We Got You Covered

Demian Hespe[1], Sebastian Lamm[2], Christian Schulz[3], and Darren Strash[4]

1 **Karlsruhe Institute of Technology, Karlsruhe, Germany**
hespe@kit.edu
2 **Karlsruhe Institute of Technology, Karlsruhe, Germany**
lamm@kit.edu
3 **University of Vienna, Faculty of Computer Science, Vienna, Austria**
christian.schulz@univie.ac.at
4 **Hamilton College, New York, USA,** dstrash@hamilton.edu

Abstract

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. For our submission, we use 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.

1998 ACM Subject Classification G.2.2 Graph Theory – Graph Algorithms, G.4 Mathematical Software – Algorithm Design and Analysis

Keywords and phrases kernelization, branch-and-reduce, local search

Digital Object Identifier 10.4230/LIPIcs.PACE.2019.

1 Solver Overview

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 including data reduction rules and branch-and-reduce for the minimum vertex cover problem [1], reduction rule optimizations for the maximum independent set problem [3, 4], iterated local search for the maximum independent set problem [2], and a branch-and-bound maximum clique solver [5]. For the sake of simplicity, we *write this report from the independent set perspective.*

We first briefly describe each of the techniques that we use, and then describe how we combine all of the techniques in our final solver.

2 Techniques

Kernelization

The most efficient algorithms for computing a maximum independent set 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 all known reduction rules for the problem: an extensive suite of reductions tested by Akiba and Iwata [1], the isolated clique reduction due to Butenko et al. [], and linear-time reductions for quick removal of paths and cycles due to Chang et al. [3]. We implement an additional technique to accelerate kernelization: dependency checking to prune reductions that cannot be applied [4]. More precisely, to compute a kernel, Akiba and Iwata [1] apply their reductions $r_1, \dots, r_j$ 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. Trying to apply every reduction to all vertices can be expensive in later stages of the algorithm where few reductions succeed. In [4], we define a scheme for checking dependencies between reductions, which allows us to avoid applying isolated vertex removal, vertex folding, and twin reductions when they will provably not succeed. After unsuccessfully trying to apply one of these reductions to a vertex v, one only has to consider v again for reduction after its neighborhood has changed. We therefore keep a set D of *viable* candidate vertices: vertices whose neighborhood has changed and vertices that have never been considered for reductions.

Branch-and-Reduce

Akiba and Iwata [1] further explored branch-and-reduce, which 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. We briefly note that their reductions do not include dependency checking or other optimizations. When branching, a vertex is chosen at random for inclusion into the vertex cover.

Branch-and-Bound

Experiments by Strash [] show that the full power of branch-and-reduce is only needed *very rarely*; 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 benchmark instances, where data reduction rules are ineffective [1], and instead add significant overhead to branch-and-bound. We use the state-of-the-art maximum clique solver (MoMC) by Li et al. [], which uses MaxSAT reasoning to prune search, and a combination of static and dynamic vertex ordering to select the vertex for branching. We briefly note that we consider running it on the kernel, and on the original graph. In preliminary experiments, we found that a kernel can somtimes be harder for the solver than the original input.

Iterated Local Search

Batsyn et al. [] 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 therefore use local search to prime branch-and-reduce and branch-and-bound solvers. We use iterated local search algorithm by Andrade et al. [2] for this purpose. The algorithm 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 solution, can find a $(1, 2)$-swap or prove that none exists. Their algorithm applies $(1, 2)$-swaps until reaching a local maximum, 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 produce a high-quality solution much faster than without kernelization [**?**, 3].

3 Putting it all Together

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

Phase 1. (Preprocessing) Our algorithm starts by kernelizing the graph using [4]. From there we use local search to produce a high-quality solution C_{init} on the (hopefully smaller) kernel.

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

Phase 3. (Branch-and-Bound, short) If Phase 2 is unsuccessful, we run the MoMC [5] clique solver on the complement of the kernel, also using a small time limit. Sometimes kernelization can make the problem harder for MoMC. If the first call was unsuccessful we also run MoMC on the complement of the input with the same 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 a longer time limit.

Phase 5. (Branch-and-Bound, remaining time) If all previous phases were unsuccessful, we run MoMC until the end of the time given to the program by the challenge.

Note the way we integrated the different solvers is such that the overall algorithm will output solutions of the "easy" instances *quickly*, while still being able to solve hard instances.

4 Material

GitHub: `https://github.com/sebalamm/pace-2019/releases/tag/pace-2019`
DOI: `https://doi.org/10.5281/zenodo.2816116`

References

1 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.

2 D. V. Andrade, M. G. C. Resende, and Renato F. Werneck. Fast local search for the maximum independent set problem. In Catherine C. McGeoch, editor, *Experimental Algorithms, 7th International Workshop, WEA 2008, Provincetown, MA, USA, May 30-June 1, 2008, Proceedings*, volume 5038 of *Lecture Notes in Computer Science*, pages 220–234. Springer, 2008.

3 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.

4 D. Hespe, C. Schulz, and D. Strash. Scalable kernelization for maximum independent sets. In *Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments, ALENEX 2018, New Orleans, LA, USA, January 7-8, 2018.*, pages 223–237, 2018.

5 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.