1 Overview of Benchmarks

In this experiment, we consider 3 prominent benchmarks *DIMACS*[1], *BHOSLIB*[2] and *REAL-WORLD*[3], which have been widely tested in previous work on MinVC [Cai *et al.*, 2015; Ma *et al.*, 2016a; Ma *et al.*, 2016b; Cai *et al.*, 2017; Wagner *et al.*, 2017; Gao *et al.*, 2017]. We note that all benchmarks can be directly downloaded at **https://drive.google.com/uc?id=1oPV92AxJi0W8ntyuNOPEfbT23OLzE7wZ&export=download**.

1.1 Details of Benchmarks

Since the primary goal of this work is to push forward the state of the art, we would like to evaluate *MetaVC* on the most difficult instances. We would like to note that the results of *MetaVC* on all the remaining easy instances are reported online.[4] Moreover, *NuMVC* [Cai *et al.*, 2013] and *TwMVC* [Cai *et al.*, 2015] are the current best solvers for solving instances from *DIMACS* and *BHOSLIB*, while *FastVC2+p* [Cai *et al.*, 2017] is the current best solver for solving instances from *REAL-WORLD*. Therefore, we use *NuMVC* and *TwMVC* to identify the hardest instances from *DIMACS* and *BHOSLIB*, and use *FastVC2+p* to identify the hardest instances from *REAL-WORLD*.

- **DIMACS-HARD:** 8 hardest instances from the *DIMACS* benchmark, which is taken from the the Second DIMACS Challenge Test Problems.
- **BHOSLIB-HARD:** 15 hardest instances from the *BHOSLIB* benchmark, which is generated in the hardest area by the model RB [Xu *et al.*, 2007].
- **REAL-WORLD-HARD:** 31 hardest application instances from the *REAL-WORLD* benchmark, which collects all undirected simple graphs (not including *DIMACS* and *BHOSLIB* graphs) from the Network Data Repository [Rossi and Ahmed, 2015].

DIMACS-HARD: As reported in the literature [Cai *et al.*, 2015], most instances in the original *DIMACS* benchmark are very easy to be solved. Since the instance family of 'brock' is much harder than other families, we thus include 4 hard 'brock' instances ('brock400_2', 'brock400_4', 'brock800_2' and 'brock800_4') into *DIMACS-HARD*.[5] In addition, we include 4 instances ('C2000.9', 'C4000.5', 'MANN_a45' and 'MANN_a81') into *DIMACS-HARD*.[6] All other instances are included in *DIMACS-EASY*.

BHOSLIB-HARD: As reported in the literature [Cai *et al.*, 2015], a great number of instances in the original *BHOSLIB* benchmarks are very easy to be solved. Since the instance families of 'frb53-24', 'frb56-25' and 'frb59-26' are much harder than other families,[7] We thus include those 15 instances in instance families of 'frb53-24', 'frb56-25' and 'frb59-26' into *BHOSLIB-HARD*. All other instances are included in *BHOSLIB-EASY*.

REAL-WORLD-HARD: According to the experimental results reported in the literature [Cai *et al.*, 2017], from the original *REAL-WORLD* benchmark, we include 31 instances, where the average solution quality found by *FastVC2+p* is worse than the best solution quality found by *FastVC2+p* or *FastVC2+p* does not perform best in the comparison reported in the literature [Cai *et al.*, 2017], into *REAL-WORLD-HARD*. All other instances are included in *REAL-WORLD-EASY*.

1.2 Training Sets of Benchmarks

Training set for the instance family of 'brock' in *DIMACS-HARD*: As reported in the literature [Cai *et al.*, 2015], for *DIMACS-HARD*, the configuration used for the *brock* instances is different from the ones used for the *C* and *MANN* instances. Hence, we separately configured *MetaVC* on *brock* instances. We randomly select 2 instances entitled 'brock400_4' and 'brock800_4' as the training instances.

Training set for *DIMACS-HARD*: As reported in the literature [Cai *et al.*, 2015], the instance 'MANN_a45' is easier to be solved than the 3 others (i.e., 'C2000.9', 'C4000.5' and 'MANN_81'). We select the easiest instance 'MANN_a45' as the training instance.

Training set for *BHOSLIB-HARD*: As reported in the literature [Cai *et al.*, 2015], the instance family of 'frb53-24' are easier to be solved than the 2 others (i.e., the instance families of 'frb56-25' and 'frb59-26'). We select the 5 instance in the easiest instance family 'frb53-24' as the training instances.

Training set for *REAL-WORLD-HARD*: According to the experimental results in the literature [Cai *et al.*, 2017], we select 12 instances, which the averaged run of *FastVC2+p* for solving is less than 300 seconds, as the training set.[8] The training instances for *REAL-WORLD-HEAD* are indicated in *italics* font (on the upside part) in Tables 3 and 4 in the submitted paper.

References

[Cai *et al.*, 2013] Shaowei Cai, Kaile Su, Chuan Luo, and Abdul Sattar. NuMVC: An efficient local search algorithm

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

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

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

[4] https://github.com/metavc/MetaVC

[5] We do not include 'brock200_2' and 'brock200_4', which can be solved by *NuMVC* and *TwMVC* with the success rate of 100% and the averaged time in less than 1 second.

[6] We do not include other instances, because all of them can be solved by *NuMVC* and *TwMVC* with the success rate of 100% and the averaged time in less than 10 seconds.

[7] For instance family of 'frb53-24', 1 instance cannot be solved by *NuMVC* and *TwMVC* with the success rate of 100%. For instance family of 'frb56-25', 2 instances cannot be solved by *NuMVC* and *TwMVC* with the success rate of 100%. For instance family of 'frb59-26', 4 instances cannot be solved by *NuMVC* and *TwMVC* with the success rate of 100%. For other instance families, all instances can be solved by *NuMVC* and *TwMVC* with the success rate of 100%.

[8] According to the experimental results in the literature [Cai *et al.*, 2017], for the other instances in *REAL-WORLD-HARD*, the averaged run of *FastVC2+p* for solving them is more than 300 seconds.

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.

[Gao *et al.*, 2017] Wanru Gao, Tobias Friedrich, Timo Kötzing, and Frank Neumann. Scaling up local search for minimum vertex cover in large graphs by parallel kernelization. In *Proc. of AI 2017*, pages 131–143, 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.

[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.

[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.

[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.