Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Systematic Exploration of Larger Local Search Neighborhoods for the Minimum Vertex Cover Problem

Maximilian Katzmann, Christian Komusiewicz
Institut für Informatik
Friedrich-Schiller-Universität Jena, Jena, Germany
{maximilian.katzmann, christian.komusiewicz}@uni-jena.de

Abstract

We investigate the potential of exhaustively exploring larger neighborhoods in local search algorithms for MINIMUM VERTEX COVER. More precisely, we study whether, for moderate values of k, it is feasible and worthwhile to determine, given a graph G with vertex cover C, if there is a k-swap S such that $(C \setminus S) \cup (S \setminus C)$ is a smaller vertex cover of G. First, we describe an algorithm running in $\Delta^{O(k)} \cdot n$ time for searching the k-swap neighborhood on n-vertex graphs with maximum degree Δ. Then, we demonstrate that, by devising additional pruning rules that decrease the size of the search space, this algorithm can be implemented so that it solves the problem quickly for $k \approx 20$. Finally, we show that it is worthwhile to consider moderately-sized k-swap neighborhoods. For our benchmark data set, we show that when combining our algorithm with a hill-climbing approach, the solution quality improves quickly with the radius k of the local search neighborhood and that in most cases optimal solutions can be found by setting $k = 21$.

Introduction

MINIMUM VERTEX COVER and its sister problem MAXIMUM INDEPENDENT SET are fundamental NP-hard optimization problems, their decision versions being among Karp's classic 21 NP-complete problems (Karp 1972). In MINIMUM VERTEX COVER we aim to find a smallest vertex set covering all edges of a graph.

MINIMUM VERTEX COVER

Input: An undirected graph $G = (V, E)$.
Task: Find a minimum-cardinality set $C \subseteq V$ such that $\forall \{u, v\} \in E\colon u \in C \vee v \in C$.

Equivalently, there should be no edges among vertices in $V \setminus C$, that is, $V \setminus C$ should be an *independent set* of maximum size. Both problems are also hard from the viewpoint of polynomial-time approximation algorithms: MINIMUM VERTEX COVER is NP-hard to approximate within a factor of 1.36 (Dinur and Safra 2005) and there is evidence that even improving upon the trivial factor-2 approximation might be impossible (Khot and Regev 2008). For MAXIMUM INDEPENDENT SET it is even NP-hard to obtain a factor-$|V|^{1-\epsilon}$ approximation in polynomial time for any $\epsilon > 0$ (Zuckerman 2007).

Despite these hardness results for both problems, heuristic algorithms have a very good performance in practice. One approach that has been applied successfully is local search: state-of-the art heuristic solvers for MINIMUM VERTEX COVER such as EWCC (Cai, Su, and Sattar 2011), NuMVC (Cai et al. 2013), and FastVC (Cai 2015) are based on local search strategies. The main focus in this line of research has been to investigate heuristic approaches that prevent cycling during the local search algorithm and avoid being stuck in a bad local optimum.

Here, we study the performance of local search in a strict hill-climbing setting. In hill-climbing, cycling is not an issue but it is crucial to avoid bad local optima. To this end, we exhaustively examine larger neighborhoods of the current solution. More precisely, we follow the approach of *parameterized local search* (Fellows 2001; Fellows et al. 2012; Gaspers et al. 2012), where the radius k of the neighborhood is used as a parameter of the algorithm and the aim is to obtain fast algorithms for searching the k-neighborhood of the solution. Here, when searching for an improvement of a current vertex cover C, we consider the *k-swap neighborhood* which precisely contains all vertex sets that can be obtained from C by adding and removing a set S (the *swap*) of altogether at most k vertices. In other words, we aim to solve the following problem.

LS-VERTEX COVER

Input: An undirected graph $G = (V, E)$, a vertex cover C of G, and an integer k.
Task: Find a swap S of size at most k, that is, a vertex set $S \subseteq V$ such that $C \oplus S := (C \setminus S) \cup (S \setminus C)$ is a vertex cover, $|C \oplus S| < |C|$, and $|S| \leq k$.

Solving LS-VERTEX COVER for larger k is commonly assumed to be infeasible: The straightforward algorithm trying all possible k-swaps runs in $n^{\Theta(k)}$ time. Moreover, there is complexity-theoretic evidence that a (much more desirable) running time of $f(k) \cdot n^{O(1)}$ where f is some computable function depending only on k cannot be achieved (Fellows et al. 2012). Due to this assumed hardness of the problem, implementations of hill-climbing local search use small k-swap neighborhoods such as 3-swap and 5-swap neighborhoods (Andrade, Resende, and Werneck 2012).

Here we show that by using the maximum degree of the input graph as a secondary parameter, we can in fact examine larger neighborhoods and that exhaustive exploration of these neighborhoods is worthwhile. More precisely, we present an algorithm that solves LS-VERTEX COVER in $\mathcal{O}(2^k(\Delta-1)^{k/2}\cdot k/2\cdot n)$ time, where Δ is the maximum degree of the input graph.[1] The algorithm is based on a search tree procedure which enumerates connected subgraphs of G. We then provide pruning rules that help to avoid searching parts of the k-swap neighborhoods that contain no solution. We then evaluate the performance of the algorithm and the general usefulness of larger swap-neighborhoods. In a nutshell, we show that we can efficiently search k-swap neighborhoods for $k \approx 20$, even on large instances. We also demonstrate the usefulness of the pruning rules concerning the running time of the algorithm and the reduction of the search space size. Moreover, we show that the solution quality improves quickly with growing k. Finally, we show that usually it should not be necessary to set k too large in order to find optimal or almost-optimal solutions: in 91 % of our benchmark instances, the optimal solution can be found by setting $k = 21$.

Preliminaries. We consider simple undirected graphs $G = (V, E)$. For a vertex $v \in V$ we denote its *neighborhood* by $N(v) := \{u \mid \{u, v\} \in E\}$. We use $G[S] := (S, \{\{u, v\} \in E \mid u, v \in S\})$ to denote the *subgraph of G induced by S*. We call a vertex set S *connected* if $G[S]$ is a connected graph. A bipartite graph with partite sets B and W is defined as $G = (B \uplus W, E)$. We use the following notation for *stacks*: Given a set X, the procedure $\mathrm{stack}(X)$ builds a stack containing the elements of X in arbitrary order, $\mathrm{pop}(X)$ removes the top element of X and returns it. For two stacks S and S', we let $S \circ S'$ denote the stack containing the elements of S in the order of S on top and then the elements of S' in the order of S'. Due to lack of space, some proofs are deferred to a full version of the article.

On the Structure of k-Swaps

Before describing our algorithm, we make several observations restricting the type of swaps $S \subseteq V$ that we need to consider when trying to improve the vertex cover C. Essentially, we show that it is sufficient to consider connected swaps where $|S \cap C|$, the number of vertices that are removed from the cover, exceeds $|S \setminus C|$, the number of vertices that are added to the cover, by exactly one. The latter is obviously true when we do not restrict ourselves to connected swaps. To show this for connected swaps, we make use of the following lemma.

Lemma 1. *Let $G = (B \uplus W, E)$ be a bipartite, undirected and connected graph with partite sets B and W where $|B| > |W|$. Then there is a vertex set $B' \subseteq B$, such that $|B'| = |W| + 1$ and $B' \cup W$ is connected.*

[1] Fellows et al. (Fellows et al. 2012) show that LS-VERTEX COVER is tractable on the more general class of graphs with bounded local treewidth. The running time bound in terms of Δ and k is, however, not given explicitly.

The following lemma now shows that we can focus on connected swaps with a size difference of one on the sides. It is an adaption of (Fellows et al. 2012, Lemma 1), which is stated for a variant of DOMINATING SET.

Lemma 2. *Let C_1, C_2 be vertex covers of a graph $G = (V, E)$ such that $|C_1 \oplus C_2| \leq k$ and $|C_1| > |C_2|$. Then there are sets $S_1, S_2 \subseteq V$ such that*

1. *$C = (C_1 \setminus S_1) \cup S_2$ is a vertex cover of G,*
2. *$|C| = |C_1| - 1$,*
3. *$|S_1 \cup S_2| \leq k$ and*
4. *$S_1 \cup S_2$ is connected.*

We exploit one further observation in the search tree algorithm: if we move a vertex from the vertex cover to the independent set, then all its neighbors in the independent set must be moved to the vertex cover.

Observation 1. *Let C be a vertex cover of G and let S be a swap of C such that $C \oplus S$ is a vertex cover, and let $v \in S \cap C$. Then, $(N(v) \setminus C) \subseteq S$.*

A Search Tree Algorithm

Lemma 2 forms the basis of our local search algorithm. It states that we may restrict ourselves to determining whether there exists what we call a *valid* swap: A vertex set $S = (B \uplus W)$ is a *valid* k-swap for a vertex cover C in G, if S is connected, $|B| = \lceil k/2 \rceil$, $|W| = \lfloor k/2 \rfloor$ and $(C \setminus B) \cup W$ is a vertex cover in G.

The local search algorithm performs hill-climbing in the k-swap neighborhood: it gets as input a graph G with a vertex cover C and a number $k_{\max}$, and tries to improve C by searching for a valid k-swap as long as possible, that is, until it encounters a solution that has no valid $k_{\max}$-swap which it then outputs. We call such a solution *k_{max}-optimal*. The pseudo-code of the main routine is given in Algorithm 1. The algorithm checks for each odd $k \leq k_{\max}$ whether there is a valid k-swap. If this is the case for some k, then the swap is applied and the search continues. Otherwise, C is $k_{\max}$-optimal and the algorithm returns C. The main algorithmic part is how to determine whether such a k-swap exists, that is, to solve LS-VERTEX COVER. To solve this problem, the

Algorithm 1 Local Search

Output: k-optimal vertex cover C

```
1:  procedure LocalSearch(G, C, k_max)
2:      for k ∈ {1, 3, 5, ..., k_max} do
3:          for v ∈ C do
4:              S ← {v} ∪ (N(v) \ C)
5:              P ← stack(N(v) \ C)
6:              p ← pop(P)
7:              F ← N(v) ∩ C
8:              S ← Enum(S, p, P, F)
9:              if S ≠ ∅ then
10:                 C ← C ⊕ S
11:                 go to 2
12:     output C
```

Algorithm 2 Subgraph Enumeration

Require: Graph G with vertex cover C and $k \in \mathbb{N}$
Output: Valid k-swap S

```
1: procedure ENUM(S, p, P, F)
2:     if (S ∩ C) is not independent then output ∅
3:     if |S ∩ C| > ⌈k/2⌉ then output ∅
4:     if |S \ C| > ⌊k/2⌋ then output ∅
5:     if |S| = k then output S
6:     for b ∈ N(p) \ (S ∪ F) do
7:         if N(b) ∩ (F \ C) = ∅ then
8:             N_b ← N(b) \ (S ∪ C)
9:             P' = stack(N_b) ∘ stack({p}) ∘ P,
10:            p' ← pop(P')
11:            ENUM(S ∪ {b} ∪ N_b, p', P', F)
12:        F ← F ∪ {b}
13:    p' ← pop(P)
14:    ENUM(S, p', P, F)
```

algorithm determines for each vertex v of the vertex cover C whether there is a valid k-swap containing v.

This is done by calls to the ENUM procedure whose pseudocode is shown in Algorithm 2. ENUM gets as input a connected vertex set S, a *pivot* vertex p, a stack P containing vertices of S that are not in C and a vertex set F. The procedure determines whether there is a valid k-swap S' with the following properties:

- S' *extends* S, that is, $S \subseteq S'$, and
- S' *avoids* F, that is, $S' \cap F = \emptyset$.

If such a swap exists, then ENUM outputs the first one that it finds. To simplify the description of the algorithm, we call vertices that are in the current vertex cover *black* and those that are in the independent set *white*. We use the pivot vertex p and the stack P to organize the search for these sets. More precisely, the pivot vertex p is the vertex from which currently neighbors may be added to extend S. The stack P contains those white vertices of S of which a neighbor $v \in S' \setminus S$ may still be added to S. The order of the stack determines the order in which the algorithm makes these vertices a pivot vertex.

We next show correctness of ENUM. In the proof the stack P can be thought of as a set; we need the stack ordering to show correctness of one of our pruning rules. The statement of Lemma 3 describes the situation at a call of ENUM, it is illustrated in Figure 1.

Lemma 3. *Let G be a graph with vertex cover C and connected set S, let F be a vertex set in G, and let $k \in \mathbb{N}$ be an odd number. Moreover, assume that*

- *every neighbor of each white vertex of $S \setminus (P \cup \{p\})$ is either contained in S or in F, and*
- $N(S \cap C) \subseteq C \cup S$.

Then, ENUM(S, p, P, F) *outputs a valid k-swap S' extending S and avoiding F if and only if such a k-swap exists.*

Proof. If $S \cap C$ is not an independent set, then there is no valid swap extending S and the algorithm correctly aborts without output in Line 2.

Next, if S contains more than $\lceil k/2 \rceil$ black vertices, then there is no valid k-swap S' extending S since any valid k-swap contains $\lceil k/2 \rceil$ black vertices. Hence, the algorithm correctly aborts without output in Line 3.

Similarly, if S contains more than $\lfloor k/2 \rfloor$ white vertices, then there is no valid k-swap S' extending S since valid k-swaps have exactly $\lfloor k/2 \rfloor$ white vertices. Hence, the algorithm correctly aborts without output in Line 4.

Now if $|S| = k$ in Line 5, then S is a valid k-swap: The set S contains at most $\lceil k/2 \rceil$ black vertices and at most $\lfloor k/2 \rfloor$ white vertices and thus it contains exactly $\lceil k/2 \rceil$ black and $\lfloor k/2 \rfloor$ white vertices. Thus, $|C \oplus S| < |C|$ and it remains to show that $C' := C \oplus S$ is a vertex cover. Since C is a vertex cover, it holds that if there is an edge not covered by C', then this edge has at least one endpoint in $S \cap C$, that is, in a black vertex of S. Since $S \cap C$ is an independent set, by the check in Line 2, it must hold furthermore, that the other endpoint of this edge is a white vertex. Since white vertices in S belong to C', we thus have that this vertex is contained in $V \setminus (C \cup S)$. This contradicts the assumption that $N(S \cap C) \subseteq (C \cup S)$. Hence, C' is a vertex cover and S is a valid k-swap fulfilling the condition of the lemma.

This already shows the "only if" part of the statement. Moreover, it shows that if $|S| = k$, then the output of the algorithm is correct.

Thus, it remains to show that, if $|S| < k$ and there exists a valid k-swap S' fulfilling the condition of the lemma, then the algorithm outputs a valid k-swap in one of the recursive calls made in Lines 11 or 14. We show this by induction on $(k - |S|)$. By the arguments above, the claim holds for $k - |S|$. Now consider any S with $|S| < k$ and assume that, by the inductive hypothesis, the algorithm is correct for all S^* with $|S^*| > |S|$.

Assume first that S' contains a vertex $b \in N(p) \setminus S$. Since S' avoids F, b is considered by the for-loop of Line 6. Without loss of generality, let b be the first vertex of $(S' \cup N(p)) \setminus S$ that is considered by this for-loop. If the algorithm outputs any swap before the for-loop considers b, then the claim holds. Otherwise, the only vertices that are added to F when the for-loop reaches b belong to $N(p) \setminus S$.

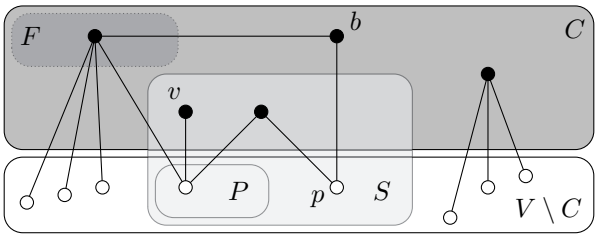

Figure 1: A situation of the enumeration process in a graph G with vertex cover C. The vertex $v \in C$ is the first vertex added to S, p is the current white pivot vertex, P is the remaining pivot stack. Adding the black vertex b to the swap S makes it a valid 5-swap. Black vertices that were not added to S are in the set F of forbidden vertices.

Thus, S' still avoids the set F when the for-loop considers b. Moreover, b has no neighbor in $F \setminus C$: by Observation 1 any neighbor of b in $V \setminus C$ must belong to S'.

Hence, b passes the test in Line 7. The algorithm now recursively calls itself. In the recursive call, the algorithm returns a valid k-swap extending $S \cup \{b\} \cup N_b$, avoiding F, and adding neighbors of vertices in P' where P' contains P, N_b, and p. Here, N_b is the set of white neighbors of b which are not yet contained in S. By Observation 1, any swap containing b must contain N_b. Thus, since S' extends $S \cup \{b\}$, it also extends $\tilde{S} = S \cup \{b\} \cup N_b$ and all vertices of $\tilde{S} \cap C$ have only neighbors in $C \cup S$. Moreover, as noted above S' avoids F. Finally, every vertex of $S \setminus (P \cup \{p\})$ is also contained in $\tilde{S} \setminus (P' \cup \{p\})$. Hence, for these vertices all neighbors are still either in $\tilde{S}$ or in F.

Altogether, this means that the parameters for the recursive call fulfill the claims of the lemma and thus we can assume by induction that the call to ENUM outputs some valid k-swap extending S and avoiding F.

The final case to consider is that no vertex of $N(p) \setminus (S \cup F)$ is contained in S', that is, S' avoids $N(p) \setminus (S \cup F)$. To show the correctness of this case, we need a second induction, this time on $|P|$. Observe that if $|P| = 0$ and $|S| < k$, then S' cannot avoid $N(p) \setminus (S \cup F)$, as we still need to add vertices to S and may only add neighbors of p. Thus, if $|P| = 0$, then the previous case applies and the algorithm is correct. Now if $|P| > 0$, then assume by induction that the algorithm is correct for all calls ENUM$(\tilde{S}, \tilde{p}, \tilde{P}, \tilde{F})$ where $|\tilde{S}| = |S|$ and $|\tilde{P}| < |P|$.

In the recursive call in Line 14, the algorithm reports a solution extending S and avoiding the modified F, which now also contains all vertices of $N(p) \setminus (S \cup F)$. By the case assumption, the solution S' fulfills these conditions. Moreover, for each white vertex of $S \setminus (P \cup \{p'\})$, all black neighbors are either contained in S or in F: by the assumption of the lemma this holds for all vertices except p, and for p this holds by the case assumption. Thus, the lemma conditions are fulfilled by S' and, by the inductive hypothesis that the algorithm is correct for calls with a smaller pivot stack, the call outputs a valid k-swap fulfilling the conditions of the lemma. □

To show correctness of LOCALSEARCH it only remains to note that the parameters of some call to ENUM are correct: If C is not $k_{\max}$-optimal, then for some odd $k \leq k_{\max}$, there exists a valid k-swap S' which contains at least one vertex $v \in C$. Consider the pass of the for-loop in LOCALSEARCH for this v. By Observation 1, S' contains all white neighbors of v. Thus, it extends $S := \{v\} \cup (N(v) \setminus C)$. Moreover, S' trivially avoids $F := \emptyset$. Finally, every vertex of S' that has some neighbor in S is adjacent to some vertex of $N(v) \setminus C := P \cup \{p\}$. Thus, the call ENUM$(S, p, P, F)$ fulfills the conditions of Lemma 3 and therefore some valid k-swap is output.

We now bound the running time of the procedure.

Lemma 4. *Each call to* ENUM *in Algorithm 1 can be performed in $\mathcal{O}(2^k(\Delta-1)^{k/2})$ time.*

Altogether, we arrive at the following.

Theorem 5. *Let $G = (V, E)$ be a graph with $|V| = n$ and let $C \subseteq V$ a vertex cover of G. In $\mathcal{O}(2^k(\Delta-1)^{k/2} \cdot k \cdot n)$ time, we either find that C is k-optimal or compute a vertex cover C' with $|C'| = |C| - 1$. In $\mathcal{O}(2^k(\Delta-1)^{k/2} \cdot k \cdot n^2)$ time, we can compute a k-optimal solution*

Pruning Rules

To speed up the algorithm, we develop pruning rules that identify search tree nodes that do not yield a valid swap. For example, if a swap S contains $\lfloor k/2 \rfloor$ white vertices, we cannot add further white vertices to it. Therefore, we consider no black vertices with at least one white neighbor outside of S, since adding them to S requires adding further white vertices.

We implemented two further more involved pruning rules. The first rule discards connected vertex sets S that contain a suboptimal part. Before stating the rule, we need some terminology concerning the structure of the set S. To this end, we say that ENUM *implicitly defines* a rooted spanning tree $\mathcal{T}$ of S where

- the root of $\mathcal{T}$ is the black vertex v that is added in the initial call to ENUM,
- a black vertex b is the child of a white vertex p if p is the current pivot vertex in the search tree node that creates the call to ENUM in which b is added,
- a white vertex w is the child of a black vertex b if $w \in N_b$ when b is added.

We say that a white vertex $p \in S$ *retires* when it is not part of the pivot stack and not a pivot vertex. In Algorithm 2, this happens in Line 14, when ENUM is called with pivot p' and pivot stack P which does not contain p.

Pruning Rule 1. *Let p be a retired vertex and let $\mathcal{T}_p$ denote the subtree of $\mathcal{T}$ rooted at p. If $\mathcal{T}_p$ contains at least as many white vertices as black vertices, then discard the current node of the search tree.*

Proof of correctness. Let S' be a valid k-swap extending the set S at the search tree node in which Pruning Rule 1 is applied. We show that then there is a valid swap of size less than k. Since in LOCALSEARCH we call ENUM for increasing values of k, it is known that no valid swap of size less than k exists which implies that S' cannot exist. Thus, assume towards a contradiction, that S' exists. Let $\mathcal{T}'$ be the rooted spanning tree of S' implied by the search tree algorithm at the search tree node that returns S'.

Now, consider the swap $\tilde{S}$ obtained from S' by removing all vertices of $\mathcal{T}_p$ and let $\tilde{C} := C \oplus \tilde{S}$ be the vertex set obtained by applying $\tilde{S}$ to C. Since $\mathcal{T}_p$ has at least as many black vertices as white vertices, we have $|\tilde{C}| < |C|$. Moreover, we show that $\tilde{C}$ is a vertex cover. This implies that there is a valid swap of size less than k leading to the desired contradiction.

Thus, it remains to show that $\tilde{C}$ is a vertex cover. Any uncovered edge e has both endpoints in $V \setminus \tilde{C}$. Since S' is a valid swap, e has one endpoint in $\mathcal{T}_p$. Since black vertices of $\mathcal{T}_p$ are contained in $\tilde{C}$ (they are in C and not contained

in the swap $\tilde{S}$) one endpoint of e is a white vertex w of $\mathcal{T}_p$. Moreover, the other endpoint of e is a black vertex b from $\tilde{S}$: the white vertices form an independent set and black vertices that are not in $\tilde{S}$ are contained in $\tilde{C}$. We show by a case distinction that such an edge $e = \{w, b\}$ does not exist.

Case 1: The black vertex b was added to S before w. In this case, the vertex w is a white neighbor of b that is not contained in the current set S when b is added. Thus, w is a child of b in $\mathcal{T}$. Consequently, since w is contained in $\mathcal{T}_p$, so is b. This contradicts the assumption that b is not contained in $\mathcal{T}_p$.

Case 2: The black vertex b was added to S after w. When w retires, all neighbors of w are either in S or in F. Since b is not in $\mathcal{T}_p$, it is not a child of w and thus not in S when w retires. Consequently, b is contained in F. This contradicts the fact that b is added to S after w retires. □

The next pruning rule applies when we may only add black vertices since S contains $\lfloor k/2 \rfloor$ white vertices. We compute an upper bound on the size of the maximum independent set formed by the remaining black vertices. The upper bound relies on the following definition. Let $G = (V, E)$ be an undirected graph. A *lower-bound-packing* is a subgraph $L = (V, E' \subseteq E)$ such that for every vertex $v \in V$ one of three conditions holds: Either 1) v is isolated or 2) v is part of an isolated edge or 3) v is part of an isolated triangle. The *size* of a lower-bound-packing is equal to the number of its components and it is an upper bound on the size of the maximum independent set in G. Thus, we may apply a pruning rule if the graph induced by the black candidate vertices has a small packing.

Pruning Rule 2. *If S contains $\lfloor k/2 \rfloor$ white vertices, and $G[N(S \setminus C) \setminus S]$ has a lower-bound-packing of size less than $|S| - k$, then discard the current node of the search tree algorithm.*

Computational Experiments

We implemented our algorithm in C++ using *NGraph* (http://math.nist.gov/∼RPozo/ngraph/) as underlying data structure. Our code is available at http://github.com/maxkatzmann/kLSVC. The experiments were run on a machine with 32 Intel Xeon 2.60GHz CPUs and 128GB RAM running Debian GNU/Linux 7. We considered real-world networks obtained from KONECT (Kunegis 2013), the 10th DIMACS challenge (Bader et al. 2014), and the Network Data Repository (Rossi and Ahmed 2015). We group the real-world networks into two categories, consisting of 16 small graphs (up to 2 000 vertices) and 19 large graphs (between 2 000 and 550 000 vertices); the instance names and some instance properties are shown in Table 1. The results for the small and large graphs are somewhat similar, thus we present only the results for the large graphs. In the plots, the boxes extend from 25th to 75th percentile with the middle line showing the median, whiskers extend to the 10th or 90th percentile.

We call a vertex cover *greedy* if it is obtained by adding a vertex with maximum degree until all edges are covered; the set V is the *trivial* vertex cover. Optimal vertex covers were computed with Gurobi 6.5.

Table 1: Networks used for our experiments.

Size	Name	n	m
Small	adjnoun_adjacency	112	425
	arenas-email	1 133	5 451
	arenas-jazz	198	2 742
	bio-celegans	453	2 025
	bio-diseasome	516	1 188
	bio-yeast	1 458	1 948
	ca-netscience	379	914
	ca-sandi-auths	86	124
	contiguous-usa	49	107
	dolphins	62	159
	inf-euroroad	1 174	1 417
	inf-USAir97	332	2 126
	moreno-zebra	27	111
	soc-firm-hi-tech	33	91
	soc-wiki-Vote	889	2 914
	ucidata-zachary	34	78
Large	bio-dmela	7 393	25 569
	ca-AstroPh	17 903	196 972
	ca-CSphd	1 882	1 740
	ca-CondMat	21 363	91 286
	ca-Erdos992	6 100	7 515
	ca-GrQc	4 158	13 422
	ca-HepPh	11 204	117 619
	citationCiteseer	268 495	1 156 647
	coAuthorsCiteseer	227 320	814 134
	coAuthorsDBLP	299 067	977 676
	coPapersCiteseer	434 102	16 036 720
	coPapersDBLP	540 486	15 245 729
	inf-openflights	2 939	15 677
	inf-power	4 941	6 594
	soc-advogato	6 541	51 127
	soc-anybeat	12 645	49 132
	soc-brightkite	56 739	212 945
	soc-hamsterster	2 426	16 630
	soc-twitter-follows	404 719	713 319

Running time and search space size. First we consider the running time of our algorithm when searching neighborhoods of increasing size. Figure 2 shows the running time to find k-optimal solutions on large graphs when starting from a greedy vertex cover. As expected, the running time increases with k. One can also see how applying our pruning rules reduces the running time for larger values of k.

We also considered the size of the search space, counted in the number of search tree vertices normalized by n. As expected, the search space size grows exponentially with k as shown in Figure 3. The pruning rules reduce the size of the search space approximately five-fold, which in turn explains the reduced running time.

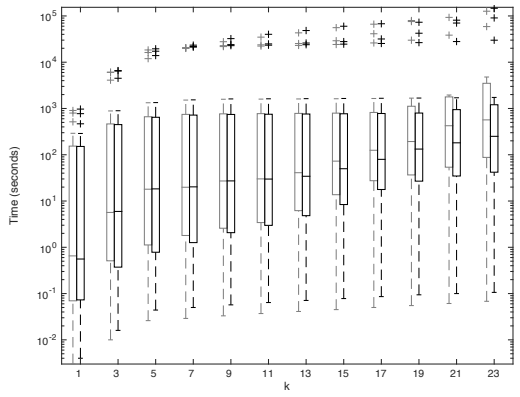

Figure 2: The total time needed to find k-optimal solutions for increasing values of k in large graphs. Grey boxes (left) represent the default algorithm, black boxes (right) represent the algorithm with pruning rules.

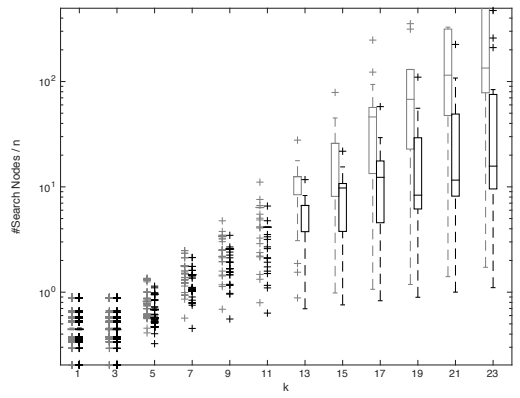

Figure 3: The size of the search space on large graphs. For each value of k the gray boxes (left) represent complete size of the search space, while the black boxes (right) show the pruned search space.

Sufficient k to reach (near)-optimal solutions. Due to the reduced size of the search space we are able to explore neighborhoods of moderate sizes, allowing us to find out which neighborhood-radius is sufficient to find optimal vertex covers by starting from a greedy or a trivial vertex cover.

Figure 4 compares the size of the optimal vertex cover to the value for k to find it. The figure includes all instances except the ones from the DIMACS challenge which are discussed below. The value of k is very small compared to the size of the optimal cover and usually smaller when starting from a greedy vertex cover than when starting from the trivial one. The fact that small neighborhoods are sufficient to find the optimum could explain the success of local search algorithms for MINIMUM VERTEX COVER.

Table 2 supports this observation: for four of the remaining five large instances, the value k that is sufficient to find the optimum starting from a greedy vertex cover is very small. For the remaining instance (the 'citationCiteseer' graph from the DIMACS challenge), we found a solution with 118 117 vertices, only two more than the optimal solution, by setting $k = 23$.

Table 2: Sufficient values for k to find the optimal vertex cover in real world networks from the 10th DIMACS challenge (Bader et al. 2014).

Name	n	m	Opt.	k
coAuthorsCiteseer	227 320	814 134	129 193	7
coAuthorsDBLP	299 067	977 676	155 618	9
coPapersCiteseer	434 102	16 036 720	386 106	9
coPapersDBLP	540 486	15 245 729	472 179	9

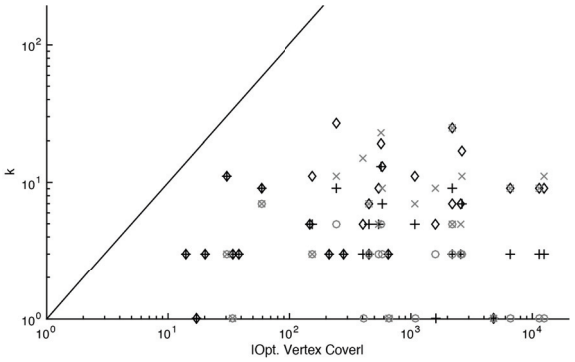

Figure 4: Size of the optimal vertex cover compared to the sufficient value for k to find it. Grey and black symbols indicate starting at greedy or trivial vertex covers, respectively; crosses show the k-value to reach covers of size $1.01 \cdot$OPT, diamonds and circles show the k-value that is sufficient to find the optimum.

Finally, we examine the amount of improvement when starting from a greedy vertex cover. As Figure 5 shows, the solution quality improves quickly with k, with the majority of the improvements obtained for $k \approx 9$. For $k \geq 9$, the median improvement in solution size is approximately 5%.

Combination with a state-of-the-art local search algorithm. For our test data, FastVC (Cai 2015) finds an optimal solution for many instances within one hour. Consequently, it outperforms our algorithm in terms of running time. For some instances, it is known, however, that FastVC does not find an optimal solution (Cai 2015). Thus, we would like to determine whether it is worthwhile to use our algorithm as a post-processing of FastVC. The following preliminary experiments hint that this might be the case. We started FastVC with a time limit of 5 minutes and seed '123456' on three hard instances: 'soc-delicious' and 'tech-RL-caida' from the Network Repository, and 'citationCiteseer' from the DIMACS challenge. We found that during the last 4 minutes of its execution FastVC finds no further improvements. In comparison, when we take the solution found by FastVC after 1 minute and let our algorithm run for 4 minutes, then it improves the vertex cover size by 4, 25, and 9, respectively.

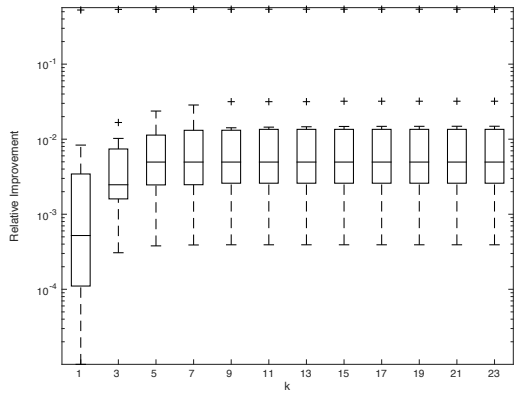

Figure 5: Relative solution improvement over the size of the greedy cover for large graphs and different neighborhood radii k.

Conclusion

We demonstrated the potential of considering larger local search neighborhoods for MINIMUM VERTEX COVER. Future work could focus on different hard computational problems such as MINIMUM DOMINATING SET or MAXIMUM CLIQUE. For MAXIMUM CLIQUE, efficient local search algorithms are known (Wang, Cai, and Yin 2016) and thus parameterized local search might prove useful for this problem as well.

Our experiments show that the sufficient neighborhood radius for finding optimal solutions depends on the initial vertex cover. This dependency should be examined systematically. For example, when starting from random vertex covers, what is the relation between number of initial vertex covers and sufficient neighborhood radius?

In addition, further more extensive experiments with other local search solvers such as FastVC (Cai 2015) should be made. In particular, the potential for using our algorithm as a postprocessing or in a hybrid algorithm that switches between hill-climbing with larger k and stochastic local search should investigated. Our preliminary experiments with FastVC show that this might be an interesting direction for the future.

Finally, we considered *strict* local search, as our algorithm only finds swaps that are inside the k-swap neighborhood of the current solution. One could also use *permissive* local search where the algorithm may return swaps that are outside the k-swap neighborhood. Theoretical evidence suggests that permissive local search is easier (Gaspers et al. 2012); it would be interesting to also provide computational evidence for this claim.

Acknowledgments. Christian Komusiewicz gratefully acknowledges the support by the DFG, project MAGZ, KO 3669 / 4-1.

References

Andrade, D. V.; Resende, M. G. C.; and Werneck, R. F. F. 2012. Fast local search for the maximum independent set problem. *J. Heuristics* 18(4):525–547.

Bader, D. A.; Kappes, A.; Meyerhenke, H.; Sanders, P.; Schulz, C.; and Wagner, D. 2014. Benchmarking for graph clustering and partitioning. In *Encyclopedia of Social Network Analysis and Mining*. Springer. 73–82.

Cai, S.; Su, K.; Luo, C.; and Sattar, A. 2013. NuMVC: an efficient local search algorithm for minimum vertex cover. *J. Artif. Intell. Res.* 46:687–716.

Cai, S.; Su, K.; and Sattar, A. 2011. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artif. Intell.* 175(9-10):1672–1696.

Cai, S. 2015. Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI '15)*, 747–753. AAAI Press.

Dinur, I., and Safra, S. 2005. On the hardness of approximating minimum vertex cover. *Ann. Math.* 162(1):439–485.

Fellows, M. R.; Fomin, F. V.; Lokshtanov, D.; Rosamond, F. A.; Saurabh, S.; and Villanger, Y. 2012. Local search: Is brute-force avoidable? *J. Comput. Syst. Sci.* 78(3):707–719.

Fellows, M. R. 2001. Parameterized complexity: The main ideas and some research frontiers. In *Proceedings of the 12th International Symposium on Algorithms and Computation, (ISAAC '01)*, volume 2223 of *Lecture Notes in Computer Science*, 291–307. Springer.

Gaspers, S.; Kim, E. J.; Ordyniak, S.; Saurabh, S.; and Szeider, S. 2012. Don't be strict in local search! In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI '12)*. AAAI Press.

Karp, R. M. 1972. Reducibility among combinatorial problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*, The IBM Research Symposia Series, 85–103. Plenum Press, New York.

Khot, S., and Regev, O. 2008. Vertex cover might be hard to approximate to within $2-\epsilon$. *J. Comput. Syst. Sci.* 74(3):335–349.

Kunegis, J. 2013. KONECT: the Koblenz network collection. In *Proceedings of the 22nd International World Wide Web Conference (WWW '13)*, 1343–1350. International World Wide Web Conferences Steering Committee / ACM.

Rossi, R. A., and Ahmed, N. K. 2015. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI '15)*, 4292–4293. AAAI Press.

Wang, Y.; Cai, S.; and Yin, M. 2016. Two efficient local search algorithms for maximum weight clique problem. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI '16)*, 805–811. AAAI Press.

Zuckerman, D. 2007. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing* 3(1):103–128.