Response to the Reviewers' comments for paper "Variable neighborhood search for solving the k-domination problem" (old title: "VNS with GA-based parameter tuning for solving the k-domination problem")

Milan Predojević, Aleksandar Kartelj, and Marko Djukanović

April 19, 2023

We thank the reviewers for their comments. All comments have been carefully considered. Below we address them all and describe the respective changes we have made to the manuscript. The reviewers' comments are quoted for clarity. All changes we made to the paper are highlighted in red.

Comments from *Reviewer #1*

> In this paper an interesting implementation of VNS algorithm is proposed. The only issue I found is missing explanation why the parameters are tuned using Genetic Algorithm and not some other package, for example IRACE? Other than that, the reported results are really outstanding.

Answer.
Thank you.
We used GA to fit into the conference framework – Evolutionary Computation, but later (after submission) we found out that this was not obligatory. Our standard procedure for parameter tuning is grid search or IRACE, so using GA in this situation was unfortunate. In the latest version of the paper, we used the grid search for tuning instead.

Comments from *Reviewer #2*

> The main contributions of the paper are summarized in the introduction (Sec. 1). Some of them, however, are not properly supported by the experiments presented later.

1. Authors claim that the VNS significantly improves the state-of-the-art, but the comparison to the literature is not completely fair. Different from the literature approaches, the VNS uses specific parameter configurations for each instance. Besides that, the termination criteria of the literature approaches are not discussed and contrasted to those used for the VNS, and using different machines or different running time limits (for example) may impact in the quality of the results.

Answer.
In the new version of the paper, a single parameter configuration was used for all instances. The configuration was obtained by the grid search method for a random sample of 20 instances (out of 60 small to medium instances in total: 20 cities for $k \in \{1, 2, 4\}$).

The source code of BS algorithm proposed in [Corcoran and Gagarin(2021)] was not available and therefore we could not make direct comparison for the small to medium instances on the same computer. Therefore, we used the results reported in the paper and come up with a termination criteria effectively similar to those in the cited paper. In our paper, we added a following discussion justifying the termination criteria:

"The algorithms SG, PG and BS terminate their execution as soon as the first feasible solution is reached. Therefore, we decided to use termination criteria which allow similar running times to those reported by comparison algorithms [Corcoran and Gagarin(2021)]. Namely, the best results were obtained with the BS configuration BS4, where 4 stands for the beam width. For this algorithm, in the case of $k = 2$, the average running times in seconds are 1736, 8834, 1257, 7156 and 3327 for the Bath, Belfast, Brighton, Bristol and Cardiff instances, respectively. As stated in [Corcoran and Gagarin(2021)], the comparison algorithms were also implemented in Python and executed on a desktop computer with somewhat better CPU: Intel Core i7-8700 @3.20GHz."

Note also the following sentence, which describes the comparison approach:
"The results of SG, PG and BS for small to medium sized instances are taken from [Corcoran and Gagarin(2021)] as reported in their experimental section. The results for the five large instances were obtained by running the original implementation of PG (acquired 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."

2. Regarding the third conclusion, which states that "VNS is able to quickly produce solutions of reasonable quality which is not the case for approaches proposed in the literature", the paper only shows the quality of the final solutions and the running time required by the VNS to find them, but no information about the solutions found during the execution is given. It would be interesting

to visualize the performance profile of some runs, identifying the best solutions found over time, comparing with the same visualizations for the literature approaches (and then argue that the VNS is the only approach that finds good solutions quickly). Moreover, Figures 3 and 4 show running times larger than 30 minutes (i.e., larger than the termination criterion) for most instances and values of k, which cannot be considered "quickly" without a more informed discussion.

Answer.
This paper is accepted as a poster limited to four pages. For this reason, we are not able to provide a detailed analysis of the requested points. We have removed this sentence from the conclusions.

According to Section 3, the proposed VNS algorithm explores neighborhoods with different sizes (number of vertices added and removed from the solution). However, such variable neighborhood strategy is used only in the shaking procedure, where a neighboring solution of a given local optima is randomly selected from one of these neighborhoods. The local search does not explore different neighborhoods. Instead, it performs a simple iterative (best) improvement procedure. Based on this, can you call this algorithm a VNS?

Answer.
The proposed VNS is well defined in the literature [Mladenović and Hansen(1997)]. The shaking step generally deals with diversification, while the local search step (as part of the VNS) is used for intensification. So, the shaking relocates the solution to its parameterized neighborhood, while local search further investigates this neighborhood thoroughly. There are several variants of VNS (see, for example, Chapter 12, devoted to VNS, in [Burke et al.(2014)]). The variant we use is called Basic VNS. General VNS, for example, uses Variable neighborhood descent (VND) as the local search step, which offers a richer set of neighborhoods. In our case, Basic VNS performs well, but we might consider General VNS or Skewed VNS in a future study.

Regarding the running time limit (30 minutes), it would be interesting to scale the time limit according to the size of instance. Figures 3 and 4 show that the required computational effort increases as instance size grows. Besides that, the paper should mention the termination criteria of the approaches from the literature and how it compares to the one used in the experimental evaluation, to ensure a fair comparison of results.

Answer.
Due to page limitations we are unable to display this scaling.

Termination criteria for comparison algorithms are explained with the sentence: "The algorithms SG, PG and BS terminate their execution as soon as the first feasible solution is reached."

As for the fairness of the comparison, we explained this in the answer to your first question.

> The parameter tuning step, detailed in Section 4.1, does not follow a well-established methodology for this task. First, the instances should be split into training and testing sets. The tuning is performed using the training set, and the parameter configurations are evaluated on the testing set. This avoids the so-called overtuning, producing parameter configurations whose performance is the same when solving new/unseen instances. As a consequence, we can compare the algorithm (with the produced configuration) with other approaches from the literature, since the latter use a single configuration/algorithm to solve all instances. In contrast, if specific configurations are used for each instance (as presented in the paper), the tuning time should be accounted in the comparison.

Answer.
Yes, we did it very badly, contrary to our previous practice. In the new version of the work, a single parameter configuration was used for all instances. The configuration was obtained by performing the grid search method on a training random sample of 20 instances.

> Finally, Tables 3 - 5 present the results for the VNS and the best approach from the literature. It would be better to show the complete results, i.e. for all tested algorithms.

Answer.
Due to page limitations, we are unable to provide a more detailed view. We believe this is sufficient because we have picked out the best solution to compare it with. Note that due to the reduction in size to a poster format, we now have only one table that summarizes most of the previous information.

> Minor issues:
>
> 1. More detail about the applications of the k-domination problem (lines 100 - 103) can be given.
> 2. The discussion about the results (lines 499 - 512) are repetitive, since the observations are the same for the different values of k. It could be summarized.
> 3. Figures 3 and 4 can use filling patterns for the different values of k to ease the visualization in grayscale.
> 4. The running times reported in Figures 3 and 4 are averages over how many replications?
> 5. Instead of using the experimental results reported in the literature (for SG, PG and BS), why not running them again, given that the source code is available?

6. If space is an issue to accomodate any suggestion, Tables 3, 4 and 5 can be aggregated in a single table, Figures 3 and 4 can be aggregated in a single figure, and Figures 1 and 2 can be replaced by a single table, aggregated with Table 1.

Answers.

1. Unfortunately, we do not have enough space for a more detailed description.
2. We did it now in a more condensed form.
3. Unfortunately, due to space limitations on the poster, we had to remove all the figures. But the running times can now be found in Table 1.
4. Now, when running times are included in Table 1. We think it is clear that they are averages for ten runs, because there is a sentence that mentions this:
"PG and VNS algorithms are run ten times (using different random seeds) per each problem instance."
5. The source code (or executables) for BS was not available, so the results for small to medium instances were simply copied from the corresponding paper. For large instances, we ran the original algorithm PG and reported the results.
6. That's a good appetizer, thank you. We merged all the results into a single table (Table 1).

Comments from *Reviewer #3*

There is a fundamental issue with the tuning performed, which is done per instance without any intention of searching for parameter configurations that can generalize to unseen instances. Moreover, it seems the same effort was not applied to tune the other competing algorithms, which makes the comparison quite unfair. In particular, beam-search has a number of very fundamental parameters that would benefit from being tuned according to the instance size.

Answer.
You are absolutely right, it was unfortunate to fit on per instance basis. In the new version of the paper, we use a random (training) sample of instances to find a single combination of control parameters using grid search (instead of GA) – see Reviewer 1's comments.

For BS we did not have the source code (or binaries), so we could not tune it. Of course, the algorithm can be re-implemented, but we would like to leave that to future research, since our work was accepted in a form of a poster and not as a full paper.

> The approach makes even less sense in the case of parameters such as dmin and dmax that have a strong effect in constraining the set of solutions that are searched. Moreover, a wrong value of those parameters can effectively make impossible finding an infeasible solution. Thus tuning them only makes sense if one knows the actual optimal solution, which is not generally true. For unseen instances of a given size, one could calculate good values based on the properties of the graph.

Answer.
We agree, d_{min} and d_{max} have a strong effect on the search trajectory. Tuning on per instance basis was very bad idea.

> Another issue is that results on small and medium instances are taken from a different paper but it is unclear if those papers used a similar termination criterion on a similar powerful computer. Even assuming that for the small and medium instances the timelimit of 30 minutes is not reached and the VNS stops after the maximum number of 2000 iterations, it is unclear how those iterations are comparable with the termination criterion of SG, PG and BS.

Answer.
Reviewer 2 asked almost the same question (question 1), so we refer to the answer to that question first. Since VNS works fundamentally differently from the algorithms we compared, we had to find an exit criterion that was relatively fair. We established exit criteria that result in similar average running times. Also, the computer we used for testing was downgraded in the new version of the paper – Intel i5 instead of Intel i9. The comparison algorithms were run on Intel i7.

> The paper lists as a contribution that the parameters are tuned. But properly tuning parameters is just a step of proper experimental comparison and not a scientific contribution. It would similar to saying that a contribution of the paper is to do statistical tests to assess significance.

Answer.
You are right, we no longer mention parameter tuning as a contribution.

> The tables should report variance (or std. deviation) to show the variance of across multiple runs. Without variances, it cannot be said if the performance difference is sufficiently clear or a statistical test is needed to assess significance.

Answer.
Done, as suggested.

> The tables could also report running times, both for VNS and the algorithm of the literature (there is plenty of space in the paper).

Answer.
Done for the VNS. The running times of comparison algorithms are only partially reported in the literature. Also, the paper is not accepted as a full paper but as a poster with a limit of four pages. Therefore, we had to combine all the information into a single table and remove all the figures, remove the pseudo-code for local search, reduce the introduction, remove descriptions of the fast fitness assessment, etc.

> The paper claims that VNS is stopped after 30 minutes, however, figures 3 and 4 show runtimes longer than 1800 seconds

Answer.
This is explained with sentences: "The termination criteria of the VNS are: (i) the maximum running time of 30 minutes, and (ii) the maximum number of 20000 iterations. The time limit of 30 minutes is not checked during **initialization** (see Algorithm 1). This means that VNS can take more than 30 minutes to finish for some very large instances, such as Dublin and Boston."

> The stacked bars in Figs 3 and 4 are difficult to understand: For example, does a run for Manchester with k=4 take 5000 seconds or 5000 - 2500? Having the bars side by side will make more sense and avoid such questions.

Answer. We think it is more appropriate now when all relevant information are included in Table 1.

> "The time limit of 30 minutes is not checked during initialization [...] This means that VNS can take more than 30 minutes to finish" -¿ However, it also means that VNS does not execute and only the initial local search has an effect. So one may wonder how much of the work reported is done by the initial local search rather than the actual VNS algorithm.

Answer.
This is a correct observation. VNS does not even start in these situations. We make this clear in the paper with a sentence:
"More precisely, in these situations, after initialization, the most important steps of the VNS (shaking and subsequent local search) are not even performed due to an expired time limit."

Since this is a poster paper now, it is by definition a work in-progress, so some aspect may be incomplete or may change in the future.

Information on poster paper from https://gecco-2023.sigevo.org/Call-for-Papers:
"Such papers should still present work in genetic and evolutionary computation that

is original and new, but has not yet reached the stage of the more mature, complete research results that are typically published in full papers at GECCO. ”

> ”each processor outside the must have” -¿ the dominating set?

Answer.
Fixed.

> ”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. [. . .] Parameter tuning for large instances was too inefficient. Therefore, we used reasonable manually configured parameters for these instances” -¿ This approach is fundamentally wrong since it is overfitting the parameters to each problem instance. Which parameter settings should be used for a new unseen instance of the problem? The fact that it was not possible to extrapolate the parameters found in small and medium sized instances to larger instances clearly indicates that nothing was learned from this tuning effort due to being too specific and not general enough.
>
> This is why actual parameter tuning methods encourage a clear separation between training and testing instances. See:
>
> https://doi.org/10.1613/jair.2861
>
> https://doi.org/10.1016/j.orp.2016.09.002
>
> Moreover, the parameters could be tuned per groups of similar instance size and an extrapolation done for larger instances: Franco Mascia, Mauro Birattari, and Thomas Stützle. Tuning Algorithms for Tackling Large Instances: An Experimental Protocol. In P. M. Pardalos and G. Nicosia, editors, Learning and Intelligent Optimization, 7th International Conference, LION 7, volume 7997 of Lecture Notes in Computer Science, pp. 410–422. Springer, Heidelberg, 2013.

Answer.
You are right. As for tuning on groups of similar instance size, we now consider all small and medium sized instances as one group. This might be handled differently in the future version of the paper once it is fully mature. Larger instances are currently solved with the same parameters, but wee will certainly consider this idea of extrapolating parameters in future research.

> The datasets and codes could have been made available for review via the supplementary material or an anonymous Zenodo upload.

Answer.
Zenodo was not known to us, thank you for this suggestion. Now all materials are available at the public GitHub repo: `https://github.com/mikiMilan/k-domination`

Comments from *Reviewer #4*

> The paper proposes an efficient search metaheuristic for solving the k-domination problem.
>
> It describes the motivation and method clearly. The experiment description, including parameter tuning, is detailed and could thus be likely reproduced from the paper.
>
> The algorithmic components are motivated well (both in terms of efficiency and solution quality) and demonstrably improve over the chosen baselines empirically. While I feel that I cannot judge whether the baselines and benchmark are appropriate for the evaluation, the results are convincing to me. I therefore recommend accepting the paper.

Answer.
Thank you.

References

[Burke et al.(2014)] Edmund K Burke, Edmund K Burke, Graham Kendall, and Graham Kendall. 2014. *Search methodologies: introductory tutorials in optimization and decision support techniques.* Springer.

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

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

Variable neighborhood search for solving the k-domination problem

Milan Predojević
M.P.
milan.predojevic@pmf.unibl.org
Faculty of Sciences and Mathematics, University of Banja Luka
Banja Luka, Serb Republic, Bosnia and Herzegovina

Aleksandar Kartelj
A.K.
aleksandar.kartelj@matf.bg.ac.rs
Faculty of Mathematics, University of Belgrade
Belgrade, Serbia

Marko Djukanović
M.Dj.
marko.djukanovic@pmf.unibl.org
Faculty of Sciences and Mathematics, University of Banja Luka
Banja Luka, Serb Republic, Bosnia and Herzegovina

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 grid search approach. 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**.

KEYWORDS

Variable neighborhood search, graph domination, metaheuristics, combinatorial optimization

ACM Reference Format:
Milan Predojević, M.P., Aleksandar Kartelj, A.K., Marko Djukanović, and M.Dj.. 2023. Variable neighborhood search for solving the k-domination problem. In *Genetic and Evolutionary Computation Conference Companion (GECCO '23 Companion), July 15–19, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3583133.3590607

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
GECCO '23 Companion, July 15–19, 2023, Lisbon, Portugal
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0120-7/23/07.
https://doi.org/10.1145/3583133.3590607

1 INTRODUCTION

One of the best known classes of problems studied from theoretical, computational and practical points of view are domination problems on graphs [10]. The basic problem of this class is the *minimum dominating set problem*. The subset $D \subset V$ is called *dominating set* if each 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) [9]. This problem has many applications, for example, in biological networks [18], document summarization [22], 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 [23], an approach that uses the fundamental cut-sets of the graph [15], etc. Heuristic approaches are more common in the literature, e.g., a genetic algorithm [11], simulated annealing [12], ant colony optimization [14], to name a few. Several generalizations of MDSP, arising from practical experience, are proposed in the literature: the minimum weight dominating set problem [21], the minimum total dominating set problem [25], 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 [16]. 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. [8], and Gagarin and Corcoran [7]. 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 [24], where a k-dominating set represents a set of processors such that each processor outside this set must have at least k neighbors in the set.

We propose a variable neighborhood search (VNS) metaheuristic for solving MkDP. The main contribution of our work is that the proposed Vns significantly improves the state-of-the-art results for all considered problem instances, in case of $k \in \{1, 2, 4\}$.

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:

$$\arg\min_{D \subseteq V} |D| \text{ s.t. } \forall v \in V \setminus D, |N(v) \cap D| \geq k. \tag{1}$$

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 [17]. 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 [20], median problems [13], 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:

- `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}.

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 ← d_max_init
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
```

- `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 solution D, the following *fitness* function is used:

$$fitness(D) = (1 + viols(D)) \cdot (1 + penalty \cdot |D|) \tag{2}$$

with $viols(D) = \sum_{v \in V \setminus D} k - C(D, v)$, $C(D, v) = \min(k, |N(v) \cap D|)$.

It can be seen that $viols(D)$ 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 $viols(\cdot)$ 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.

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.

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 from the literature. The following four methods for MkDP are compared: (*i*) The standard greedy method from [8, 19], denoted by SG; (*ii*) Greedy method from [7], denoted by PG; (*iii*) Beam search approach from [4], denoted by BS; (*iv*) VNS approach, as described in Section 3 .

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 through 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]. Unlike the five large networks, these 20 instances are the same as in [4] – we obtained them directly from the authors.

Testing environment. Experiments were performed on a computer with Intel(R) Core(TM) i5-8265U CPU 1.80GHz and 16GB RAM, under Microsoft Windows 11 Pro OS. VNS is implemented in Python 3.9. 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 (acquired 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 it 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 20000 iterations. The time limit of 30 minutes is not checked during initialization (line 4 in Algorithm 1). It means that VNS can take more than 30 minutes to finish for some very large instances, such as `Dublin` and `Boston`. More precisely, in these situations, after initialization, the most important steps of VNS (shaking and subsequent local search) are not even performed due to an expired time limit.

[1] All instances, source codes of the VNS and grid search tuning algorithms, and detailed experimental results are publicly available on the GitHub page of this work: https://github.com/mikiMilan/k-domination.

The algorithms SG, PG and BS terminate their execution as soon as the first feasible solution is reached. Therefore, we decided to use termination criteria which allow similar running times to those reported by comparison algorithms [4]. Namely, the best results were obtained with the BS configuration BS4, where 4 stands for the beam width. For this algorithm, in the case of $k = 2$, the average running times in seconds are 1736, 8834, 1257, 7156 and 3327 for the Bath, Belfast, Brighton, Bristol and Cardiff instances, respectively. As stated in [4], the comparison algorithms were also implemented in Python and executed on a desktop computer with somewhat better CPU: Intel Core i7-8700 @3.20GHz.

Parameters tuning. As mentioned earlier, VNS for MkDPA involves four control parameters: $d_{min}, d_{max}, p_{move}$ and $penalty$. We chose a grid search method for tuning these parameters. As recommended in [17], we set the parameter d_{min} with the value 1. The possible values of parameters during tuning were: $d_{max} \in \{5, 10, 15, 20, 25, 30, 40, 50, 100\}$, $p_{move} \in \{0, 0.25, 0.5, 0.75, 1\}$, and $penalty \in \{0.005, 0.01, 0.015, 0.02\}$. The parameter space thus consists of 180 $(d_{max}, p_{move}, penalty)$ parameter configurations. From 60 small to medium sized instances (20 cities for $k \in \{1, 2, 4\}$), 20 instances were randomly selected for parameter tuning. For each of the 20 instances and each configuration, the VNS was run for 100 iterations. The best configuration, selected according to the best average rank over these 20 instances, is $(d_{max}, p_{move}, penalty) = (50, 0.5, 0.005)$.

4.1 Experimental results

Table 1 contains the overall results. The first column gives the name of the city road network. The next block contains network size information. The next three blocks give the VNS results and the best results from the literature so far, for $k = 1, 2, 4$, respectively. Each of these (three) blocks consists of 6 columns, where the first three columns show the results of VNS: average solution quality ($\overline{|D|}$), standard deviation ($\sigma(|D|)$) and average running time ($\bar{t}$). The other three columns report the results of the best approach from the literature: name of the approach (Alg.), average solution quality over ten runs ($\overline{|D|}$) and the corresponding standard deviation ($\sigma(|D|)$). Note that labels BS1, BS2 or BS4 correspond to the 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 $\approx$ 5–16% improvement rate over the second best, PG approach.
- The similar conclusions hold 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 164.2 over BS4 which achieved a score 195.2). For the large instances, VNS outperforms PG in all cases.

Table 1: Results

	Network size		Results for $k = 1$						Results for $k = 2$						Results for $k = 4$					
City	$\|V\|$	$\|E\|$	$\overline{\|D\|}$	$\sigma(\|D\|)$	$\overline{t}$	Alg.	$\overline{\|D\|}$	$\sigma(\|D\|)$	$\overline{\|D\|}$	$\sigma(\|D\|)$	$\overline{t}$	Alg.	$\overline{\|D\|}$	$\sigma(\|D\|)$	$\overline{\|D\|}$	$\sigma(\|D\|)$	$\overline{t}$	Alg.	$\overline{\|D\|}$	$\sigma(\|D\|)$
Bath	910	18560	**38**	0	661.7	BS4	44.6	0.9	**71.1**	0.3	720.7	BS1	89	1.4	**140.1**	0.7	644.8	BS4	160	1.1
Belfast	1700	62617	**39**	0	1800.1	BS4	50.2	1.5	**76.3**	0.5	1800.3	BS4	97.6	1	**148.3**	0.7	1800.1	BS4	179.6	2
Brighton	976	35012	**21**	0	1334	BS4	28.2	0.6	**40.1**	0.3	1789.7	BS4	49.4	0.5	**78**	0.5	1800.1	BS4	94.8	1.9
Bristol	1569	47522	**37**	0	1800.1	BS2	47.4	1	**73.8**	0.4	1800.1	BS4	94	1.4	**146.6**	1.1	1797.8	BS4	176.4	0.8
Cardiff	1127	23155	**39**	0	968.2	BS4	50.6	1	**78.3**	0.5	900.5	BS4	95.6	1.6	**157.5**	0.8	660.8	BS4	183.2	1.4
Coventry	1175	26689	**38**	0	1098.1	BS4	44.8	0.4	**73**	0	1002.5	BS4	85.1	0.7	**149.2**	0.9	827.3	BS4	172.6	1.4
Exeter	1250	31997	**38**	0	1365.8	BS4	50.6	0.5	**77**	0	1544.2	BS4	95.7	1	**158.1**	0.7	943.2	BS4	182.3	0.6
Glasgow	1137	24323	**50.1**	0.3	920.6	BS4	59.2	0.7	**94**	0.5	1068.4	BS4	110.6	1.7	**175.2**	0.9	745.6	BS4	199.8	1.6
Leeds	1647	56511	**40**	0	1800.1	BS4	52.4	0.8	**79.5**	0.5	1800.1	BS4	99.6	1	**152.8**	0.8	1800.1	BS4	187.1	0.7
Leicester	1531	48219	**38**	0	1800.1	BS4	51.5	0.5	**75**	0	1800.1	BS4	94.1	0.8	**149.3**	0.7	1759	BS4	177.7	1.8
Liverpool	1273	42564	**28**	0	1800.1	BS4	38.4	0.5	**57**	0.5	1800.1	BS4	72	0.8	**112.8**	0.6	1800.1	BS4	133	0.8
Manchester	1991	77286	**38.3**	0.7	1800.2	BS4	45.9	0.5	**77.9**	0.3	1800.1	BS4	91.5	0.9	**155.2**	0.6	1800.1	BS4	178.5	1
Newcastle	1109	26614	**44**	0	1146.4	BS4	52.6	1.1	**83.6**	0.5	1020.5	BS4	95.4	1.1	**152.4**	0.5	951.3	BS2	171.5	1.2
Nottingham	1739	51595	**44**	0	1799.1	BS4	56.6	0.8	**84.7**	0.5	1800.2	BS4	103.3	0.8	**164.2**	0.8	1800.2	BS4	195.2	1.2
Oxford	479	8396	**24**	0	263.1	BS4	27.9	0.5	**47**	0	298	BS4	54.9	0.7	**89**	0	254.4	BS2	100.8	0.9
Plymouth	1122	35070	**31**	0	1398.8	BS4	40.3	0.8	**61.3**	0.5	1694.2	BS4	75	1.1	**115.6**	0.5	1688	BS4	137	1.2
Sheffield	1582	50534	**42**	0	1800.2	BS4	52.5	0.7	**84.6**	0.5	1747.7	BS4	98.9	1.3	**161.4**	0.8	1800	BS4	182.2	1.2
Southampton	796	19942	**25**	0	750.1	BS4	29.6	0.8	**49.2**	0.4	807.2	BS4	61.1	0.7	**97.6**	0.5	1129	BS4	113.2	1.4
Sunderland	1346	42013	**36**	0	1559.3	BS4	46.3	0.4	**73**	0	1049	BS4	89.1	1.1	**141**	0.5	1438.3	BS4	163.6	1
York	1044	23774	**32**	0	856.8	BS4	39.1	0.3	**68**	0	573	BS4	77.6	0.6	**130.4**	0.5	784.5	BS4	145.8	1.2
Belgrade	19586	7561185	**86.5**	1.5	1805.9	PG	103.4	0.5	**171.1**	2.4	1803.8	PG	197.3	0.9	**341.9**	2.2	1802.3	SG	374.5	1.4
Berlin	29461	9944851	**102.1**	1.9	1817.9	PG	125.9	0.5	**204.9**	1.9	1878.7	PG	240.1	1.2	**396.4**	3.1	1804.8	PG	446.2	1.8
Boston	44797	28164740	**94.3**	1.9	2391.9	PG	102.7	1.3	**175.4**	2	2007	PG	191.6	0.9	**341**	0	3819.2	PG	368.7	1.5
Dublin	37982	21630466	**101.5**	1.1	1819.8	PG	113.8	1.2	**193.2**	4.8	1815.9	PG	211.3	2.7	**363**	0	3002.4	PG	390.2	2
Minsk	10487	1375618	**102.1**	1.1	1801.5	PG	126	0.9	**200**	1.9	1800.5	PG	240.4	1.4	**387.7**	3.5	1801	PG	457.6	2.4

5 CONCLUSIONS AND FUTURE WORK

In this paper we have solved the k-domination problem, a generalized version of the prominent minimum dominating set problem, with the variable neighborhood search (VNS) metaheuristic. The efficiency of VNS has been validated on the real-world benchmark, where it has been shown that VNS outperforms 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. VNS could be compared to exact approaches such as integer linear programming models, solved by general-purpose solvers such as Cplex or Gurobi.

ACKNOWLEDGMENTS

We thank Padraig Corcoran and Andrei Gagarin for providing problem instances and their algorithm implementations.
A. Kartelj was supported by grant 451-03-47/2023-01/200104 funded by Ministry of Science Technological Development and Innovations of the Republic of Serbia.

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 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] 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.
[8] 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.
[9] Fabrizio Grandoni. 2006. A note on the complexity of minimum dominating set. *Journal of Discrete Algorithms* 4, 2 (2006), 209–214.
[10] Teresa W Haynes, Stephen Hedetniemi, and Peter Slater. 2013. *Fundamentals of domination in graphs*. CRC press.
[11] 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.
[12] 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.
[13] 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.
[14] 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.
[15] Ali KARCİ. 2020. New algorithms for minimum dominating set in any graphs. *Computer Science* 5, 2 (2020), 62–70.
[16] 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.
[17] Nenad Mladenović and Pierre Hansen. 1997. Variable neighborhood search. *Computers & operations research* 24, 11 (1997), 1097–1100.
[18] Jose C Nacher and Tatsuya Akutsu. 2016. Minimum dominating set-based methods for analyzing biological networks. *Methods* 102 (2016), 57–63.
[19] Abhay K Parekh. 1991. Analysis of a greedy heuristic for finding small dominating sets in graphs. *Information processing letters* 39, 5 (1991), 237–240.
[20] 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.
[21] 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.
[22] 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.
[23] Johan MM Van Rooij and Hans L Bodlaender. 2011. Exact algorithms for dominating set. *Discrete Applied Mathematics* 159, 17 (2011), 2147–2164.
[24] 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.
[25] 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.