VNS with GA-based parameter tuning for solving the k-domination problem

Anonymous Author(s)

ABSTRACT

In this paper we are concerned with solving a generalized version of the well-known minimum dominating set problem, the so-called k-domination problem, $k \in \mathbb{N}$. This problem is about finding a minimal cardinality subset D of vertices of a graph $G = (V, E)$ such that every $v \in V$ belongs to D or has at least k neighbors from D. The k-domination problem has applications in distributed systems, biological networks etc. We propose a variable neighborhood search (VNS) metaheuristic for solving the k-domination problem. The VNS is equipped with an efficient fitness function that allows it to consider both feasible and infeasible solutions, while appropriately penalizing infeasible solutions. The control parameters of the VNS are tuned using a genetic algorithm. The method is compared to the best known heuristic approaches from the literature: the beam search and several greedy approaches. Experimental evaluations are performed on a real-world benchmark set whose instances represent the road networks of different cities. The VNS provided new state-of-the-art results for all considered problem instances with $k \in \{1, 2, 4\}$.

CCS CONCEPTS

• **Theory of computation → Optimization with randomized search heuristics**; *Evolutionary algorithms.*

KEYWORDS

Variable neighborhood search, graph domination, genetic algorithm, parameter tuning

ACM Reference Format:

Anonymous Author(s). 2023. VNS with GA-based parameter tuning for solving the k-domination problem. In *GECCO '23: xx, xx, xx*. ACM, New York, NY, USA, 7 pages. https://doi.org/xxxx

1 INTRODUCTION

A graph $G = (V, E)$ is an abstract mathematical structure in which V represents a set of elements called vertices (or nodes) and a set of pairs $e = uv = (u, v) \in E \subseteq V \times V$ called edges of G. In this paper, we are concerned with simple undirected graphs that have no loops and where the edges have no directions, i.e., $e = \{u, v\} = uv = vu \in E$. Graphs serve as models for many real-world problems describing relationships among various objects in biology, physics, social networks, etc. [12, 19, 23, 26].

One of the best known classes of problems studied from theoretical, computational and practical points of view are domination problems on graphs [11]. The basic problem of this class is the *minimum dominating set problem*. The subset $D \subset V$ is called *dominating set* if every vertex $v \in V$ belongs to D or there is at least one vertex $w \in D$ such that $uw \in E$. The search for the smallest possible dominating set D of the graph G is called *the minimum dominating set problem* (MDSP) [10]. This problem has many applications, for example, in biological networks [21], document summarization [27], graph mining [2], etc. From an algorithmic point of view, this problem is NP-hard. It is solved by various exact approaches, such as branch-and-reduce algorithms [28], an approach that uses the fundamental cut-sets of the graph [17], etc. Heuristic approaches are more common in the literature, e.g., a genetic algorithm [13], simulated annealing [14], ant colony optimization [16], to name a few. Several generalizations of MDSP, arising from practical experience, are proposed in the literature: the minimum weight dominating set problem [25], the minimum total dominating set problem [30], the minimum connected dominating set problem [1], etc.

In this paper, we study the *minimum k-domination problem* (MkDP) [4], for a fixed $k \in \mathbf{N}$. The *k-dominating* set D of a graph G is such a subset of V that every vertex not belonging to D is adjacent to at least k vertices in D [18]. A minimum k-dominating set represents the optimal solution of MkDP. The NP-completeness of the k-domination decision problem is formally proved for the split graphs, see details in the above citation. (Note that the definition of this problem is not unique in the literature – there is a definition where the goal is to find a minimum cardinality vertex set D such that every vertex of G is within distance k of some vertex in D [3].)

Regarding previous solutions to this problem, several greedy approaches have been proposed by Couture et al. [5], Gagarin et al. [9], and Gagarin and Corcoran [8]. Recently, Corcoran and Gagarin proposed a Beam search approach [4], which is currently the best-performing heuristic approach for small to medium sized real-world instances.

MkDP has applications in distributed systems [29], where a k-dominating set in these systems is a set of processors such that each processor outside the must have at least k neighbors in the set.

We propose a variable neighborhood search (VNS) metaheuristic for solving MkDP. The contributions of our work can be summarized as follows:

- The control parameters of the VNS are tuned with genetic algorithm `PyGAD` [7].
- VNS for MkDPA significantly improves the state-of-the-art results for all considered problem instances, where $k \in \{1, 2, 4\}$.
- VNS is able to quickly produce solutions of reasonable quality which is not the case for approaches previously proposed in the literature.

GECCO '23, 2023, Lisbon, Portugal

ACM ISBN xxxxx…$15.00
https://doi.org/xxxx

2 FORMAL PROBLEM DEFINITION

Let $G = (V, E)$ be a simple undirected graph and $k \in \mathbb{N}$ be fixed. For $v \in V$, $N(v)$ is a set of all adjacent vertices of v in the graph G, i.e. $N(v) = \{w \mid vw \in E\}$. A set $D \subseteq V$ is called k-dominating set if for every $v \in V \setminus D$ holds $|N(v) \cap D| \geq k$. The MkDP is an optimization problem whose objective is to find a k-dominating set with minimal cardinality:

$$\arg\min_{D \subseteq V} |D| \tag{1}$$

$$s.t.\ \forall v \in V \setminus D, |N(v) \cap D| \geq k. \tag{2}$$

As for the search space of the MkDP, we adapt it to the needs of our VNS. Not only feasible solutions are handled, but also infeasible ones – they are additionally penalized, see Section 3. In other words, every subset of V is a candidate solution in our VNS. Therefore, the size of the search space is $2^{|V|}$.

3 THE PROPOSED ALGORITHM

In this section, we first give an overview of the variable neighborhood search (VNS). Then we introduce the main components of VNS for solving MkDP: the fitness function, the shaking procedure and the efficient local search.

3.1 Variable neighborhood search

Variable neighborhood search is a single-based solution metaheuristic proposed by Mladenović and Hansen [20]. The basic idea of the approach is to systematically exchange neighborhoods of the current best solution (incumbent solution) to avoid getting stuck in a local optimum. VNS has proven to be one of the most powerful metaheuristic, achieving excellent results on diverse classes of problems, such as scheduling problems [6], vehicle routing problems [24], median problems [15], etc.

The VNS for solving the MkDP is given in Algorithm 1.

VNS generally requires at least two control parameters $d_{min}, d_{max} \in \mathbb{N}$, which define the increasing sizes of the neighborhood structures $\mathcal{N}_{d_{min}}(D), \ldots, \mathcal{N}_{d_{max}}(D)$ around given solution D. A solution D' belongs to the neighborhood $\mathcal{N}_d(D)$ of the solution D if it can be obtained from D by removing $\min(d, |D|)$ vertices from D and then adding d vertices from $V \setminus D$ into D.

The third parameter commonly used in VNS is $p_{move} \in [0, 1]$. This parameter corresponds to the probability of moving to a new solution if it has the same quality (fitness) as the incumbent solution. Finally, the fourth control parameter *penalty* is specific to MkDP. It is real-valued and is used to define the relative influence of solution feasibility and solution quality (size of k-dominating set) on the overall value of the fitness function (more details are given in Section 3.2). The return value of VNS is called D_{best} – it is the incumbent solution.

At the beginning, the neighborhood size d is set to the smallest, i.e. to d_{min}. The initial incumbent solution D_{best} is generated by performing local search on an empty set (local search is explained in Section 3.3). Then the algorithm enters the main loop (lines 5-18). The loop is run until at least one of the termination criteria is met. At each iteration of the loop, the following steps are executed:

Algorithm 1 VNS scheme for solving MkDP

```
1:  Input: d_min, d_max, p_move, penalty
2:  Output: best found solution D_best
3:  d ← d_min, d_max_init ← d_max
4:  D_best ← LocalSearch(∅)
5:  while TerminationCriteriaNotMet() do
6:    D' ← Shaking(N_d(D_best))
7:    D'' ← LocalSearch(D')
8:    if fitness(D'') < fitness(D_best) ∨ (fitness(D'') = fitness(D_best) ∧ r ∈ U_[0,1] < p_move) then
9:      D_best ← D''
10:     d ← d_min
11:     d_max ← min(d_max_init, |D_best|/2)
12:   else
13:     d ← d + 1    // try with next neighborhood
14:     if d > d_max then
15:       d ← d_min
16:     end if
17:   end if
18: end while
19: return D_best
```

- `Shaking`$(\mathcal{N}_d(D_{best}))$ – a solution D' is selected randomly from the set of solutions belonging to the d-th neighborhood structure around the solution D_{best}.
- `LocalSearch` – the selected solution D' may be improved by a local search procedure, as explained in Section 3.3.
- The solution D' becomes new incumbent if it has better fitness than the previous incumbent. Alternatively, it may become new incumbent with probability p_{move} if its fitness is the same. In both cases, d is reset to $d_{\min}$. The parameter d_{max} is dynamically set to $\min(d_{\max\_init}, |D_{best}|/2)$ to prevent neighborhoods that are too large, i.e., neighborhoods larger than half the incumbent size.
- If the solution D' does not become new incumbent, d is increased – this further increases diversification. If this increase leads to $d > d_{max}$, d is circularly reset to d_{min}.

3.2 Fitness function

To evaluate the solution D, the following nonlinear *fitness* function is used:

$$fitness(D) = (1 + violations(D)) \cdot (1 + penalty \cdot |D|) \tag{3}$$

where

$$violations(D) = \sum_{v \in V \setminus D} k - C(D, v) \tag{4}$$

and

$$C(D, v) = \min(k, |N(v) \cap D|). \tag{5}$$

It can be seen that $violations(D)$ quantifies the overall degree of solution D inadmissibility, i.e., for each vertex v that does not belong to a candidate dominating set D, $k - C(D, v)$ measures how strongly vertex v locally violates the k-domination condition. Thus, $C(D, v)$ quantifies the opposite – how strongly the vertex v satisfies the k-domination condition. In particular, when vertex v has k or

more vertex neighbors in D, the value of $k - \min(k, |N(v) \cap D|)$ is zero. Otherwise, $C(D, v)$ is positive and at most k.

Therefore, the proposed fitness function evaluates both feasible and infeasible solutions. Since the fitness function is to be minimized, the following three observations can be made about the values of the fitness function:

- For sufficiently small values of the parameter *penalty*, the feasibility of the solution is relatively preferred over the cardinality of the solution. This means that when comparing feasible and infeasible solutions, the feasible solution is favored.
- When comparing two infeasible solutions of the same cardinality, the *less infeasible* solution is preferred, i.e. the one with the lower $violations(\cdot)$ value.
- When comparing two feasible solutions, the one with the lower cardinality is preferred.

3.3 Local search

The goal of local search (LS) is to improve the solution D' obtained in the shaking phase by applying multiple local improvements to the structure of the solution. The LS tailored to MkDP is described in Algorithm 2.

Our LS procedure consists of two phases. In the first phase, the vertices are added to achieve feasibility – when the solution is not feasible. In the second phase, the vertices are removed to improve the objective function. The removal is done in a way that does not affect the feasibility previously achieved. In both phases, the best improvement strategy is used. This means that all eligible vertices are checked for inclusion/exclusion and then the best vertex is included/excluded. Each phase ends with the first iteration where no improvement in fitness is found.

Fast fitness evaluation. Computing the fitness function (3) takes $O(|E|)$ time – the most time consuming part is computing the $violations(\cdot)$ function. For dense graphs, this complexity can go up to $O(|V|^2)$. Since local search makes many fitness function calls it is too costly to compute the fitness function from scratch every time. In order to execute `LocalSearch` procedure efficiently, fast calculations of the fitness function are applied. Before LS enters the first `while` loop, $C(D_{best}, v), \forall v \in V$ for given solution D_{best} are computed and saved. During vertex addition, that is $D' = D_{best} \cup \{v\}$, $violations(D')$ is calculated in the following way:

$$\begin{aligned} violations(D') = violations(D_{best}) \\ - (k - C(D_{best}, v))\mathtt{I}_{(C(D_{best},v)<k)} \\ - \sum_{w \in N(v) \setminus D_{best}} \mathtt{I}_{(C(D_{best},w)<k)} \end{aligned} \quad (6)$$

where I stands for the indicator function.

The part $-(k - C(D_{best}, v))\mathtt{I}_{(C(D_{best},v)<k)}$ adjusts for the effect of the added vertex v, i.e., if v was previously *satisfied* before (satisfied the k-domination condition), then this indicator function takes the value zero, so nothing changes. This is true because v remains satisfied – it is now part of k-dominating set. Otherwise, if vertex v has previously violated the k-domination condition, the total violations are reduced by the previous degree of vertex v violation, i.e. $(k - C(D_{best}, v))$.

Algorithm 2 `LocalSearch`

Input: a solution D
Output: a (possibly) improved solution D_{best}
$D_{best} \leftarrow D$
$best_{fit} \leftarrow fitness(D_{best})$
// first, achieve feasibility by adding vertices
improved $\leftarrow$ True
while *improved* **do**
  $improved \leftarrow False$
  $best_v \leftarrow$ None
  for $v \in V \setminus D_{best}$ **do**
    $D' \leftarrow D_{best} \cup \{v\}$
    if $new_{fit} = fitness_{fast}(D') < best_{fit}$ **then**
      $best_v \leftarrow v$
      $best_{fit} \leftarrow new_{fit}$
      $improved \leftarrow True$
    end if
  end for
  if *improved* **then**
    $D_{best} \leftarrow D_{best} \cup \{best_v\}$
  end if
end while
// second, remove vertices, but keep the feasibility
$improved \leftarrow True$
while *improved* **do**
  $improved \leftarrow False$
  $best_v \leftarrow$ None
  for $v \in D_{best}$ **do**
    $D' \leftarrow D_{best} \setminus \{v\}$
    if $new_{fit} = fitness_{fast}(D') < best_{fit}$ **then**
      $best_v \leftarrow v$
      $best_{fit} \leftarrow new_{fit}$
      $improved \leftarrow True$
    end if
    if *improved* **then**
      $D_{best} \leftarrow D_{best} \setminus \{best_v\}$
    end if
  end for
end while
return D_{best}

The part $-\sum_{w \in N(v) \setminus D_{best}} \mathtt{I}_{(C(D_{best},w)<k)}$ adjust for the effect on the vertex v neighbors, which were not in the k-dominating set D_{best}, i.e. $W = N(v) \setminus D_{best}$. If some of these vertices $w \in W$ violated k-domination condition, adding the adjacent vertex v to the solution reduces the total violations. Otherwise, if a vertex w was satisfied, adding v to the solution does not change anything.

In case of a vertex removal, the similar idea is used.

4 EXPERIMENTAL EVALUATION

In this section we analyze the quality of the proposed VNS method.[1] For this purpose, we include the competing heuristic approaches

[1] All instances, source codes of the VNS and GA tuning algorithms, and detailed experimental results are available on the GitHub page for this work: anonymousURL.

Table 1: Large-sized instance characteristics.

City	$\|V\|$	$\|E\|$
Belgrade	19,586	7,561,185
Berlin	29,461	9,944,851
Boston	44,797	28,164,740
Dublin	37,982	21,630,466
Minsk	10,487	1,375,618

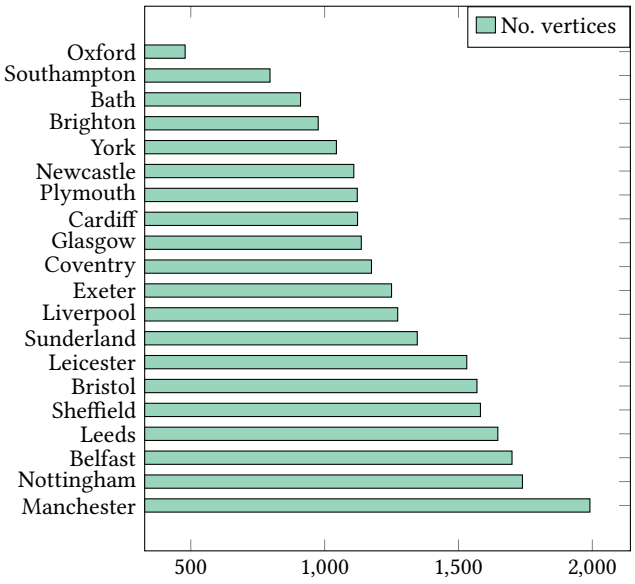

Figure 1: The number of vertices in the small to medium sized instances.

from the literature. The following four methods for MkDP are compared:

- The standard greedy method from [9, 22], denoted by SG;
- Greedy method from [8], denoted by PG;
- Beam search approach from [4], denoted by BS;
- VNS approach, as described in Section 3, denoted by VNS.

Benchmark instances. For comparison, we consider the benchmark introduced in [4]. It consists of 20 small to medium sized instances, where all instances represent road networks of different cities modeled as reachability graphs. In addition to these, the authors have provided the program that generate road networks for five large cities: Belgrade, Berlin, Boston, Dublin, and Minsk. Since the road networks change over time, the obtained graphs do not fully match those used in the [4], so their properties are given in Table 1.

The characteristics of 20 small to medium sized instances can be found in [4]. Unlike the five large networks, these 20 instances are exactly the same as in [4] – we obtained them directly from the authors. Figures 1–2 show the number of vertices and edges of these instances.

Testing environment. Experiments were performed on a computer running Intel Core i9-9900KF CPU @3.6GHz with 64GB RAM, under Microsoft Windows 10 Pro OS. VNS is implemented in Python 3.9.

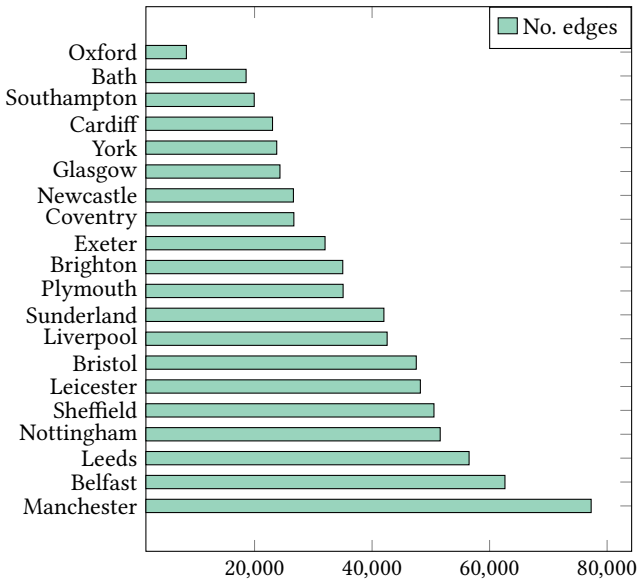

Figure 2: The number of edges in the small to medium sized instances.

The results of SG, PG and BS for small to medium sized instances are taken from [4] as reported in their experimental section. The results for the five large instances were obtained by running the original implementation of PG (obtained from the authors). According to the same authors, BS was inefficient for the large instances due to its high computational complexity. Therefore, we did not include this algorithm in the large instance comparison. PG and VNS algorithms are run ten times (using different random seeds) per each problem instance.

The termination criteria of the VNS are: (*i*) the maximum running time of 30 minutes, and (*ii*) the maximum number of 2000 iterations. The time limit of 30 minutes is not checked during initialization (line 4 in Algorithm 1). This means that VNS can take more than 30 minutes to finish for some very large instances, such as Dublin and Boston.

4.1 Parameters tuning

As mentioned earlier, VNS for MkDPA involves four control parameters: d_{min}, d_{max}, p_{move} and *penalty*. The range of parameters used for tuning are given in Table 2.

Table 2: VNS parameter ranges.

Parameter	Domain	Step
d_{min}	[1, 10]	1
d_{max}	[2, 100]	1
p_{move}	[0, 1]	0.05
penalty	[0.001, 0.02]	0.001

As for the tuning tool, we have decided to use genetic algorithm called `PyGAD`. This implementation, proposed in [7], is available as

an open-source Python library. All (20) small to medium sized problem instances were tuned separately for each $k \in \{1, 2, 4\}$, resulting in a total of 60 control parameter configurations. `PyGAD` was used as an outer optimizer calling the inner VNS method in a multiprocessing manner – each `PyGAD` chromosome (solution) encodes a single VNS control parameter configuration and corresponds to a separate process. VNS was limited to 100 iterations. The `PyGAD` parameters were as follows:

- *num_generations = 50* – number of generations;
- *num_parents_mating = 5* – number of chromosomes selected as parents for the mating pool;
- *sol_per_pop = 10* – number of chromosomes in the population;
- *mutation_probability = 0.1* – the mutation probability;
- *num_genes = 4* – number of genes in the chromosome, the same as the number of control parameters of VNS.

Parameter tuning for large instances was too inefficient. Therefore, we used reasonable manually configured parameters for these instances: $d_{min} = 1, d_{max} = +\infty, p_{move} = 0.5, penalty = 0.01$. (Note that d_{max} has implicit upper bound – it is set in line 11 of Algorithm 1.)

4.2 Experimental results

Tables 3-5 contain the experimental results for $k \in \{1, 2, 4\}$, respectively. The first column gives the name of the instance (city) for which the results are reported. The next block contains the results of VNS – the best result and the average solution quality over ten runs. The third block reports the results of the best approach from the literature for each instance – the name of the approach, the best obtained result and average solution quality over ten runs. Note that the labels BS1, BS2 or BS4 correspond to BS approach with beam widths of 1,2 and 4, respectively.

The following conclusions can be drawn from these results.

- For $k = 1$, VNS outperformed all competing approaches for small to medium sized instances. VNS also outperformed the PG approach for the large-sized instances.
- Concerning the results for $k = 2$, VNS outperformed all competing approaches by nearly 15% in terms of the average solution quality. Similar conclusions hold for the large instances: VNS produces $\approx$ 5–15% improvement rate over the second best PG approach.
- The situation is similar for $k = 4$. VNS outperforms the second-best algorithm by more than 15% in some cases (see, for example, the instance `Nottingham` where VNS achieved a score 165 over BS4 which achieved a score 193). For the large instances, VNS again outperforms PG in all cases.

The average times (in seconds) for reaching the best solution in VNS are shown as stacked bar graphs in Figures 3-4. The smaller the size of an instance, the less time VNS takes on average to reach the best solution. Figures 3-4 clearly show that as k increases, the average times to reach the best solutions increase rapidly. In case of the large instances, it can be seen from Figure 4 that the most time consuming instances in terms of reaching the best solution are `Boston` and `Dublin`. These two instances are indeed the largest in terms of number of vertices/edges (see Table 1).

Table 3: Results for k=1.

	VNS		Literature		
City	Best	Avg.	Alg.	Best	Avg.
Bath	**38**	38	BS4	43	44.6
Belfast	**39**	39	BS4	48	50.2
Brighton	**21**	21	BS4	28	28.2
Bristol	**37**	37	BS2	46	47.4
Cardiff	**39**	39	BS4	48	50.6
Coventry	**38**	38.2	BS4	44	44.8
Exeter	**38**	38	BS4	50	50.6
Glasgow	**50**	50.2	BS4	58	59.2
Leeds	**40**	40	BS4	51	52.4
Leicester	**38**	38	BS4	51	51.5
Liverpool	**28**	28	BS4	38	38.4
Manchester	**38**	38.8	BS4	45	45.9
Newcastle	**44**	44	BS4	51	52.6
Nottingham	**44**	44	BS4	55	56.6
Oxford	**24**	24	BS4	27	27.9
Plymouth	**31**	31	BS4	39	40.3
Sheffield	**42**	42	BS4	51	52.5
Southampton	**25**	25	BS4	28	29.6
Sunderland	**36**	36	BS4	46	46.3
York	**32**	32	BS4	39	39.1
Belgrade	**85**	88.4	PG	103	103.4
Berlin	**102**	104.5	PG	125	125.9
Boston	**99**	101.2	PG	101	102.7
Dublin	**93**	99	PG	112	113.8
Minsk	**100**	102	PG	125	126

5 CONCLUSIONS AND FUTURE WORK

In this paper we have studied the k-domination problem, $k \in \mathbb{N}$, a generalized version of the prominent minimum dominating set problem. This problem has been solved so far by several constructive and incremental approaches. We proposed the variable neighborhood search (VNS) metaheuristic to solve this problem. It is equipped with an effective fitness function that evaluates both feasible and infeasible solutions. Moreover, an efficient local search procedure with best-improvement strategy and fast fitness function evaluation plays an important role in obtaining high-quality solutions. The efficiency of our VNS has been validated on the real-world benchmark, where it has been shown that VNS outperform all existing heuristic state-of-the-art approaches.

For future work, one could consider improving the VNS to work more efficiently with very large graphs, such as social networks. Also, VNS could be compared to exact approaches such as integer linear programming models, solved by state-of-the art solvers, such as Cplex or Gurobi.

ACKNOWLEDGMENTS

We thank Padraig Corcoran and Andrei Gagarin for providing problem instances and their algorithm implementations.

REFERENCES

[1] Sergiy Butenko, Xiuzhen Cheng, Carlos A Oliveira, and Panos M Pardalos. 2004. A new heuristic for the minimum connected dominating set problem on ad hoc

Table 4: Results for $k = 2$.

	VNS		Literature		
City	Best	Avg.	Alg.	Best	Avg.
Bath	**71**	71.8	BS1	86	89
Belfast	**76**	76.2	BS4	96	97.6
Brighton	**40**	41.1	BS4	49	49.4
Bristol	**73**	73.7	BS4	91	94
Cardiff	**79**	79.4	BS4	92	95.6
Coventry	**73**	73.2	BS4	84	85.1
Exeter	**77**	77.3	BS4	94	95.7
Glasgow	**93**	94.2	BS4	108	110.6
Leeds	**79**	80.5	BS4	98	99.6
Leicester	**75**	75.2	BS4	93	94.1
Liverpool	**57**	57	BS4	71	72
Manchester	**77**	78.4	BS4	90	91.5
Newcastle	**83**	84.3	BS4	94	95.4
Nottingham	**84**	85.3	BS4	102	103.3
Oxford	**47**	47	BS4	54	54.9
Plymouth	**62**	62.1	BS4	73	75
Sheffield	**84**	84.7	BS4	97	98.9
Southampton	**50**	50.1	BS4	60	61.1
Sunderland	**73**	73.7	BS4	87	89.1
York	**68**	68	BS4	77	77.6
Belgrade	**169**	172.3	PG	196	197.3
Berlin	**206**	207.1	PG	239	240.1
Boston	**181**	194.6	PG	190	191.6
Dublin	**181**	184.4	PG	209	211.3
Minsk	**197**	201.1	PG	238	240.4

Table 5: Results for $k = 4$.

	VNS		Literature		
City	Best	Avg.	Alg.	Best	Avg.
Bath	**139**	140.4	BS4	159	160
Belfast	**147**	148.5	BS4	177	179.6
Brighton	**78**	79.3	BS4	92	94.8
Bristol	**145**	146.7	BS4	175	176.4
Cardiff	**160**	161.9	BS4	181	183.2
Coventry	**150**	150.6	BS4	170	172.6
Exeter	**157**	158.3	BS4	181	182.3
Glasgow	**174**	175.4	BS4	197	199.8
Leeds	**152**	152.4	BS4	186	187.1
Leicester	**151**	152.6	BS4	175	177.7
Liverpool	**112**	114	BS4	132	133
Manchester	**154**	156.4	BS4	177	178.5
Newcastle	**153**	154.8	BS2	169	171.5
Nottingham	**165**	166.6	BS4	193	195.2
Oxford	**89**	89.2	BS2	99	100.8
Plymouth	**115**	116	BS4	135	137
Sheffield	**160**	161.6	BS4	180	182.2
Southampton	**97**	97.8	BS4	112	113.2
Sunderland	**142**	142.5	BS4	162	163.6
York	**129**	130.1	BS4	144	145.8
Belgrade	**344**	346.3	PG	372	374.5
Berlin	**408**	409.2	PG	444	446.2
Boston	**341**	341	PG	367	368.7
Dublin	**363**	363	PG	388	390.2
Minsk	**387**	391.5	PG	454	457.6

wireless networks. In *Recent developments in cooperative control and optimization*. Springer, 61–73.

[2] David Chalupa. 2018. An order-based algorithm for minimum dominating set with application in graph mining. *Information Sciences* 426 (2018), 101–116.

[3] Gerard Jennhwa Chang. 1983. K-DOMINATION AND GRAPH COVERING PROBLEMS. (1983).

[4] Padraig Corcoran and Andrei Gagarin. 2021. Heuristics for k-domination models of facility location problems in street networks. *Computers & Operations Research* 133 (2021), 105368.

[5] Mathieu Couture, Michel Barbeau, Prosenjit Bose, and Evangelos Kranakis. 2006. Incremental construction of k-dominating sets in wireless sensor networks. In *International Conference On Principles Of Distributed Systems*. Springer, 202–214.

[6] Krzysztof Fleszar and Khalil S Hindi. 2004. Solving the resource-constrained project scheduling problem by a variable neighbourhood search. *European Journal of Operational Research* 155, 2 (2004), 402–413.

[7] Ahmed Fawzy Gad. 2021. Pygad: An intuitive genetic algorithm python library. *arXiv preprint arXiv:2106.06158* (2021).

[8] Andrei Gagarin and Padraig Corcoran. 2018. Multiple domination models for placement of electric vehicle charging stations in road networks. *Computers & Operations Research* 96 (2018), 69–79.

[9] Andrei Gagarin, Anush Poghosyan, and Vadim Zverovich. 2013. Randomized algorithms and upper bounds for multiple domination in graphs and networks. *Discrete Applied Mathematics* 161, 4-5 (2013), 604–611.

[10] Fabrizio Grandoni. 2006. A note on the complexity of minimum dominating set. *Journal of Discrete Algorithms* 4, 2 (2006), 209–214.

[11] Teresa W Haynes, Stephen Hedetniemi, and Peter Slater. 2013. *Fundamentals of domination in graphs*. CRC press.

[12] Teresa W. Haynes, Sandra M. Hedetniemi, Stephen T. Hedetniemi, and Michael A. Henning. 2002. Domination in Graphs Applied to Electric Power Networks. *SIAM Journal on Discrete Mathematics* 15, 4 (2002), 519–529. https://doi.org/10.1137/S0895480100375831

[13] Abdel-Rahman Hedar and Rashad Ismail. 2010. Hybrid genetic algorithm for minimum dominating set problem. In *International conference on computational science and its applications*. Springer, 457–467.

[14] Abdel-Rahman Hedar and Rashad Ismail. 2012. Simulated annealing with stochastic local search for minimum dominating set problem. *International Journal of Machine Learning and Cybernetics* 3, 2 (2012), 97–109.

[15] Alberto Herrán, J Manuel Colmenar, and Abraham Duarte. 2019. A variable neighborhood search approach for the Hamiltonian p-median problem. *Applied Soft Computing* 80 (2019), 603–616.

[16] Chin Kuan Ho, Yashwant Prasad Singh, and Hong Tat Ewe. 2006. An enhanced ant colony optimization metaheuristic for the minimum dominating set problem. *Applied Artificial Intelligence* 20, 10 (2006), 881–903.

[17] Ali KARCİ. 2020. New algorithms for minimum dominating set in any graphs. *Computer Science* 5, 2 (2020), 62–70.

[18] James K Lan and Gerard Jennhwa Chang. 2013. Algorithmic aspects of the k-domination problem in graphs. *Discrete Applied Mathematics* 161, 10-11 (2013), 1513–1520.

[19] Alireza R Mashaghi, Abolfazl Ramezanpour, and Vahid Karimipour. 2004. Investigation of a protein complex network. *The European Physical Journal B-Condensed Matter and Complex Systems* 41, 1 (2004), 113–121.

[20] Nenad Mladenović and Pierre Hansen. 1997. Variable neighborhood search. *Computers & operations research* 24, 11 (1997), 1097–1100.

[21] Jose C Nacher and Tatsuya Akutsu. 2016. Minimum dominating set-based methods for analyzing biological networks. *Methods* 102 (2016), 57–63.

[22] Abhay K Parekh. 1991. Analysis of a greedy heuristic for finding small dominating sets in graphs. *Information processing letters* 39, 5 (1991), 237–240.

[23] Shariefuddin Pirzada. 2007. Applications of graph theory. In *PAMM: Proceedings in Applied Mathematics and Mechanics*, Vol. 7. Wiley Online Library, 2070013–2070013.

[24] Dhekra Rezgui, Jouhaina Chaouachi Siala, Wassila Aggoune-Mtalaa, and Hend Bouziri. 2019. Application of a variable neighborhood search algorithm to a fleet size and mix vehicle routing problem with electric modular vehicles. *Computers & Industrial Engineering* 130 (2019), 537–550.

[25] QATAR SERBIA Romania. 2010. Ant colony optimization applied to minimum weight dominating set problem. In *Proceedings of the 12th WSEAS international conference on automatic control, modelling and simulation. Catania, Italy*. 29–31.

[26] Preya Shah, Arian Ashourvan, Fadi Mikhail, Adam Pines, Lohith Kini, Kelly Oechsel, Sandhitsu R Das, Joel M Stein, Russell T Shinohara, Danielle S Bassett,

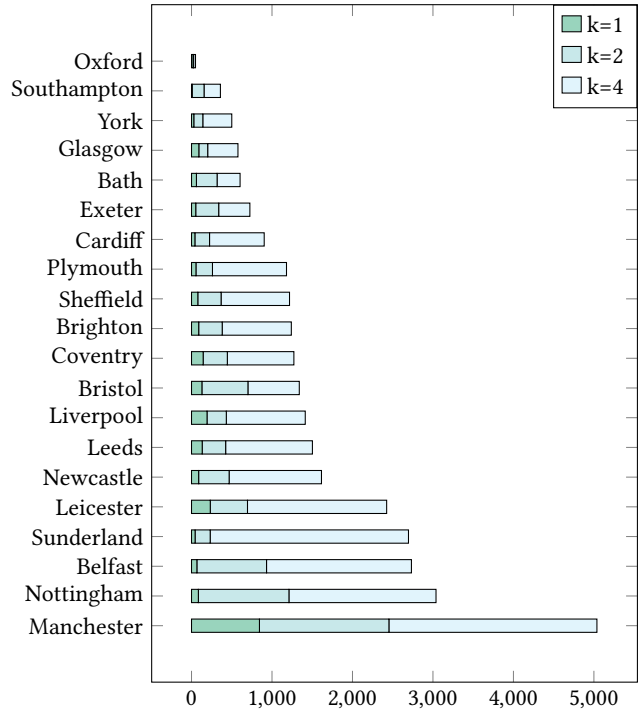

Figure 3: Average times (in seconds) of finding the best solution for small to medium sized instances.

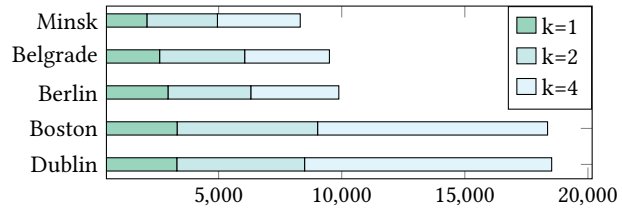

Figure 4: Average times (in seconds) of finding the best solution for large instances.

et al. 2019. Characterizing the role of the structural connectome in seizure dynamics. *Brain* 142, 7 (2019), 1955–1972.

[27] Chao Shen and Tao Li. 2010. Multi-document summarization via the minimum dominating set. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*. 984–992.

[28] Johan MM Van Rooij and Hans L Bodlaender. 2011. Exact algorithms for dominating set. *Discrete Applied Mathematics* 159, 17 (2011), 2147–2164.

[29] Guangyuan Wang, Hua Wang, Xiaohui Tao, Ji Zhang, and Jinhua Zhang. 2013. Minimising k-dominating set in arbitrary network graphs. In *International Conference on Advanced Data Mining and Applications*. Springer, 120–132.

[30] Fuyu Yuan, Chenxi Li, Xin Gao, Minghao Yin, and Yiyuan Wang. 2019. A novel hybrid algorithm for minimum total dominating set problem. *Mathematics* 7, 3 (2019), 222.