Local Search with Efficient Automatic Configuration for Minimum Vertex Cover

Chuan Luo1,2, Holger H. Hoos2,3, Shaowei Cai4, Qingwei Lin1, Hongyu Zhang5 and Dongmei Zhang1
1Microsoft Research, China
2Leiden Institute of Advanced Computer Science, Leiden University, The Netherlands
3Department of Computer Science, University of British Columbia, Canada
4State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
5The 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 32V-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 32V-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 $S \subseteq V$ 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_re then |
| 12 | $C := ReConstruct(C)$; |
| 13 | continue; |
| 14 | if performBMS then $S := filterBMS(C)$; |
| 15 | else $S := C$; |
| 16 | $v := pickRmVertex()$; |
| 17 | $C := C \setminus \{v\}$; |
| 18 | $e := pickUncovEdge()$; |
| 19 | $u := pickAddVertex(e)$; |
| 20 | $C := C \cup \{u\}$; |
| 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 Added

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].

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
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, which is taken from the the Second DIMACS Challenge Test Problems.

- **BHOSLIB-HARD**: The 15 hardest instances from the BHOSLIB benchmark, generated in the hardest area of model RB [Xu et al., 2007].

<table>
<thead>
<tr>
<th>Graphs</th>
<th>MetaVC</th>
<th>NuMVC</th>
<th>TwMVC</th>
<th>FastVC2+p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance Name</td>
<td>VC**</td>
<td>Succ. rate</td>
<td>Succ. rate</td>
<td>Succ. rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>time</td>
<td>time</td>
<td>time</td>
</tr>
<tr>
<td>frb56-25-1</td>
<td>1344*</td>
<td>871.65</td>
<td>731.73</td>
<td>1131.59</td>
</tr>
<tr>
<td>frb56-25-2</td>
<td>1344*</td>
<td>1005.04</td>
<td>1159.36</td>
<td>1401.79</td>
</tr>
<tr>
<td>frb56-25-3</td>
<td>1344*</td>
<td>140.65</td>
<td>173.62</td>
<td>178.85</td>
</tr>
<tr>
<td>frb56-25-4</td>
<td>1344*</td>
<td>57.10</td>
<td>76.58</td>
<td>68.98</td>
</tr>
<tr>
<td>frb56-25-5</td>
<td>1344*</td>
<td>28.75</td>
<td>39.80</td>
<td>44.11</td>
</tr>
<tr>
<td>frb59-26-1</td>
<td>1475*</td>
<td>1436.38</td>
<td>1557.70</td>
<td>1870.86</td>
</tr>
<tr>
<td>frb59-26-2</td>
<td>1475*</td>
<td>45%</td>
<td>37%</td>
<td>35%</td>
</tr>
<tr>
<td>frb59-26-3</td>
<td>1475*</td>
<td>2747.71</td>
<td>2926.69</td>
<td>2938.82</td>
</tr>
<tr>
<td>frb59-26-4</td>
<td>1475*</td>
<td>45%</td>
<td>37%</td>
<td>35%</td>
</tr>
<tr>
<td>frb59-26-5</td>
<td>1475*</td>
<td>1187.73</td>
<td>1666.28</td>
<td>2226.29</td>
</tr>
<tr>
<td>frb59-26-6</td>
<td>1475*</td>
<td>68.90</td>
<td>88.27</td>
<td>142.36</td>
</tr>
</tbody>
</table>

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, 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. 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.

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.

---

1http://ics.ioc.ac.cn/~caisw/Resource/DIMACS%20complementary%20graphs.tar.gz
2http://sites.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm
3http://ics.ioc.ac.cn/~caisw/Resource/realworld%20graphs.tar.gz
4As 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.
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/~cnsi/w/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-WORLD-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 families 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-

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.
Graphs MetaVC2 MetaVC NuMVC TwMVC FastVC2+p
Instance Name VC min (avg) time VC min (avg) time VC min (avg) time VC min (avg) time VC min (avg) time
inf-road-usa* 11527630 11525352 (11525847.64) N/A (N/A) N/A (N/A) N/A (N/A)
inf-road-NetCA* 1001065 999854 (999911.64) N/A (N/A) N/A (N/A) N/A (N/A)
sc-ldoor 856754 74685 (74690.37) N/A (N/A) N/A (N/A) N/A (N/A)
sc-pwtk* 207673 207674 (207675.33) N/A (N/A) N/A (N/A) N/A (N/A)
socfb-berkeley13 117246 116849 (116871.15) N/A (N/A) N/A (N/A) N/A (N/A)
socfb-Indiana 31161 31158 (31159.57) N/A (N/A) N/A (N/A) N/A (N/A)
socfb-Penn94* 28165 28166 (28170.56) N/A (N/A) N/A (N/A) N/A (N/A)
socfb-Texas48* 11261 11262 (11262.08) N/A (N/A) N/A (N/A) N/A (N/A)
socfb-UCSB37 27303 27304 (27305.56) N/A (N/A) N/A (N/A) N/A (N/A)
socfb-UI6* 27305 27306 (27307.91) N/A (N/A) N/A (N/A) N/A (N/A)
socfb-Ullinois* 24091 24092 24093 (24094.00) N/A (N/A) N/A (N/A) N/A (N/A)
socfb-Wisconsin87* 18382 18383 (18383.89) N/A (N/A) N/A (N/A) N/A (N/A)
tech-RL-caida 74959 74621 74624 (74625.39) N/A (N/A) N/A (N/A) N/A (N/A)
tech-skitter 525163 525183 (525196.02) 525186 (525200.99) N/A (N/A) N/A (N/A) N/A (N/A)

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 was 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 \(y_{i:p}\), 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 \(y_i\), 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
Table 5: The optimized configurations of MetaVC for all benchmarks.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Optimized Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>brock-HARD</td>
<td>performPreProcess=True, initConstruct=2, performReConstruct=True, prob_rec=0.0028421872317207584, t=100, performBMS=False, pickRmVertex=1, performEdgeWeight=True, edgeWeight=1</td>
</tr>
<tr>
<td>DIMACS-HARD</td>
<td>pickUncorEdges=2, pickAddVertex=1, performBMS=True, performEdgeWeight=True, edgeWeight=1</td>
</tr>
<tr>
<td>BHOSLIB-HARD</td>
<td>performPreProcess=True, initConstruct=2, performReConstruct=False, performBMS=False, pickRmVertex=2, pickAddVertex=1, performEdgeWeight=True, edgeWeight=1</td>
</tr>
<tr>
<td>REAL-WORLD-HARD</td>
<td>performPreProcess=True, initConstruct=2, performReConstruct=True, prob_rec=3.90865983029911E-5, t=84, performBMS=True, bms_k=633, pickRmVertex=2, pickUncorEdges=1, pickAddVertex=2, performEdgeWeight=False</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Optimized Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL-WORLD-HARD</td>
<td>performPreProcess=True, initConstruct=2, performReConstruct=True, prob_rec=0.0886947578801404E-3, t=100, performBMS=True, bms_k=720, pickRmVertex=1, pickUncorEdges=1, pickAddVertex=2, performEdgeWeight=False</td>
</tr>
</tbody>
</table>

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.\(^2\)

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/chuanluos/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


\(^{10}\)In this work, during the enhanced automatic configuration process for 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.


