elements in the large-size set on the small-size set. The complexity of this operation is $\mathcal{O}(\max\{n_{k_1}, n_{k_2}\} \log(\min\{n_{k_1}, n_{k_2}\}))$, where n_{k_1} and n_{k_2} are cardinalities of sets $\mathcal{N}(u, k_1)$ and $\mathcal{N}(w, k_2)$, respectively, leading to the complexity $\mathcal{O}(|D_k|^2 n_k \max\{n_{k_1}, n_{k_2}\} \log(\min\{n_{k_1}, n_{k_2}\}))$ of the whole Algorithm 5. We also note that if we do not separate k into k_1 and k_2, the complexity of the algorithm becomes $\mathcal{O}(|D_k|^2 n_k^2)$.

It is observable that when the gap between k_1 and k_2 gets larger, the computational cost $\max\{n_{k_1}, n_{k_2}\} \log(\min\{n_{k_1}, n_{k_2}\}))$ gets higher. As a result, we set $\left\{k_1, k_2\right\} = \{\lfloor 1/2(k+1) \rfloor, k - \lfloor 1/2(k+1) \rfloor\}$ that guarantees $|k_1 - k_2| \leq 1$. Inside the FOR loop 6–19, a number of k_2 neighbor sets $\mathcal{N}(w, k_2)$ are computed while only one k_1 neighbor set $\mathcal{N}(u, k_1)$ must be evaluated. Therefore, it is better if $k_1 \geq k_2$; and we assign $k_1 = \lfloor 1/2(k+1) \rfloor$ and $k_2 = k - k_1$. For example, in case of $k = 3$, we set $k_1 = 2$ and $k_2 = 1$. The complexity of Algorithm 5 becomes

$\mathcal{O}(|D_k|^2 n_3 n_2 \log(n_1)) \approx \mathcal{O}(|D_k|^2 \overline{d}^5 \log(\overline{d}))$, which is better than $\mathcal{O}(|D_k|^2 n_3^2) \approx \mathcal{O}(|D_k|^2 \overline{d}^6)$, the complexity of the algorithm if we directly verify the condition $w \in \mathcal{N}(u,k)$. Here, we recall that $\overline{d}$ is the degree on average of vertices in the graph.

After finishing the greedy redundant vertex removal, we continue to perform the second post-optimization phase by solving MILP programs as follows. We divide the vertices in the obtained k-dominating set D_k of degree less than a given value d_p into several groups; each contains n_p vertices maximum. For such a group B, let X be the set of neighbors of the vertices in B, i.e., $X = \cup_{v \in B} \mathcal{N}(v,1)$. Let S be the set of vertices that are only dominated by vertices in B and not by ones in $D_k \setminus B$. We solve the following integer programming problem in a limited time of t_p. The number of groups is about $|D_k|/n_p$ and the running time to tackle each group is limited to t_p, the total running time in the worst case is therefore $t_p|D_k|/(n_p.n_t)$, where n_t is the number of threads used for this phase.

$$\text{Minimize} \sum_{v \in X} z_v, \tag{4}$$

$$\text{subject to} \sum_{v \in \mathcal{N}(u,k) \cap X} z_v \geq 1, \forall u \in S, \tag{5}$$

$$z_v \in \{0,1\}, \forall v \in X. \tag{6}$$

If the feasible solution B' has smaller size than B, we replace elements of B in D_k by B', i.e., $D_k = (D_k \setminus B) \cup B'$. The values of d_p, n_p, and t_p must be carefully selected so that the performance of the algorithm is assured while the running time is still kept reasonable. By experiments, we decide to use the setting $n_p = 15,000$ and $t_p = 6$ s. The algorithm is first run with the value of $d_p = 500$ and then is repeated with $d_p = 5000$ to search for further improvement.

Experimental results

This section presents the results of the proposed methods on graphs of various sizes. Experiments are conducted on a computer with Intel Core i7—8750h 2.2 GHz running *Ubuntu* OS. The programming language is *Python* using *igraph* package to perform graph computations. We use `CPLEX 12.8.0` to solve MILP programs. The pre-processing and set dominating construction phases take 1 thread while the MILP solver takes 4 threads.

We test the approaches on three instance classes categorized by the size of their graphs. Small instances are taken from [14] with the number of vertices varying from 50 to 1000. This dataset contains 540 instances. To avoid long result tables, we select to show results for only five groups, each contains 10 instances with the same vertex and edge numbers. Six medium-size instances are from the Network Data Repository source [22] which are also used by [8] to test their algorithm. The third instance class includes six large-size instances: two with approximately 17 million vertices and 30 million edges extracted from the data of our partner (soc-partner-1 and

Table 1 Instance characteristics and results of the pre-processing phase

Instances			NoC			NoR		
Name	$\|V\|$	$\|E\|$	$k=1$	$k=2$	$k=3$	$k=1$	$k=2$	$k=3$
s-1	50	50	0	0	0	0	0	0
s-2	100	250	0	0	0	0	0	0
s-3	300	1000	0	0	0	0	0	0
s-4	800	10k	0	0	1	0	0	800
s-5	1000	15k	0	0	1	0	0	1000
soc-BlogCatalog	89k	2093k	0	0	0	0	0	0
ca-GrQc	4k	13k	0	0	0	0	0	0
ca-AstroPh	18k	197k	0	0	0	0	0	0
ca-HepPh	11k	118k	0	0	0	0	0	0
email-enron-large	34k	181k	0	0	0	0	0	0
ca-CondMat	21k	91k	0	0	0	0	0	0
soc-delicious	536k	1366k	0	0	0	0	0	0
soc-flixster	2523k	7919k	0	0	0	0	0	0
hugebubbles	2680k	2161k	532k	585k	617k	1063k	1224k	1360k
soc-livejournal	4033k	27933k	0	0	0	0	0	0
soc-partner-1	17642k	33397k	4850	4852	4852	4853	4867	4867
soc-partner-2	16819k	26086k	9799	9896	9896	9866	16171	16171

soc-partner-2) and four from Network Data Repository source. Table 1 shows the characteristics of the instances containing name, vertex size (column $|V|$), and edge size (column $|E|$). It also reports the results of the pre-processing phase including the number of isolated clusters (NoC) and the number of vertices in isolated clusters (NoR) in three cases corresponding to three values of k: 1, 2 and 3.

As can be seen in Table 1, the number of isolated clusters and reduces vertices increases when the value of k is higher. On the small graphs, these numbers are all zero except two classes s-4 and s-5 in the case $k = 3$ where the pre-processing phase can reduce 800 and 1000 vertices, respectively. Remarkably, in these cases, all the vertices are reduced; hence, the algorithm gets the optimal solution right after the pre-processing phase. On the medium-size graphs, the pre-processing procedure cannot remove any vertex. But in half instances of large graphs, the number of isolated clusters and removed vertices is significant.

We compare the performance of four algorithms: the MILP formulation with running time limited to 400 s, the greedy algorithm currently used by our partner HEU_1, the best algorithm proposed by [8] called HEU_3, and our new algorithm called HEU_4 including all components mentioned in the last section. For each method, we report the objective value of its solutions (Sol) and the running time (T) in seconds. For the method using MILP formulation, we also show the gaps (Gap) returned by CPLEX. Because the MILP-based method cannot handle efficiently medium and large-size graphs, we only present its results obtained on small-size graphs. In result tables, the numbers in italic show the best found k-dominating sets over all methods and the marks "−" denote the instances that cannot be solved by HEU3 in the running time of several days or due to "out of memory" status.

Table 2 Result of algorithms on small graphs

Data	HEU_1		MILP			HEU_3		HEU_4	
	Sol	T (s)	Sol	T (s)	Gap	Sol	T (s)	Sol	T (s)
$k = 1$									
s-1	21.5	0.00	*17.0*	147.72	0.00	19.8	0.00	*17.0*	0.40
s-2	29.3	0.00	*19.9*	403.94	0.90	23.6	0.00	*19.9*	0.54
s-3	77.1	0.00	53.1	403.15	0.52	57.1	0.00	*49.4*	12.39
s-4	90.3	0.00	64.3	400.27	0.98	56.4	0.01	*53.0*	12.59
s-5	98.9	0.00	72.7	400.40	0.69	61.0	0.02	*58.6*	12.66
$k = 2$									
s-1	13.5	0.00	*10.0*	52.33	0.00	12.2	0.00	10.2	0.42
s-2	10.9	0.00	*5.8*	36.97	0.00	6.7	0.00	5.9	0.47
s-3	20.1	0.00	12.0	400.17	0.68	13.2	0.01	*11.7*	2.44
s-4	15.7	0.00	4.6	403.21	0.78	*4.3*	0.13	*4.3*	7.56
s-5	6.4	0.00	*4.0*	406.08	0.50	*4.0*	0.23	*4.0*	11.36
$k = 3$									
s-1	9.8	0.00	*7.4*	8.55	0.00	8.7	0.00	7.5	0.34
s-2	3.0	0.00	*2.0*	0.09	0.00	*2.0*	0.00	2.1	0.35
s-3	5.3	0.00	*2.9*	2.35	0.00	3.0	0.02	*2.9*	0.42
s-4	*1.0*	0.00	*1.0*	0.02	0.00	*1.0*	0.42	*1.0*	0.02
s-5	*1.0*	0.00	*1.0*	0.01	0.00	*1.0*	0.76	*1.0*	0.02

Table 2 shows the experimental results on the small graphs which are average values over 10 instances. The numbers in italic show the best found k-dominating sets overall methods. An interesting observation is that the MILP-based method can solve to optimality more instances when k increases. More precisely, it can solve all instances with $k = 3$. Therefore, for exact methods, instances with larger values of k tend to be easier. HEU_1 is the worst in terms of solution quality, but it is the fastest. Considering HEU_3 and HEU_4's solution quality, HEU_4 dominates HEU_3 in 10 cases while HEU_3 is better in only one case. HEU_4 also provides better solutions than MILP formulation in several instances that cannot be solved to optimality, i.e., when gap values are greater than zero.

Table 3 shows the experiments on the medium-size graphs. The algorithm HEU_4 performs better than HEU_1 and HEU_3 on all instances but one in terms of solution quality. And finally, Table 4 shows experiments for the large instances. As can be seen, although slower as expected, HEU_4 still provides significantly better solutions than HEU_1. The heuristic HEU_3 gets trouble on large-scale instances when it cannot give any solution in several days of computation for five over six instances. This shows the scalability of the new algorithm HEU_4 compared with HEU_3. An interesting observation is that when the value of k increases, the running time of the algorithms tends to decrease. An explanation for this phenomenon is that the increase of k leads to solutions with smaller cardinality of k-dominating sets. More precisely, if the cardinality of k-dominating set D_k is smaller, the FOR loop 6–16 of Algorithm 4 would tend to be finished faster because the IF condition on Line 7 would halt the FOR loop 6–16 if every vertex is covered. In the post-optimization phase, the cardinality of D_k also affects the running time of both steps. For the greedy redundant vertex removal, the number of operations of FOR loops 4–23 and 9–14 of Algorithm 5 is proportional to the cardinality of D_k. For the post-optimization

Table 3 Result of algorithms on medium graphs

Data	HEU_1		HEU_3		HEU_4	
	Sol	***T*** **(s)**	**Sol**	***T*** **(s)**	**Sol**	***T*** **(s)**
$k = 1$						
ca-GrQc	1210	0.00	803	0.15	*776*	1.38
ca-HepPh	2961	0.01	1730	1.54	*1662*	6.49
ca-AstroPh	3911	0.02	2175	1.79	*2055*	15.22
ca-CondMat	5053	0.04	3104	4.20	*2990*	21.35
email-enron-large	12283	0.10	2005	4.48	*1972*	37.71
soc-BlogCatalog	49433	0.72	*4896*	26.89	4915	1839.26
$k = 2$						
ca-GrQc	415	0.00	285	0.10	*260*	1.77
ca-HepPh	879	0.00	473	0.99	*410*	13.76
ca-AstroPh	1073	0.02	457	2.93	*381*	47.02
ca-CondMat	1617	0.02	922	1.65	*806*	17.32
email-enron-large	1256	0.03	360	7.91	*346*	31.45
soc-BlogCatalog	2870	0.18	_	_	*229*	883.09
$k = 3$						
ca-GrQc	251	0.01	120	0.35	*102*	2.71
ca-HepPh	430	0.02	138	14.63	*117*	53.76
ca-AstroPh	438	0.06	122	75.60	*106*	203.18
ca-CondMat	898	0.02	302	5.82	*266*	63.16
email-enron-large	724	0.14	_	_	*92*	203.72
soc-BlogCatalog	87	0.06	_	_	*15*	1616.70

using MILP, the number of programs to solve and their size also depend on the cardinality of D_k. However, this phenomenon is not observed in HEU_4 on several instances because of the execution of the post-optimization phase with CPLEX, whose running time could depend on not only the size of the dominating sets but also other unknown characteristics of input data.

Conclusion

In this paper, we study the *k*-dominating problem in the context of very large-scale input data. The problem has important applications in social network monitoring and management. Our main contribution is a new heuristic with three components: the pre-processing phase, the greedy solution construction, and the post-optimization phase. We perform extensive experiments on graphs of vertex size varying from several thousand to tens of millions. The obtained results show that our algorithm provides a better trade-off between the solution quality and the computation time than existing methods. In particular, it helps to improve the solutions of the method currently used by our industrial partner. All in all, our new algorithm becomes the state-of-the-art approach proposed to solve the M*k*DSP on very large-scale graphs of social networks with million vertices and edges.

Table 4 Result of algorithms on large graphs

Data	HEU_1		HEU_3		HEU_4	
	Sol	***T*** **(s)**	**Sol**	***T*** **(s)**	**Sol**	***T*** **(s)**
$k = 1$						
soc-delicious	215261	19.07	*56066*	1464.84	56600	5679.63
soc-flixster	1452450	999	_	_	*91543*	27374.44
hugebubbles	1213638	2087.83	_	_	*1169394*	7498.20
soc-livejournal	1538044	2689.72	_	_	*930632*	75185.96
soc-partner-1	6263241	64228.04	_	_	*29278*	26740.42
soc-partner-2	4129393	19109	_	_	*38303*	38644.65
$k = 2$						
soc-delicious	34516	3.57	_	_	*8155*	2064.00
soc-flixster	48789	85.93	_	_	*9860*	23694.58
hugebubbles	943233	792.96	_	_	*777960*	41874.54
soc-livejournal	447552	1582.87	_	_	*189121*	16728.22
soc-partner-1	*11102*	59.78	_	_	12200	31962.21
soc-partner-2	20343	84.15	_	_	*19896*	27159.79
$k = 3$						
soc-delicious	14806	2.44	_	_	*1505*	1695.77
soc-flixster	20996	29.71	_	_	*313*	3333.45
hugebubbles	843077	649.47	_	_	*688817*	17221.76
soc-livejournal	211894	394.98	_	_	*83710*	42600.51
soc-partner-1	6337	55.57	_	_	*5158*	5200.14
soc-partner-2	12807	78.3	_	_	*10905*	5481.59

Abbreviations

MDSP: Minimum dominating set problem; M*k*DSP: Minimum *k*-dominating set problem; MILP: Mixed integer linear programming; MWDS: Minimum weight dominating set.

Acknowledgements

This work is finished during the research stay of the corresponding author at the Vietnamese Institute for Advanced Studies in Mathematics (VIASM). He wishes to thank this institution for their kind hospitality and support.

Authors' contributions

The first author provided the ideas and implemented the algorithms. The second and third authors provided the ideas for the algorithms and verified their performance. The last author provided and processed the data. All authors read and approved the final manuscript.

Funding

The authors gratefully acknowledge the support from the UTS-VNU Joint Technology and Innovation Research Centre (JTIRC).

Availability of data and materials

The data including two very large-scale instances from the industrial partner are available upon request.

Competing interests

The authors declare that they have no competing interests.

Author details

[1] ORLab, VNU University of Engineering and Technology, Hanoi, Vietnam. [2] ORLab, Faculty of Computer Science, Phenikaa University, Hanoi, Vietnam. [3] Faculty of Engineering and Information Technology, University of Technology Sydney, Sydney, Australia. [4] FPT Technology Research Institute, FPT University, Hanoi, Vietnam.

Received: 21 February 2020 Accepted: 25 June 2020
Published online: 20 July 2020

References

1. Wang F, Du H, Camacho E, Xu K, Lee W, Shi Y, Shan S. On positive influence dominating sets in social networks. Theor Comput Sci. 2011;412(3):265–9.
2. Wang G, Wang H, Tao X, Zhang J. Finding weighted positive influence dominating set to make impact to negatives: a study on online social networks in the new millennium. In: Kaur H, Tao X, editors. ICTs and the millennium development goals, vol. 412. Berlin: Springer; 2014. p. 67–80.
3. Khomami MMD, Rezvanian A, Bagherpour N, Meybodi MR. Minimum positive influence dominating set and its application in influence maximization: a learning automata approach. Appl Intell. 2018;48:570–93.
4. Yua J, Wang N, Wang G, Yu D. Connected dominating sets in wireless ad hoc and sensor networks—a comprehensive survey. Comput Commun. 2013;36(2):121–34.
5. Wuchty S. Controllability in protein interaction networks. Proc Natl Acad Sci. 2014;111:7156–60.
6. Nacher JC, Akutsu T. Minimum dominating set-based methods for analyzing biological networks. Methods. 2016;102:57–63.
7. Östergård PRJ. Constructing covering codes by tabu search. J Comb Des. 1997;5(1):71–80.
8. Campan A, Truta TM, Beckerich M. Approximation algorithms for *d*-hop dominating set problem. In: 12th international conference on data mining. 2016. p. 86–91.
9. Li X, Zhang Z. Two algorithms for minimum 2-connected *r*-hop dominating set. Inf Process Lett. 2010;110(22):986–91.
10. Michael Q, Rieck SP, Dhar S. Distributed routing algorithms for wireless ad hoc networks using *d*-hop connected *d* -hop dominating sets. Comput Netw. 2005;47(6):785–99.
11. Rooij JMMv, Bodlaender HL. Exact algorithms for dominating set. Discret Appl Math. 2011;159(17):2147–64.
12. Grandoni F. A note on the complexity of minimum dominating set. J Discret Algorithms. 2006;4(2):209–14.
13. Ugurlu O, Tanir D. A hybrid genetic algorithm for minimum weight dominating set problem. In: Zadeh L, Yager R, Shahbazova S, Reformat M, Kreinovich V, editors. Recent developments and the new direction in soft-computing foundations and applications, vol. 361., Studies in fuzziness and soft computingBerlin: Springer; 2018. p. 137–48.
14. Albuquerque M, Vidal T. An efficient matheuristic for the minimum-weight dominating set problem. Appl Soft Comput. 2018;72:527–38.
15. 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). 2018. p. 1514–22.
16. Brandstädt A, Dragan FF. A linear-time algorithm for connected *r*-domination and steiner tree on distance-hereditary graphs. Networks. 1998;31:177–82.
17. Dragan F. Ht-graphs: centers, connected *r*-dominated and steiner trees. Comput Sci J Moldova. 1993;1(2):64–83.
18. Borradaile G, Le H. Optimal dynamic program for r-domination problems over tree decompositions. In: 11th international symposium on parameterized and exact computation—IPEC 2016, Aarhus, Denmark. 2016.
19. Coelho RS, Moura PFS, Wakabayashi Y. The *k*-hop connected dominating set problem: hardness and polyhedra. Electron Notes Discret Math. 2015;50:59–64.
20. Coelho RS, Moura PFS, Wakabayashi Y. The k-hop connected dominating set problem: approximation and hardness. J Comb Optim. 2017;34:1060–83.
21. Nguyen MH, Hà MH, Hoang DT, Nguyen DN, Dutkiewicz E, Tran T. An efficient algorithm for the k-dominating set problem on very large-scale networks (extended abstract). In: Tagarelli A, Tong H, editors. Computational data and social networks—8th international conference, CSoNet 2019, Ho Chi Minh City, Vietnam, November 18–20, 2019, proceedings. Lecture notes in computer science, vol. 11917. p. 74–6.
22. Rossi RA, Ahmed NK. The network data repository with interactive graph analytics and visualization. In: AAAI'15: proceedings of the twenty-ninth AAAI conference on artificial intelligence. 2015. p. 4292–3.

Publisher's Note