WeGotYouCovered

Demian Hespe* Sebastian Lamm† Christian Schulz‡ Darren Strash§

Abstract

We present the winning solver of the PACE 2019 Implementation Challenge Vertex Cover Track. The vertex cover problem is one of a handful of problems for which *kernelization*—the repeated reducing of the input size via *data reduction rules*—is known to be highly effective in practice. Our algorithm uses a portfolio of techniques, including an aggressive kernelization strategy with all known reduction rules, local search, branch-and-reduce, and a state-of-the-art branch-and-bound solver. Of particular interest is that several of our techniques were *not* from the literature on the vertex over problem: they were originally published to solve the (complementary) maximum independent set and maximum clique problems. Lastly, we perform extensive experiments to show the impact of the different solver techniques on the number of instances solved during the challenge.

1 Introduction

A *vertex cover* of a graph $G = (V, E)$ is a set of vertices $S \subseteq V$ of G such that every edge of G has at least one of member of S as an endpoint (i.e., $\forall (u, v) \in E$ $[u \in S$ or $v \in S]$). A minimum vertex cover is a vertex cover of minimum cardinality. Complementary to vertex covers are independent sets and cliques. An independent set is a set of vertices $I \subseteq V$, all pairs of which are not adjacent, and an clique is a set of vertices $K \subseteq V$ all pairs of which are adjacent. A maximum independent set (maximum clique) is an independent set (clique) of maximum cardinality. The goal of the maximum independent set problem (maximum clique problem) is to compute a maximum independent set (maximum clique).

Many techniques have been proposed for solving these problems, and papers in the literature usually focus on one of these problems in particular. However, all of these problems are equivalent: a minimum vertex cover C in G is the complement of a maximum independent set $V \setminus C$ in G, which is a maximum clique $V \setminus C$ in $\overline{G}$. Thus, an algorithm that solves one of these problems can be used to solve the others. For our approach, we use a portfolio of solvers, using techniques from the literature on all three problems. These include data reduction rules and branch-and-reduce for the minimum vertex cover problem [2], iterated local search for the maximum independent set problem [3], and a state-of-the-art branch-and-bound maximum clique solver [14].

We first briefly describe releated work. Then we outline each of the techniques that we use, and finally describe how we combine all of the techniques in our final solver that scored most of the points during the PACE 2019 Implementation Challenge. Lastly, we perform an experimental evaluation to show the impact of the components used on the final number of instances solved during the challenge.

2 Related Work

We now present important related work. This includes exact branch-and-bound algorithms as well as reduction based approaches. Much research has been devoted to improve exact branch-and-bound algorithms for the independent set and its complementary problems. These improvements include different pruning methods and sophisticated branching schemes [16, 5, 4, 18]. Warren and Hicks [18] proposed three combinatorial branch-and-bound algorithms that are able to quickly solve DIMACS and weighted random graphs. These algorithms use weighted clique covers to generate upper bounds that reduce the search space via pruning. Furthermore, they all use a branching scheme proposed by Balas and Yu [5]. In particular, their first algorithm is an extension and improvement of a method by Babel [4]. Their second one uses a modified version of the algorithm by Balas and Yu that uses clique covers that borrow structural features from the ones by Babel [4]. Finally, their third approach is a hybrid of both previous algorithms. Overall, their algorithms are able to quickly solve instances with hundreds of vertices.

An important technique to reduce the base of the exponent for exact branch-and-bound algorithms are so-called *reduction rules*. Reduction rules are able to reduce the input graph to an irreducible *kernel* by removing well-defined subgraphs. This is done by

*Karlsruhe Institute of Technology, Karlsruhe, Germany
†Karlsruhe Institute of Technology, Karlsruhe, Germany
‡University of Vienna, Faculty of Computer Science, Austria
§Hamilton College, New York, USA

selecting certain vertices that are provably part of some maximum independent set, thus maintaining optimality. We can then extend a solution on the kernel to a solution on the input graph by undoing the previously applied reductions. There exist several well-known reduction rules for the unweighted vertex cover problem (and in turn for the unweighted MIS problem) [2].

As noted by Larson [13], it is possible that in the unweighted case the initial critical set found by Butenko and Trukhanov might be empty. To prevent this case, Larson [13] proposed an algorithm that finds a *maximum* (unweighted) critical independent set. His algorithm accumulates vertices that are in some critical set and removes their neighborhood. Additionally, he provides a method to quickly check if a given vertex is part of some critical set. Later Iwata [12] has shown how to remove a large collection of vertices from a maximum matching all at once; however, it is not known if these reductions are equivalent.

For the maximum weight clique problem, Cai and Lin [8] give an exact branch-and-bound algorithm that interleaves between clique construction and reductions. In particular, their algorithm picks different starting vertices to form a clique and then maintains a candidate set to iteratively extend this clique. In each iteration, the vertex to be added is selected using a benefit estimation function and a dynamic best from multiple selection heuristic [7]. Once the candidate set is empty, the new solution is compared to the best solution found so far. If an improvement is found, their algorithm then tries to apply reductions and reduce the graph size. To be more specific, they use two reduction rules that are able to remove a vertex v by computing upper bounds related to the weight of different neighborhoods of v. We briefly note that their algorithm and reductions are targeted at sparse graphs, and therefore their reductions would likely work well for the maximum weighted independent set problem on *dense* graphs.

3 Solver Techniques

We now describe the main techniques that we use within our solver.

Kernelization. The most efficient algorithms for computing a minimum vertex cover in both theory and practice use *data reduction rules* to obtain a much smaller problem instance. If this smaller instance has size bounded by a function of some parameter, it's called a *kernel*.

We use an extensive (though not exhaustive) collection of data reduction rules whose efficacy was studied by Akiba and Iwata [2]. To compute a kernel, Akiba and Iwata [2] apply their reductions $r_1, \ldots, r_j$ by iterating over all reductions and trying to apply the current reduction r_i to all vertices. If r_i reduces at least one vertex, they restart with reduction r_1. When reduction r_j is executed, but does not reduce any vertex, all reductions have been applied exhaustively, and a kernel is found. Following their study we order the reductions as follows: degree-one vertex (i.e., pendant) removal, unconfined vertex removal [19], a well-known linear-programming relaxation [12, 15] related to crown removal [1], vertex folding [10], and twin, funnel, and desk reductions [19].

Branch-and-Reduce. Branch-and-reduce is a paradigm that intermixes data reduction rules and branching. We use the algorithm of Akiba and Iwata, which exhaustively applies their full suite of reduction rules before branching, and includes a number of advanced branching rules. When branching, a vertex is chosen at random for inclusion into the vertex cover.

Branch-and-Bound. Experiments by Strash [17] show that the full power of branch-and-reduce is only needed *very rarely* in real-world instances; kernelization followed by standard branch-and-bound solver is sufficient for many real-world instances. Furthermore, branch-and-reduce does not work well on many synthetic benchmark instances, where data reduction rules are ineffective [2], and instead add significant overhead to branch-and-bound. We use a state-of-the-art branch-and-bound maximum clique solver (MoMC) by Li et al. [14], which uses incremental MaxSAT reasoning to prune search, and a combination of static and dynamic vertex ordering to select the vertex for branching. We run the clique solver on the complement graph, giving a maximum independent set from which we derive a minimum vertex cover. In preliminary experiments, we found that a kernel can sometimes be harder for the solver than the original input; therefore, we run the algorithm on both the kernel and on the original graph.

Iterated Local Search. Batsyn et al. [6] showed that if branch-and-bound search is primed with a high-quality solution from local search, then instances can be solved up to thousands of times faster. We use iterated local search algorithm by Andrade et al. [3] to prime the branch-and-reduce solver with a high-quality initial solution. Iterated local search was originally implemented for the maximum independent set problem, and is based on the notion of (j, k)-swaps. A (j, k)-swap removes j nodes from the current solution and inserts k nodes. The authors present a fast linear-time implementation that, given a maximal independent set, can find a $(1, 2)$-swap or prove that none exists. Their algorithm applies $(1, 2)$-swaps until reaching a local maximum, then perturbs the solution and repeats. We implemented the algorithm to find a high-quality solution on *the kernel*. Calling local search on the kernel has been shown to pro-

duce a high-quality solution much faster than without kernelization [9, 11].

4 Putting it all Together

Our algorithm first runs a preprocessing phase, followed by 4 phases of solvers.

Phase 1. (Preprocessing) Our algorithm starts by computing a kernel of the graph using the reductions by Akiba and Iwata [2]. From there we use iterated local search to produce a high-quality solution S_{init} on the (hopefully smaller) kernel.

Phase 2. (Branch-and-Reduce, short) We prime a branch-and-reduce solver with the initial solution S_{init} and run it with a short time limit.

Phase 3. (Branch-and-Bound, short) If Phase 2 is unsuccessful, we run the MoMC [14] clique solver on the complement of the kernel, also using a short time limit. Sometimes kernelization can make the problem harder for MoMC. Therefore, if the first call was unsuccessful we also run MoMC on the complement of the original (unkernelized) input with the same short time limit.

Phase 4. (Branch-and-Reduce, long) If we have still not found a solution, we run branch-and-reduce on the kernel using initial solution S_{init} and a longer time limit. We opt for this second phase because, while most graphs amenable to reductions are solved very quickly with branch-and-reduce (less than a second), experiments by Akiba and Iwata [2] showed that other slower instances either finish in at most a few minutes, or take significantly longer—more than the time limit allotted for the challenge. This second phase of branch-and-reduce is meant to catch any instances that still benefit from reductions.

Phase 5. (Branch-and-Bound, remaining time) If all previous phases were unsuccessful, we run MoMC on the original (unkernelized) input graph until the end of the time given to the program by the challenge. This is meant to capture only the most hard-to-compute instances.

The ordering and time limits were carefully chosen so that the overall algorithm outputs solutions of the "easy" instances *quickly*, while still being able to solve hard instances.

5 Experimental Results

We now look at the impact of the algorithmic components on the number of instances solved. Here, we use the public instances – obtained from `https://pacechallenge.org/files/pace2019-vc-exact-public-v2.tar.bz2` – of the PACE 2019 Track A implementation challenge. This set contains 100 instances overall. Afterwards, we present the results comaring against the second and third best competing algorithms during the challenge.

5.1 Methodology and Setup. All of our experiments were run on a machine with four Sixteen-Core Intel Xeon Haswell-EX E7-8867 processors running at 2.5 GHz, 1 TB of main memory, and 32768 KB L2-Cache. The machine runs Debian GNU/Linux 9 and Linux kernel version 4.9.0-9. All algorithms were implemented in C++11 and compiled with gcc version 6.3.0 with optimization flag `-O3`. Each algorithm was run sequentially with a time limit of 30 minutes. Our evaluations focus on the total number of instances solved.

5.2 Algorithm Configurations. Variants:

- MoMC solves 30 / 100 instances
- Kernel + MoMC solves 68 / 100 instances
- Kernel + BnR solves 55 / 100 instances
- Kernel + BnR - Local Search solves 42 / 100 instances
- Full solver solves 82 / 100 instances

6 TODOs

- TODO add more exact stuff, heuristics?
- TODO insert final three solvers of pace challenge
- create a table with instances solved

References

[1] Faisal N. Abu-Khzam, Michael R. Fellows, Michael A. Langston, and W. Henry Suters. Crown structures for vertex cover kernelization. *Theor. Comput. Syst.*, 41(3):411–430, 2007. `doi:10.1007/s00224-007-1328-0`.

[2] T. Akiba and Y. Iwata. Branch-and-reduce exponential/FPT algorithms in practice: A case study of vertex cover. *Theor. Comput. Sci.*, 609, Part 1:211–225, 2016. `doi:10.1016/j.tcs.2015.09.023`.

inst #	n	m	MoMC	RMoMC	BnR	BnR-LS	FullA
001	6 160	40 207	-	X	X	X	X
003	60 541	74 220	-	X	X	X	X
005	200	819	X	X	X	X	X
007	8 794	10 130	-	X	X	X	X
009	38 452	174 645	-	X	X	X	X
011	9 877	25 973	-	X	X	X	X
013	45 307	55 440	-	X	X	X	X
015	53 610	65 952	-	X	X	X	X
017	23 541	51 747	-	X	X	X	X
019	200	884	X	X	X	X	X
021	24 765	30 242	-	X	X	X	X
023	27 717	133 665	-	X	X	X	X
025	23 194	28 221	-	X	X	X	X
027	65 866	81 245	-	X	X	X	X
029	13 431	21 999	-	X	X	X	X
031	200	813	X	X	X	X	X
033	4 410	6 885	-	X	X	X	X
035	200	884	X	X	X	X	X
037	198	824	X	X	X	X	X
039	6 795	10 620	-	X	X	X	X
041	200	1 040	X	X	X	X	X
043	200	841	X	X	X	X	X
045	200	1 044	X	X	X	X	X
047	200	1 120	X	X	X	X	X
049	200	957	X	X	X	X	X
051	200	1 135	X	X	X	X	X
053	200	1 062	X	X	X	X	X
055	200	958	X	X	X	X	X
057	200	1 200	X	X	X	X	X
059	200	988	X	X	X	X	X
061	200	952	X	X	X	X	X
063	200	1 040	X	X	X	X	X
065	200	1 037	X	X	X	X	X
067	200	1 201	X	X	X	X	X
069	200	1 120	X	X	X	X	X
071	200	984	X	X	X	X	X
073	200	1 107	X	X	X	X	X
075	26 300	41 500	-	-	X	-	X
077	200	988	X	X	X	X	X
079	26 300	41 500	-	-	X	-	X
081	199	1 124	X	X	X	X	X
083	200	1 215	X	X	X	X	X
085	11 470	17 408	-	-	-	-	-
087	13 590	21 240	-	X	-	-	X
089	57 316	77 978	-	-	-	-	-
091	200	1 196	X	X	X	X	X
093	200	1 207	X	X	X	X	X
095	15 783	24 663	-	X	-	-	X
097	18 096	28 281	-	X	-	-	X
099	26 300	41 500	-	-	X	-	X

inst#	n	m	MoMC	RMoMC	BnR	BnR-LS	FullA
101	26 300	41 500	-	-	X	-	X
103	15 783	24 663	-	X	-	-	X
105	26 300	41 500	-	-	X	-	X
107	13 590	21 240	-	X	-	-	X
109	66 992	90 970	-	-	-	-	-
111	450	17 831	X	X	-	-	X
113	26 300	41 500	-	-	X	-	X
115	18 096	28 281	-	X	-	-	X
117	18 096	28 281	-	X	-	-	X
119	18 096	28 281	-	X	-	-	X
121	18 096	28 281	-	X	-	-	X
123	26 300	41 500	-	-	X	-	X
125	26 300	41 500	-	-	X	-	X
127	18 096	28 281	-	X	-	-	X
129	15 783	24 663	-	X	-	-	X
131	2 980	5 360	X	-	-	-	X
133	15 783	24 663	-	X	-	-	X
135	26 300	41 500	-	-	X	-	X
137	26 300	41 500	-	-	X	-	X
139	18 096	28 281	-	X	-	-	X
141	18 096	28 281	-	X	-	-	X
143	18 096	28 281	-	X	-	-	X
145	18 096	28 281	-	X	-	-	X
147	18 096	28 281	-	X	-	-	X
149	26 300	41 500	-	-	X	-	X
151	15 783	24 663	-	X	-	-	X
153	29 076	45 570	-	-	-	-	-
155	26 300	41 500	-	-	X	-	X
157	2 980	5 360	X	-	-	-	X
159	18 096	28 281	-	X	-	-	X
161	138 141	227 241	-	-	-	-	-
163	18 096	28 281	-	X	-	-	X
165	18 096	28 281	-	X	-	-	X
167	15 783	24 663	-	X	-	-	X
169	4 768	8 576	-	-	-	-	-
171	18 096	28 281	-	X	-	-	X
173	56 860	77 264	-	-	-	-	-
175	3 523	6 446	-	-	-	-	-
177	5 066	9 112	-	-	-	-	-
179	15 783	24 663	-	X	-	-	X
181	18 096	28 281	-	X	X	-	X
183	72 420	118 362	-	-	-	-	-
185	3 523	6 446	-	-	-	-	-
187	4 227	7 734	-	-	-	-	-
189	7 400	13 600	-	-	-	-	-
191	4 579	8 378	-	-	-	-	-
193	7 030	12 920	-	-	-	-	-
195	1 150	81 068	-	-	-	-	-
197	1 534	127 011	-	-	-	-	-
199	1 534	126 163	-	-	-	-	-

[3] D. V. Andrade, M. G.C. Resende, and R. F. Werneck. Fast local search for the maximum independent set problem. *Journal of Heuristics*, 18(4):525–547, 2012. doi:10.1007/s10732-012-9196-4.

[4] Luitpold Babel. A fast algorithm for the maximum weight clique problem. *Computing*, 52(1):31–38, 1994.

[5] Egon Balas and Chang Sung Yu. Finding a maximum clique in an arbitrary graph. *SIAM Journal on Computing*, 15(4):1054–1068, 1986.

[6] M. Batsyn, B. Goldengorin, E. Maslov, and P. Pardalos. Improvements to MCS algorithm for the maximum clique problem. *J. Comb. Optim.*, 27(2):397–416, 2014. doi:10.1007/s10878-012-9592-6.

[7] Shaowei Cai. Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[8] Shaowei Cai and Jinkun Lin. Fast solving maximum weight clique problem in massive graphs. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 568–574. AAAI Press, 2016. URL: http://www.ijcai.org/Proceedings/16/Papers/087.pdf.

[9] Lijun Chang, Wei Li, and Wenjie Zhang. Computing a near-maximum independent set in linear time by reducing-peeling. In *Proc. 2017 ACM International Conference on Management of Data (SIGMOD 2017)*, pages 1181–1196. ACM, 2017. doi:10.1145/3035918.3035939.

[10] Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *Journal of Algorithms*, 41(2):280–301, 2001. doi:10.1006/jagm.2001.1186.

[11] Jakob Dahlum, Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Accelerating local search for the maximum independent set problem. In *Proc. 15th International Symposium on Experimental Algorithms (SEA 2016)*, volume 9685 of *LNCS*, pages 118–133. Springer, 2016. doi:10.1007/978-3-319-38851-9_9.

[12] Y. Iwata, K. Oka, and Y. Yoshida. Linear-time FPT Algorithms via Network Flow. In *Proc. 25th ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 1749–1761. SIAM, 2014. URL: http://dl.acm.org/citation.cfm?id=2634074.2634201.

[13] C.E. Larson. A note on critical independence reductions. volume 51 of *Bulletin of the Institute of Combinatorics and its Applications*, pages 34–46, 2007.

[14] C.-M. Li, H. Jiang, and F. Manyà. On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem. *Computers & Operations Research*, 84:1–15, 2017. doi:10.1016/j.cor.2017.02.017.

[15] G.L. Nemhauser and L. E. Trotter Jr. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8(1):232–248, 1975. doi:10.1007/BF01580444.

[16] Patric RJ Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120(1-3):197–207, 2002.

[17] Darren Strash. On the power of simple reductions for the maximum independent set problem. In *Proc. 22nd International Computing and Combinatorics Conference (COCOON 2016)*, volume 9797 of *LNCS*, pages 345–356. Springer, 2016. doi:10.1007/978-3-319-42634-1_28.

[18] Jeffrey S Warren and Illya V Hicks. Combinatorial branch-and-bound for the maximum weight independent set problem. 2006. URL: https://www.caam.rice.edu/~ivhicks/jeff.rev.pdf.

[19] M. Xiao and H. Nagamochi. Confining sets and avoiding bottleneck cases: A simple maximum independent set algorithm in degree-3 graphs. *Theor. Comput. Sci.*, 469:92–104, 2013. doi:10.1016/j.tcs.2012.09.022.