Local Search with Efficient Automatic Configuration for Minimum Vertex Cover

Chuan Luo[1,2] , **Holger H. Hoos**[2,3] , **Shaowei Cai**[4] , **Qingwei Lin**[1] ,
Hongyu Zhang[5] and **Dongmei Zhang**[1]
[1]Microsoft Research, China
[2]Leiden Institute of Advanced Computer Science, Leiden University, The Netherlands
[3]Department of Computer Science, University of British Columbia, Canada
[4]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
[5]The University of Newcastle, Australia
{chuan.luo, qlin, dongmeiz}@microsoft.com, hh@liacs.nl, caisw@ios.ac.cn,
hongyu.zhang@newcastle.edu.au

Abstract

Minimum vertex cover (MinVC) is a prominent NP-hard problem in artificial intelligence, with considerable importance in applications. Local search solvers define the state of the art in solving MinVC. However, there is no single MinVC solver that works best across all types of MinVC instances, and finding the most suitable solver for a given application poses considerable challenges. In this work, we present a new local search framework for MinVC called *MetaVC*, which is highly parametric and incorporates many effective local search techniques. Using an automatic algorithm configurator, the performance of *MetaVC* can be optimized for particular types of MinVC instances. Through extensive experiments, we demonstrate that *MetaVC* significantly outperforms previous solvers on medium-size hard MinVC instances, and shows competitive performance on large MinVC instances. We further introduce a neural-network-based approach for enhancing the automatic configuration process, by identifying and terminating unpromising configuration runs. Our results demonstrate that *MetaVC*, when automatically configured using this method, can achieve improvements in the best known solutions for 16 large MinVC instances.

1 Introduction

Given an undirected graph $G = (V, E)$, a vertex cover is a set of vertices $S \subseteq V$, such that each edge $e \in E$ has at least one endpoint in S. The problem of minimum vertex cover (MinVC) is to find a vertex cover of minimum size in a given undirected graph.

MinVC is a prominent problem in artificial intelligence, combinatorial optimization and graph theory, with a broad range of real-world applications in feature selection [Xie and Qin, 2018], network security [Cai *et al.*, 2017] and sensor networks [Kavalci *et al.*, 2014]. In computational theory, MinVC is a widely studied NP-hard problem, whose optimal solutions are known to be hard to approximate; specifically, it is NP-hard to approximate the optimal solutions for MinVC within any factor smaller than 1.3606 [Dinur and Safra, 2005]. MinVC is closely related to the maximum independent set problem [Lamm *et al.*, 2017; Chang *et al.*, 2017], which has many important real-world applications.

Because of the practical importance of MinVC, many MinVC solvers [Akiba and Iwata, 2016; Cai *et al.*, 2017] have been proposed. Due to the NP-hardness of the problem, much of the research on solving MinVC is focused on heuristic algorithms, including construction heuristics [Khalil *et al.*, 2017; Xu *et al.*, 2018] and local search [Richter *et al.*, 2007; Ma *et al.*, 2016b; Cai *et al.*, 2017; Friedrich *et al.*, 2017; Weise *et al.*, 2019]. Local search approaches are able to find vertex cover with much smaller size than construction approaches, and have been studied particularly intensely, resulting in 3 state-of-the-art local search solvers for MinVC: *NuMVC* [Cai *et al.*, 2013], *TwMVC* [Cai *et al.*, 2015] and *FastVC2+p* [Cai *et al.*, 2017]. Also, local search approaches are known to be effective in solving other NP-hard problems in graph theory [Pullan and Hoos, 2006; Fan *et al.*, 2019].

However, currently there is no single solver that achieves good performance across all types of MinVC instances. From our experiments with several state-of-the-art MinVC solvers, described in Tables 1–4, *NuMVC* and *TwMVC* perform much worse than *FastVC2+p* for solving large instances, while *FastVC2+p* shows much worse performance than *NuMVC* and *TwMVC* for solving medium-size hard instances.

It is well known that different types of MinVC instances are best solved using different techniques; nevertheless, current state-of-the-art MinVC solvers only incorporate a limited number of algorithmic strategies. To address this issue, Khalil *et al.* have recently used reinforcement learning to directly learn a suitable MinVC construction algorithm for each benchmark, resulting in a MinVC solver called *S2V-DQN* [Khalil *et al.*, 2017]. However, their empirical results only cover graphs with up to 1200 vertices, and in practice there remains a significant performance gap between *S2V-DQN* and state-of-the-art MinVC solvers.

A promising algorithm design paradigm dubbed programming by optimization (PbO) [Hoos, 2012] urges algorithm developers to significantly expand the design space of an algorithm under development, to actively consider design alternatives and to expose parameters. PbO-based approaches

have shown effectiveness in many NP-hard problems, *e.g.,* propositional satisfiability [KhudaBukhsh *et al.*, 2016]. To our best knowledge, there is no previous work applying the idea of PbO to MinVC solving.

In this work, we endeavour to design a more effective and robust local search solver for MinVC, with the goal of advancing the state of the art in MinVC solving. Our main contributions are as follows.

First, following the PbO paradigm, we propose a rich and flexible local search framework for MinVC called *MetaVC*, which is highly parametric. *MetaVC* is a meta-solver, which provides a top-level design, where each key function is an abstraction and can be instantiated with a concrete function.

Different from previous local-search-based MinVC solvers that apply the same technique to any given instance, *MetaVC* incorporates various techniques that are automatically customized and combined, using an algorithm configurator, for effectively solving MinVC instances. Extensive experiments show that *MetaVC* performs much better than *NuMVC* and *TwMVC* on medium-size hard instances, indicating the effectiveness of the *MetaVC* framework. In addition, *MetaVC* achieves competitive performance compared to *FastVC2+p* on large MinVC instances.

Second, to improve the performance of *MetaVC* on large MinVC instances, we introduce a neural-network-based approach to enhance automatic configuration. During automatic configuration, we use a neural network model to identify and terminate unpromising runs. Intuitively, this can save much time and explore many configurations within a given time budget. We use this approach to configure *MetaVC* on large instances; the resulting configuration of *MetaVC*, dubbed *MetaVC2*, significantly outperforms *FastVC2+p* on nearly all instances. Notably, *MetaVC2* can improve the best known solutions for 16 large MinVC instances, further confirming the effectiveness of the *MetaVC* framework.

2 Preliminaries

An undirected graph $G = (V, E)$ is defined by a vertex set V and an edge set $E \subseteq V \times V$. For an edge $e = \{v, u\}$, where $v, u \in V$, v and u are *endpoints* of e. Given an undirected graph $G = (V, E)$, the problem of *minimum vertex cover (MinVC)* is to find a subset $S \in V$ such that every edge in E has at least one endpoint in S.

For MinVC, a *candidate solution* C is a subset of the vertex set V. Given a candidate solution C, an edge $e \in E$ is *covered* by C if at least one endpoint e belongs to C, and *uncovered* otherwise; the *state* of a vertex $v \in V$ is modelled by a Boolean variable: $True$ indicates $v \in C$, and $False$ means $v \notin C$. For a vertex $v \in C$, the *loss* of v is the number (or the total weight, when using an edge weighting scheme) of covered edges that would become uncovered by removing v from C. For a vertex $v \notin C$, the *gain* of v is the number (or the total weight, when using edge weighting) of uncovered edges that would become covered by adding v into C. The *age* of a vertex v is the number of search steps since the last change of v's state. The *degree* of a vertex v is the number of edges that have v as an endpoint.

Algorithm 1 The pseudo-code of *MetaVC* framework

```
Input: graph G = (V, E)
Output: vertex cover of G
1  if performPreProcess then
2      run pre-processing techniques on graph G;
3  generate the initial vertex cover C via initConstruct();
4  C* := C;
5  while no termination criterion is met do
6      if no uncovered edge exists then
7          C* := C;
8          remove a vertex with the smallest loss from C;
9          continue;
10     if performReConstruct then
11         if with probability prob_rc then
12             C := ReConstruct(C);
13             continue;
14     if performBMS then S := filterBMS(C);
15     else S := C;
16     v := pickRmVertex();
17     C := C \ {v};
18     e := pickUncovEdge();
19     u := pickAddVertex(e);
20     C := C ∪ {u};
21     if performEdgeWeight then
22         run edge weighting scheme edgeWeight();
23 return C*;
```

3 Meta Local Search Framework for MinVC

In this section, we present *MetaVC*, our new local search framework for solving MinVC. Different from existing MinVC solvers, which apply the same technique to any given instance, *MetaVC* is a novel algorithmic framework that is highly parametric and includes many effective algorithmic techniques for MinVC in its design space. As a result of this, *MetaVC* can achieve state-of-the-art performance for different types of MinVC instances when using effective parameter settings and choices of algorithmic components.

The top-level design of *MetaVC* is comprised of three essential phases (see also Algorithm 1): pre-processing, construction and search. In pre-processing, the given graph is simplified. In construction, an initial vertex cover is generated to obtain a good starting point for the subsequent local search process. In the search phase, *MetaVC* conducts local search to iteratively optimize the vertex cover.

3.1 Pre-processing Phase

Effective pre-processing techniques can considerably simplify the given graph and thus reduce the search space, which renders the given instance easier [Cai *et al.*, 2017]. To boost performance, *MetaVC* integrates an effective pre-processing component in the beginning (Lines 1–2 in Algorithm 1).

The activation of the pre-processing component is controlled by a Boolean parameter $performPreProcess$: if $performPreProcess$=$True$, the pre-processing component is called. In particular, *MetaVC* employs the pre-processing

technique used by *FastVC2+p*, which consists of four reduction rules [Cai *et al.*, 2017].

3.2 Construction Phase

The construction component plays a critical role in MinVC solvers; a good initial solution can lead to performance improvements [Khalil *et al.*, 2017]. To make construction efficient, *MetaVC* incorporates two simple yet effective construction methods, resulting in 2 instantiations of the construction component *initConstruct* (Lines 3–4 in Algorithm 1).

(1) **Gain-based method:** Starting with an empty vertex set C, repeat adding a vertex $v \notin C$ with the greatest *gain* to C, until C becomes a vertex cover.

(2) **Degree-based method:** Starting with an empty vertex set C, repeat adding a vertex $v \notin C$ with the greatest *degree* to C, until C becomes a vertex cover.

3.3 Search Phase

The search phase is the most important component of all local-search-based MinVC solvers. In the search phase, *MetaVC* solves MinVC by iteratively tackling the associated decision problem (Lines 5–22 in Algorithm 1). The general scheme of the search phase is as follows: whenever a vertex cover of size n is found, one vertex is removed, and then *MetaVC* continues to search for a vertex cover of size $n-1$.

For solving the decision problem, *MetaVC* employs an iterative method. In each iteration, *MetaVC* exchanges a pair of vertices: given the current vertex cover C, a vertex $v \in C$ is removed from C, and a vertex $u \notin C$ is added to C. The heuristics used to determine the vertex v to be removed and the vertex u to be added are crucial for this procedure.

To diversify the search process, we use a reconstruction mechanism *ReConstruct* in the beginning of each iteration (Lines 10–13 in Algorithm 1). The activation of *ReConstruct* is controlled by a Boolean parameter $performReConstruction$ and a probability $prob\_rc$. If $performReConstruction$=$True$ and with probability $prob\_rc$, *ReConstruct* works as follows: given the current vertex cover C, t vertices originally in C are removed from C; then the t vertices with the greatest *gains* are selected from $V \setminus C$ (the set of vertices previously not in C) and added to C (where t is an integer-valued parameter).

Additionally, *MetaVC* invokes an edge weighting scheme (Lines 21–22 in Algorithm 1) after the exchange step, to further strengthen diversification.

Heuristics for Selecting the Vertex to be Removed

To achieve a good balance between the time complexity and the quality of the selected vertex (*i.e.,* the *loss* value), *MetaVC* performs a filter process based on the BMS heuristic (*filterBMS*) [Cai *et al.*, 2017] before the greedy selection (*pickRmVertex*). The activation of *filterBMS* is depended on a Boolean parameter $performBMS$. If activated, *filterBMS* will filter the vertices in the current vertex cover C via BMS[1] [Cai *et al.*, 2017], resulting in a smaller candidate vertex set $S \subseteq C$; otherwise, S is the same as C. Then *pickRmVertex* selects a vertex $v \in S$ to be removed from S. *MetaVC* implements 3 instantiations of *pickRmVertex*:

(1) Select a vertex $v \in S$ with the smallest *loss*, ties broken for vertices with higher *age*.

(2) Select a vertex $v \in S$ with the smallest *loss*, ties broken uniformly at random.

(3) Select a vertex based on configuration checking [Cai *et al.*, 2013].

Heuristics for Selecting the Vertex to be Added

For selecting the vertex to be added to the current candidate solution, existing state-of-the-art MinVC solvers usually select a vertex from an uncovered edge. Following this approach, *MetaVC* also selects a vertex to be added from an uncovered edge (*pickUncovEdge*). Different from existing solvers, which randomly select an uncovered edge, we developed a new approach, based on edge weighting, and integrated it into the design space of *MetaVC*. In this work, *MetaVC* supports 2 instantiations of *pickUncovEdge*:

(1) Select an uncovered edge uniformly at random.

(2) If the edge weighting scheme is activated, select an uncovered edge according to a probability distribution, with probability proportional to the edge weight.

After that, *MetaVC* selects the vertex to be added from the uncovered edge e (*pickAddVertex*); there are 3 instantiations of *pickAddVertex*:

(1) Select a vertex u in e with the greatest *gain*, ties broken for vertices with higher *age*.

(2) Select a vertex u in e with the greatest *gain*, ties broken uniformly at random.

(3) Select a vertex based on a tabu method [Cai *et al.*, 2010].

Edge Weighting Schemes

Inspired by the success of edge weighting schemes [Cai *et al.*, 2013; Cai *et al.*, 2015] in MinVC solving, we also implemented edge weighting (*edgeWeight*) in *MetaVC*. In this work, *MetaVC* supports 2 instantiations of *edgeWeight*:

(1) An additive edge weighting scheme [Cai *et al.*, 2015].

(2) A new multiplicative edge weighting scheme, inspired by the clause weighting scheme used in the well-known SAT solver, *SAPS* [Hutter *et al.*, 2002].

3.4 Automatic Configuration Process

MetaVC is a highly parametric framework and can be configured to instantiate many novel local search solvers for MinVC. In particular, the parameters and the components of *MetaVC* described in Sections 3.1–3.3 are configurable, and their settings can be automatically determined by a general-purpose algorithm configurator.

To maximize performance for a given class of benchmark instances, we used a state-of-the-art automatic algorithm configurator called *SMAC* [Hutter *et al.*, 2011] to configure *MetaVC*. The full configuration space (*i.e.,* the search space for *SMAC*) and the default configuration (*i.e.,* the starting point for *SMAC*) are described online.[2]

[1] Due to the page limit, please refer to the literature [Cai *et al.*, 2017] for details of the BMS heuristic.

[2] https://github.com/chuanluocs/MetaVC

Graphs		*MetaVC*	*NuMVC*	*TwMVC*	*FastVC2+p*
Instance Name	VC^*	succ. rate time	succ.~rate time	succ.~rate time	succ. rate time
brock400_4	367*	100% **0.41**	100% 4.29	100% 3.36	100% 161.03
brock800_4	774*	**100%** **537.74**	56% 2336.36	30% 3102.01	1% 3579.29
MANN_a45	690*	100% **57.08**	100% 92.49	99% 524.86	0% 3600.00
brock400_2	371*	100% **6.68**	100% 177.56	100% 120.30	22% 3253.92
brock800_2	776*	**67%** **2317.45**	2% 3557.66	5% 3517.54	0% 3600.00
C2000.9	1920	**10%** **3470.28**	4% 3523.01	0% 3600.00	0% 3600.00
C4000.5	3982	100% 219.04	100% **171.45**	100% 195.64	76% 940.83
MANN_a81	2221	**87%** **1517.13**	40% 2751.22	37% 2915.07	0% 3600.00

Table 1: Results for *MetaVC*, *NuMVC*, *TwMVC* and *FastVC2+p* on the *DIMACS-HARD* benchmarks. For each instance, entries in the VC^* column marked with an asterisk ('*') indicate that the best known vertex cover size is known to be provably optimal. For the *brock* instance family, each solver was evaluated with its optimized configuration trained on 2 instances (*brock400_4* and *brock400_4*). For other instances, each solver was evaluated with its optimized configuration trained on a single instance (*MANN_a45*).

Following the recommend protocol [Hutter *et al.*, 2017], we utilized *SMAC* to optimize solution quality – here, size of the vertex cover; we allowed a time budget of 2 days for the entire configuration process, and used a cutoff time of 300 CPU seconds for each solver run during the configuration process. For each training set of instances, we conducted 25 independent runs of *SMAC*, resulting in 25 optimized configurations of *MetaVC*. Next, each of the resulting configurations was evaluated on all training instances. Finally, the configuration with the best solution quality (*i.e.,* the lowest average size of the found vertex cover across all training instances) was chosen as the final result.

3.5 Experimental Setup

To study the performance of the MinVC solvers considered in this work, we conducted extensive experiments and analyzed the results thus obtained. Here we introduce the benchmarks, the MinVS solvers and our evaluation methodology.

For our experiments, we chose 3 prominent benchmarks, which have been widely used in previous work on MinVC [Cai *et al.*, 2015; Cai *et al.*, 2017; Ma *et al.*, 2016a; Wagner *et al.*, 2017].

- **DIMACS-HARD:** The 8 hardest instances from the *DIMACS* benchmark,[3] which is taken from the the Second DIMACS Challenge Test Problems.
- **BHOSLIB-HARD:** The 15 hardest instances from the *BHOSLIB* benchmark,[4] generated in the hardest area of model RB [Xu *et al.*, 2007].

[3]http://lcs.ios.ac.cn/~caisw/Resource/DIMACS%20complementary%20graphs.tar.gz

[4]http://sites.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm

Graphs		*MetaVC*	*NuMVC*	*TwMVC*	*FastVC2+p*
Instance Name	VC^*	succ. rate time	succ. rate time	succ. rate time	succ. rate time
frb53-24-1	1219*	92% **1357.84**	92% 1480.43	82% 1722.53	9% 3448.58
frb53-24-2	1219*	100% 264.69	100% **226.22**	100% 410.92	60% 2352.37
frb53-24-3	1219*	100% **64.29**	100% 65.68	100% 81.37	100% 388.35
frb53-24-4	1219*	100% **316.75**	100% 425.73	100% 482.50	87% 1508.22
frb53-24-5	1219*	100% **48.98**	100% 52.47	100% 60.01	100% 333.02
frb56-25-1	1344*	98% 871.65	**99%** **731.73**	94% 1131.59	47% 2745.71
frb56-25-2	1344*	95% **1005.04**	95% 1159.36	94% 1401.79	16% 3333.33
frb56-25-3	1344*	100% **140.65**	100% 173.62	100% 178.85	57% 2343.79
frb56-25-4	1344*	100% **57.10**	100% 76.58	100% 68.98	100% 466.30
frb56-25-5	1344*	100% **28.75**	100% 39.80	100% 44.11	100% 197.27
frb59-26-1	1475*	85% 1436.38	**90%** 1557.70	79% 1870.86	1% 3597.38
frb59-26-2	1475*	**45%** **2747.71**	37% 2926.69	35% 2938.82	0% 3600.00
frb59-26-3	1475*	**100%** **753.39**	96% 1025.92	99% 1020.74	39% 2727.73
frb59-26-4	1475*	**92%** **1187.73**	86% 1666.28	66% 2226.29	36% 2843.16
frb59-26-5	1475*	100% **68.90**	100% 88.27	100% 142.36	100% 114.88

Table 2: Results for *MetaVC*, *NuMVC*, *TwMVC* and *FastVC2+p* on the *BHOSLIB-HARD* benchmark. Each instance in this set has a hidden optimal vertex cover, whose size is indicated in the VC^* column and marked with an asterisk ('*'). For all instances, each solver was evaluated using the optimized configuration trained on the 5 instances shown in the upper part of the table, in *italics*.

- **REAL-WORLD-HARD:** The 31 hardest instances from the *REAL-WORLD* benchmark,[5] which is comprised of all undirected simple graphs (not including *DIMACS* and *BHOSLIB* graphs) from the Network Data Repository [Rossi and Ahmed, 2015].

To configure *MetaVC*, we needed to construct a training set for each benchmark. For *DIMACS-HARD*, *BHOSLIB-HARD* and *REAL-WORLD-HARD*, we selected 3, 5 and 12 instances, respectively.[6] The training instances for each benchmark are indicated in *italics* and shown in the upper parts of Tables 1 and 2 as well as the whole of Table 3. The detailed procedures for selecting training instances are described online.[2]

We selected 3 state-of-the-art local search MinVC solvers as the competitors to our approach. *NuMVC* [Cai *et al.*, 2013] and *TwMVC* [Cai *et al.*, 2015] are the best solvers currently known for solving medium-size hard MinVC instances, such as *DIMACS-HARD* and *BHOSLIB-HARD*. *FastVC2+p* [Cai *et al.*, 2017] is the best solver currently known for solving large-sized MinVC instances, such as *REAL-WORLD-HARD*.

[5]http://lcs.ios.ac.cn/~caisw/Resource/realworld%20graphs.tar.gz

[6]As reported in the literature [Cai *et al.*, 2015], with respect to *DIMACS-HARD*, the configuration used for the *brock* instances is different from those used for the *C* and *MANN* instances. Hence, we separately configured *MetaVC* on the *brock* instances.

Graphs		*MetaVC2*	*MetaVC*	*NuMVC*	*TwMVC*	*FastVC2+p*
Instance Name	VC^*	min (avg) time	min (avg) time	min (avg) time	min (avg) time	min (avg) time
*inf-roadNet-PA**	555046	**554320 (554346.98)** 3207.33	554530 (554582.20) 3535.97	560061 (560166.87) 1338.30	579584 (580311.25) 3599.62	555029 (555079.56) 460.39
sc-msdoor	381558	381558 (381558.00) **42.88**	381558 (381558.00) 311.35	381559 (381563.60) 1530.07	381560 (381562.44) 3086.87	381558 (381558.40) 787.99
*sc-nasasrb**	51239	51238 **(51240.06)** 1056.52	51239 (51240.90) 1188.29	51241 (51243.93) 2602.79	51238 (51244.95) 1477.46	51239 (51240.59) 788.41
*sc-pkustk13**	89217	89232 (89234.12) 1195.98	89223 (89224.66) 1208.97	**89216 (89217.58)** 2773.71	89226 (89230.04) 623.05	89227 (89230.19) 350.75
soc-livejournal	1868903	**1868905 (1868910.32)** 1330.85	1868907 (1868910.69) 2162.63	1875974 (1878644.78) 3599.57	N/A (N/A) N/A	1868916 (1868918.17) 70.17
socfb-CMU	4986	4986 (4986.00) **8.64**	4986 (4986.41) 688.20	4986 (4986.00) 441.86	4986 (4986.73) 462.51	4986 (4986.10) 775.75
socfb-OR	36547	36547 **(36547.00)** 340.18	36549 (36550.08) 1226.69	36555 (36558.88) 2038.75	36562 (36568.78) 2205.01	36547 (36548.33) 904.39
*socfb-UCLA**	15222	**15221 (15222.41)** 983.15	15222 (15223.60) 1067.07	15223 (15226.42) 1520.18	15227 (15230.47) 1907.70	15222 (15223.72) 808.39
socfb-UConn	13230	13230 **(13230.03)** 751.81	13230 (13231.67) 1189.18	13231 (13233.00) 1397.86	13233 (13236.05) 1834.57	13230 (13230.98) 725.68
web-it-2004	414507	414515 (414516.06) 1024.03	414515 **(414515.95)** 1260.80	414699 (414718.73) 1552.27	414688 (414696.31) 3058.27	414524 (414527.02) 717.16
web-webbase-2001	2651	2652 (2652.00) **<0.01**	2652 (2652.00) 0.10	**2651 (2651.87)** 132.98	2652 (2652.00) 1.85	2652 (2652.00) <0.01
web-wikipedia2009	648294	648294 **(648294.00)** 228.54	648294 (648296.36) 1680.80	649192 (649246.29) 1831.47	N/A (N/A) N/A	648302 (648312.56) 407.67

Table 3: Results for *MetaVC2*, *MetaVC*, *NuMVC*, *TwMVC* and *FastVC2+p* on the 12 training instances from *REAL-WORLD-HARD*. For each instance, we use an asterisk (*) in the 'Instance Name' column to indicate that one of the solvers found a smaller vertex cover than the best known results from the literature. For all training instances, each solver was evaluated with its optimized configuration trained on the 12 training instances shown in this table. We use 'N/A' to indicate cases where none of the runs of a specific solver were successful.

The source code of *NuMVC* is publicly available,[7] and the sources for *TwMVC* and *FastVC2+p* were provided by their authors. *NuMVC*, *TwMVC* and *FastVC2+p* have 2, 3 and 1 configurable parameters, respectively.

In this work, all our experiments were carried out on a cluster of computers, where each computer is equipped with 32 Intel Xeon E5-2683 CPUs and 94 GB memory, running the operating system of CentOS 7.6.1810. For each solver, we performed 100 independent runs per instance, with a cutoff time of one hour per run. For each instance, we considered the size of the optimal (or previously best known) vertex cover ('VC^*').[8] Following previous work on medium-size hard instances [Cai *et al.*, 2015], for each instance from *DIMACS-HARD* and *BHOSLIB-HARD*, we report the success rate ('succ rate'), *i.e.,* the number of successful runs divided by the total number of runs, where a run is considered successful if a vertex cover with the size of 'VC^*' is found, and the running time ('time') measured in CPU seconds required for finding a cover of size VC^*, averaged over total runs. Furthermore, following previous work on large instances [Cai *et al.*, 2017], for each instance from *REAL-WORLD-HARD*, we report the minimum ('min') and average ('avg') cover size found by the respective solver over total runs, and the running time ('time') measured in CPU seconds required for obtaining the final solutions averaged over total runs. For each solver, if there are no successful runs, we report 'N/A'. For each instance, we use **boldface** to indicate the best results.

[7]http://lcs.ios.ac.cn/~caisw/Code/NuMVC_v2015.8.zip

[8]For *DIMACS-HARD* and *BHOSLIB-HARD*, we used the VC^* values reported in the literature [Cai *et al.*, 2015]. For *REAL-WROLD-HARD*, we collected the VC^* values from a number of previous studies [Ma *et al.*, 2016a; Ma *et al.*, 2016b; Cai *et al.*, 2017].

3.6 Experimental Results

The results from our experiments with *MetaVC* and all competitors on all benchmarks are reported in Tables 1–4. The empirical comparison is fair, because for each benchmark, all competitors were configured using *SMAC* with the same configuration protocol. The optimized configurations for *MetaVC* are shown in Table 5 and can be found online, along with the optimized configurations for the competitors.[2]

On the medium-size hard benchmarks (*i.e., DIMACS-HARD* and *BHOSLIB-HARD*), *MetaVC* stands out as the best solver and performs much better than all other solvers. In particular, on 4 difficult instances in *DIMACS-HARD* (*brock800_4*, *brock800_2*, *C2000.9* and *MANN_a81*), *MetaVC* achieved much higher success rates than its competitors; for example, on instance *brock800_2*, we observed a success rate of 67% for *MetaVC*, compared to 2%, 5% and 0% for *NuMVC*, *TwMVC* and *FastVC2+p*, respectively. On the hardest family within *BHOSLIB-HARD*, *frb59-26*, *MetaVC* performed best on 4 out of 5 instances. These results clearly indicate that *MetaVC* advances the state of the art in solving medium-size hard MinVC instances.

As seen in Tables 3 and 4, *MetaVC* and *FastVC2+p* significantly outperformed the other two solvers on the large real-world instances. These results demonstrate that *MetaVC* achieves performance competitive to that of *FastVC2+p* on large instances. For almost all other, easy instances from the original *DIMACS,* BHOSLIB *and* REAL-WORLD benchmarks (not included in *DIMACS-HARD*, *BHOSLIB-HARD* and *REAL-WORLD-HARD*), *MetaVC* always found the best known solutions with a success rate of 100%.[9]

[9]In fact, it failed to do so only for a single instance, *socfb-Stanford3* from *REAL-WORLD*, for which the best solution qual-

Graphs		*MetaVC2*	*MetaVC*	*NuMVC*	*TwMVC*	*FastVC2+p*
Instance Name	VC^*	min (avg) time	min (avg) time	min (avg) time	min (avg) time	min (avg) time
inf-road-usa*	11527630	**11525352 (11525847.64)** 3496.00	11527313 (11527749.68) 3569.33	12238416 (12238420.60) 29.25	N/A (N/A) N/A	11527505 (11527723.35) 2706.00
inf-roadNet-CA*	1001065	**999854 (999911.64)** 3380.57	1000347 (1000500.03) 3492.74	1010709 (1013890.06) 3599.80	N/A (N/A) N/A	1001052 (1001103.34) 888.15
sc-ldoor	856754	856754 (856754.00) **15.62**	856754 (856754.00) 110.05	856798 (856808.74) 3533.05	856996 (857067.75) 3593.76	856754 (856754.01) 606.23
sc-pwtk*	207673	207674 (207676.66) 1183.57	207674 (207678.02) 2095.84	207724 (207736.31) 203.48	207688 (207702.96) 2359.07	**207671 (207676.19)** 2280.41
sc-shipsec1*	117246	**116849 (116871.15)** 3129.61	116922 (116945.75) 3472.12	117129 (117207.61) 3438.61	117332 (117371.38) 3178.57	117224 (117259.71) 3465.81
sc-shipsec5*	147043	**146768 (146787.57)** 3366.05	146912 (146967.64) 3495.28	147091 (147129.83) 3568.67	147055 (147091.11) 2573.51	147028 (147046.51) 3239.04
soc-delicious*	85341	85358 (85372.75) 2801.96	85368 (85382.47) 2894.37	85518 (85558.60) 3056.06	85474 (85528.86) 199.72	**85336 (85340.26)** 1232.35
soc-orkut*	2171200	**2170773 (2170854.66)** 3350.66	2171860 (2171951.40) 3563.37	N/A (N/A) N/A	N/A (N/A) N/A	N/A (N/A) N/A
soc-pokec*	843377	**843348 (843355.56)** 3223.25	843364 (843374.87) 3288.76	844296 (844338.42) 1055.88	N/A (N/A) N/A	843375 (843380.55) 1686.59
socfb-Berkeley13*	17210	17209 **(17209.91)** 774.61	17210 (17212.98) 1122.34	17213 (17216.26) 1636.39	17218 (17221.00) 2020.29	17209 (17211.48) 716.95
socfb-Indiana*	23314	23313 **(23313.97)** 926.32	23314 (23317.30) 1685.76	23317 (23323.67) 2272.78	23322 (23329.57) 1879.88	23313 (23315.78) 1358.08
socfb-Penn94*	31161	**31158 (31159.82)** 1201.90	31161 (31164.72) 1514.71	31168 (31178.93) 1970.55	31181 (31190.28) 1803.34	31159 (31162.35) 1362.67
socfb-Texas84*	28165	**28164 (28165.60)** 908.47	28166 (28170.83) 1751.38	28170 (28175.56) 2518.91	28180 (28188.93) 1926.83	28166 (28167.54) 1251.17
socfb-UCSB37	11261	11261 **(11261.00)** 52.98	11261 (11262.08) 422.38	11261 (11262.74) 1147.20	11262 (11264.36) 1878.33	11261 (11261.56) 943.51
socfb-UF*	27305	**27303 (27303.90)** 1046.48	27305 (27308.52) 1855.44	27309 (27319.16) 2144.79	27321 (27326.52) 1978.55	27304 (27307.18) 1455.44
socfb-UIllinois*	24091	**24089 (24090.48)** 1228.43	24092 (24094.00) 1628.33	24096 (24103.25) 2059.37	24101 (24109.15) 2019.51	24090 (24092.46) 1468.54
socfb-Wisconsin87*	18383	**18382 (18382.93)** 352.19	18383 (18384.89) 1564.20	18386 (18389.55) 1554.29	18390 (18394.67) 2099.69	18383 (18384.08) 797.49
tech-RL-caida	74593	74621 (74630.29) 2758.99	74621 **(74629.39)** 2968.13	74691 (74741.37) 3488.23	74697 (74714.98) 1085.94	74626 (74631.21) 3180.03
tech-as-skitter	525163	**525183 (525196.02)** 2737.99	525186 (525200.99) 3322.72	525924 (525957.09) 1054.60	526576 (526813.02) 3599.31	525247 (525260.18) 3367.72

Table 4: Results for *MetaVC2*, *MetaVC*, *NuMVC*, *TwMVC* and *FastVC2+p* on the 19 testing instances from *REAL-WORLD-HARD*. For each testing instance, we use an asterisk (*) in the 'Instance Name' column to indicate that one of the solvers found a smaller vertex cover than the best known results from the literature. For all testing instances, each solver was evaluated with its optimized configuration trained on the 12 training instances shown in Table 3. We use 'N/A' to indicate cases where none of the runs of a specific solver were successful.

4 A Neural-Network-based Approach for Enhancing Automatic Configuration

The comparison between *MetaVC* and *FastVC2+p* in Tables 3 and 4 indicates that there is still room for improving the performance of *MetaVC* on solving large real-world instances; this motivated us to explore better configurations of our *MetaVC* framework. The starting point for our work in this area was the observation that during automatic configuration, much time could be wasted on evaluating bad configurations of the given target algorithm [Domhan *et al.*, 2015]. Here, we propose a neural-network-based method for identifying and terminating unpromising target algorithm runs during the automatic configuration process.

4.1 LSTM-Network-based Predictive Model

Since *MetaVC* is able to output its current best solution quality at any time, for each solver run, we can observe the time series vector $y_{1:p}$, where y_i $(1 \leq i \leq p)$ denotes the solution quality output by *MetaVC* at time i. In this work, we record one solution quality value per CPU second. The main idea is that, using the observed time series vector $y_{1:p}$, we can build a predictive model for solution quality y_q $(q > p)$; if y_q in the current run is significantly worse than the value in previous runs, the current run is identified as unpromising.

As the predictive model will be deployed in a real-time environment, we need to train it fast. Long short-term memory (LSTM) networks [Hochreiter and Schmidhuber, 1997] satisfy this criterion and have been used successfully for time series prediction [Schmidhuber *et al.*, 2005]. To keep training time at a minimum, we use a LSTM-based model with low network complexity. In particular, we use only two layers, where the first is a LSTM layer, while the second is a dense layer. Our LSTM-based model minimizes mean squared error using the *Adam* optimizer [Kingma and Ba, 2015].

We used our LSTM-based model to enhance automatic configuration in Section 3.4. During automatic configuration, for each training instance, the entire time series vector $y^*_{1:T}$ observed in the best solver run is recorded and updated as needed. Whenever *SMAC* performs a solver run, our model is

ity is 8517. Although *MetaVC*, *NuMVC*, *TwMVC* and *FastVC2+p* all failed to achieve that solution quality, all of them found a vertex cover of size 8518 with the success rate of 100%. Furthermore, *MetaVC2*, which will be presented in Section 4, was able to reach the best solution quality for this instance with a success rate of 100%.

Benchmark	Optimized Configuration
brock-HARD	$performPreProcess$=$True$, $initConstruct$=2, $performReConstruct$=$True$, $prob\_rc$=0.0028421872317207584, t=100, $performBMS$=$False$, $pickRmVertex$=3, $pickUncovEdge$=2, $pickAddVertex$=1, $performEdgeWeight$=$True$, $edgeWeight$=2
DIMACS-HARD	$performPreProcess$=$False$, $initConstruct$=1, $performReConstruct$=$False$, $performBMS$=$False$, $pickRmVertex$=1, $pickUncovEdge$=2, $pickAddVertex$=1, $performEdgeWeight$=$True$, $edgeWeight$=1
BHOSLIB-HARD	$performPreProcess$=$True$, $initConstruct$=2, $performReConstruct$=$False$, $performBMS$=$False$, $pickRmVertex$=2, $pickUncovEdge$=1, $pickAddVertex$=3, $tabu\_tenure$=5, $performEdgeWeight$=$True$, $edgeWeight$=1
REAL-WORLD-HARD	$performPreProcess$=$True$, $initConstruct$=2, $performReConstruct$=$True$, $prob\_rc$=3.908659583029911E-5, t=84, $performBMS$=$True$, bms_k=633, $pickRmVertex$=2, $pickUncovEdge$=1, $pickAddVertex$=2, $performEdgeWeight$=$False$

Table 5: The optimized configurations of *MetaVC* for all benchmarks.

Benchmark	Optimized Configuration
REAL-WORLD-HARD	$performPreProcess$=$True$, $initConstruct$=2, $performReConstruct$=$True$, $prob\_rc$=3.0886947578801404E-5, t=76, $performBMS$=$True$, bms_k=720, $pickRmVertex$=1, $pickUncovEdge$=1, $pickAddVertex$=2, $performEdgeWeight$=$False$

Table 6: The optimized configuration of *MetaVC2* for the *REAL-WORLD-HARD* benchmark.

activated simultaneously. During the current solver run, our model is trained every t CPU seconds, using the latest t observations; at time j, our model predicts the solution quality y_{j+r} expected r CPU seconds later; if $y_{j+r} \leq (1+\delta) \cdot y^*_{j+r}$ (recalling that MinVC is a minimization problem), the current run is allowed to continue; otherwise, the current *MetaVC* run is identified as unpromising and terminated immediately.

Svegliato *et al.* proposed a performance predictor based on nonlinear regression to stop anytime algorithms at a time point suitable for obtaining good running time and solution quality [Svegliato *et al.*, 2018]. In contrast, our predictive model is based on an LSTM network and aims to enhance automatic algorithm configuration, in order to find better configurations of a given target algorithm within limited time.

4.2 Empirical Evaluation

To show the effectiveness of our LSTM-based approach, we used the enhanced automatic configuration method (described above) to configure *MetaVC* on the *REAL-WORLD-HARD* benchmark.[10] Besides incorporating our LSTM-based approach, we used the same automatic configuration process as described in Section 3.4. The version of *MetaVC* with the optimized configuration obtained from our enhanced automatic configuration method is denoted as *MetaVC2*; it is shown in Table 6 and also available online.[2]

For our empirical evaluation, we used the evaluation methodology and execution environment described in Section 3.5. Performance results for *MetaVC2*, *MetaVC* and their competitors on the *REAL-WORLD-HARD* benchmark are summarized in Tables 3 and 4. It is clear that *MetaVC2* significantly outperforms *MetaVC* and all other competitors on the majority of the instances from *REAL-WORLD-HARD*. Notably, *MetaVC2* was able to improve the best known solutions for 16 large MinVC instances, indicating that it can provide substantial benefits in practice. Moreover, *MetaVC2* also performs well on the other 2 benchmarks (*i.e., DIMACS-HARD* and *BHOSLIB-HARD*); additional experimental results for *MetaVC2* are available online.[2]

[10]In this work, during the enhanced automatic configuration process for automatically configuring *MetaVC* on the *REAL-WORLD-HARD* benchmark (which will result in *MetaVC2* for the *REAL-WORLD-HARD* benchmark), T, t, r and δ were set to 300, 30, 10 and 0.0001, respectively.

5 Conclusions

In this work, we first proposed a new, highly parametric local search framework for MinVC, dubbed *MetaVC*, which can be configured to instantiate many new MinVC solvers. After being automatically configured using an automatic algorithm configurator for various benchmarks, *MetaVC* was able to achieve state-of-the-art performance on medium-size hard MinVC instances and competitive performance on large instances. Then, to further improve the performance of *MetaVC* on large instances, we introduced a neural-network-based approach for identifying and terminating unpromising target algorithm runs during the configuration process. The enhanced configuration of *MetaVC*, dubbed *MetaVC2*, was able to improve the best known solutions for 16 large MinVC instances from the *REAL-WORLD-HARD* benchmark. Overall, our results indicate that *MetaVC* dramatically advances the state of the art in MinVC solving. The implementation of *MetaVC* and additional information are available at https://github.com/chuanluocs/MetaVC.

In future work, we plan to develop more sophisticated methods for identifying and terminating unpromising target algorithm runs, to further enhance automatic configuration.

Acknowledgements

Shaowei Cai is supported by the Youth Innovation Promotion Association, Chinese Academy of Sciences (No. 2017150).

References

[Akiba and Iwata, 2016] Takuya Akiba and Yoichi Iwata. Branch-and-reduce exponential/FPT algorithms in practice: A case study of vertex cover. *Theoretical Computer Science*, 609:211–225, 2016.

[Cai *et al.*, 2010] Shaowei Cai, Kaile Su, and Qingliang Chen. EWLS: A new local search for minimum vertex cover. In *Proc. of AAAI 2010*, pages 45–50, 2010.

[Cai *et al.*, 2013] Shaowei Cai, Kaile Su, Chuan Luo, and Abdul Sattar. NuMVC: An efficient local search algorithm for minimum vertex cover. *Journal of Artificial Intelligence Research*, 46:687–716, 2013.

[Cai *et al.*, 2015] Shaowei Cai, Jinkun Lin, and Kaile Su. Two weighting local search for minimum vertex cover. In *Proc. of AAAI 2015*, pages 1107–1113, 2015.

[Cai *et al.*, 2017] Shaowei Cai, Jinkun Lin, and Chuan Luo. Finding a small vertex cover in massive sparse graphs: Construct, local search, and preprocess. *Journal of Artificial Intelligence Research*, 59:463–494, 2017.

[Chang *et al.*, 2017] Lijun Chang, Wei Li, and Wenjie Zhang. Computing A near-maximum independent set in linear time by reducing-peeling. In *Proc. of SIGMOD 2017*, pages 1181–1196, 2017.

[Dinur and Safra, 2005] Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162:439–485, 2005.

[Domhan *et al.*, 2015] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proc. of IJCAI 2015*, pages 3460–3468, 2015.

[Fan *et al.*, 2019] Yi Fan, Yongxuan Lai, Chengqian Li, Nan Li, Zongjie Ma, Jun Zhou, Longin Jan Latecki, and Kaile Su. Efficient local search for minimum dominating sets in large graphs. In *Proc. of DASFAA 2019*, pages 211–228, 2019.

[Friedrich *et al.*, 2017] Tobias Friedrich, Timo Kötzing, and Markus Wagner. A generic bet-and-run strategy for speeding up stochastic local search. In *Proc. of AAAI 2017*, pages 801–807, 2017.

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[Hoos, 2012] Holger H. Hoos. Programming by optimization. *Communications of the ACM*, 55(2):70–80, 2012.

[Hutter *et al.*, 2002] Frank Hutter, Dave A. D. Tompkins, and Holger H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *Proc. of CP 2002*, pages 233–248, 2002.

[Hutter *et al.*, 2011] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proc. of LION 2011*, pages 507–523, 2011.

[Hutter *et al.*, 2017] Frank Hutter, Marius Lindauer, Adrian Balint, Sam Bayless, Holger H. Hoos, and Kevin Leyton-Brown. The configurable SAT solver challenge (CSSC). *Artificial Intelligence*, 243:1–25, 2017.

[Kavalci *et al.*, 2014] Vedat Kavalci, Aybars Ural, and Orhan Dagdeviren. Distributed vertex cover algorithms for wireless sensor networks. *International Journal of Computer Networks & Communications*, 6:95–110, 2014.

[Khalil *et al.*, 2017] Elias B. Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Proc. of NIPS 2017*, pages 6351–6361, 2017.

[KhudaBukhsh *et al.*, 2016] Ashiqur R. KhudaBukhsh, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. SATenstein: Automatically building local search SAT solvers from components. *Artificial Intelligence*, 232:20–42, 2016.

[Kingma and Ba, 2015] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. of ICLR 2015*, 2015.

[Lamm *et al.*, 2017] Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Finding near-optimal independent sets at scale. *Journal of Heuristics*, 23(4):207–229, 2017.

[Ma *et al.*, 2016a] Zongjie Ma, Yi Fan, Kaile Su, Chengqian Li, and Abdul Sattar. Local search with noisy strategy for minimum vertex cover in massive graphs. In *Proc. of PRICAI 2016*, pages 283–294, 2016.

[Ma *et al.*, 2016b] Zongjie Ma, Yi Fan, Kaile Su, Chengqian Li, and Abdul Sattar. Random walk in large real-world graphs for finding smaller vertex cover. In *Proc. of ICTAI 2016*, pages 686–690, 2016.

[Pullan and Hoos, 2006] Wayne J. Pullan and Holger H. Hoos. Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research*, 25:159–185, 2006.

[Richter *et al.*, 2007] Silvia Richter, Malte Helmert, and Charles Gretton. A stochastic local search approach to vertex cover. In *Proc. of KI 2007*, pages 412–426, 2007.

[Rossi and Ahmed, 2015] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proc. of AAAI 2015*, pages 4292–4293, 2015.

[Schmidhuber *et al.*, 2005] Jürgen Schmidhuber, Daan Wierstra, and Faustino J. Gomez. Evolino: Hybrid neuroevolution/optimal linear search for sequence learning. In *Proc. of IJCAI 2005*, pages 853–858, 2005.

[Svegliato *et al.*, 2018] Justin Svegliato, Kyle Hollins Wray, and Shlomo Zilberstein. Meta-level control of anytime algorithms with online performance prediction. In *Proc. of IJCAI 2018*, pages 1499–1505, 2018.

[Wagner *et al.*, 2017] Markus Wagner, Tobias Friedrich, and Marius Lindauer. Improving local search in a minimum vertex cover solver for classes of networks. In *Proc. of CEC 2017*, pages 1704–1711, 2017.

[Weise *et al.*, 2019] Thomas Weise, Zijun Wu, and Markus Wagner. An improved generic bet-and-run strategy with performance prediction for stochastic local search. In *Proc. of AAAI 2019*, 2019.

[Xie and Qin, 2018] Xiaojun Xie and Xiaolin Qin. Dynamic feature selection algorithm based on minimum vertex cover of hypergraph. In *Proc. of PAKDD 2018*, pages 40–51, 2018.

[Xu *et al.*, 2007] Ke Xu, Frédéric Boussemart, Fred Hemery, and Christophe Lecoutre. Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artificial Intelligence*, 171(8-9):514–534, 2007.

[Xu *et al.*, 2018] Hong Xu, Kexuan Sun, Sven Koenig, and T. K. Satish Kumar. A warning propagation-based linear-time-and-space algorithm for the minimum vertex cover problem on giant graphs. In *Proc. of CPAIOR 2018*, pages 567–584, 2018.