# WeGotYouCovered: The Winning Solver from the PACE 2019 Implementation Challenge, Vertex Cover Track*

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

**Abstract**

We present the winning solver of the PACE 2019 Implementation Challenge, Vertex Cover Track. The minimum 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, 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.

Aside from illustrating our solver's performance in the PACE 2019 Implementation Challenge, our experiments provide several key insights not yet seen before in the literature. First, kernelization can boost the performance of branch-and-bound clique solvers enough to outperform branch-and-reduce solvers. Second, local search can significantly boost the performance of branch-and-reduce solvers. And finally, somewhat surprisingly, kernelization can sometimes make branch-and-bound algorithms perform *worse* than running branch-and-bound alone.

## 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 $v \in S$]). The minimum vertex cover problem—that of computing a vertex cover of minimum cardinality—is a fundamental NP-hard problem, and has applications spanning many areas. These include computational biology [12], classification [20], mesh rendering [35], and many more through its complementary problems [19, 18, 21, 45].

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. To win the PACE 2019 Implementation Challenge, we deployed 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 [28].

**Our Results.** In this paper, we describe our techniques and solver in detail and analyze the results of our experiments on the data sets provided by the challenge. Not only do our experiments illustrate the power of the techniques spanning the literature, they also provide several new insights not yet seen before. In particular, kernelization followed by branch-and-bound can outperform branch-and-reduce solvers; seeding branch-and-reduce by an initial solution from local search can significantly boost its performance; and, somewhat surprisingly, kernelization is sometimes counterproductive: branch-and-bound algorithms can perform significantly worse on the kernel than on the original input graph.

**Organization.** We first briefly describe related work in Section 3. Then in Section 4 we outline each of the techniques that we use, and in Section 5 finally describe how we combine all of the techniques in our

*The research leading to these results has received funding from the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013) /ERC grant agreement No. 340506

†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

final solver that scored the most points in the PACE 2019 Implementation Challenge. Lastly, in Section 6 we perform an experimental evaluation to show the impact of the components used on the final number of instances solved during the challenge.

## 2 Preliminaries

We work with an undirected graph $G = (V, E)$ where $V$ is a set of $n$ vertices and $E \subset \{\{u, v\} \mid u, v \in V\}$ is a set of $m$ edges. The open neighborhood of a vertex $v$, denoted $N(v)$, is the set of all vertices $w$ such that $(v, w) \in E$. We further denote the closed neighborhood by $N[v] = N(v) \cup \{v\}$. We similarly define the open and closed neighborhoods of a set of vertices $U$ to be $N(U) = \bigcup_{u \in U} N(u)$ and $N[U] = N(U) \cup U$, respectively. The set of vertices of distance $d$ of a vertex $u$ is denoted by $N^d(u)$, where $N^2(u)$ is called the *two-neighborhood* of $u$. Lastly, for vertices $S \subseteq V$, the induced subgraph $G[S] \subseteq G$ is the graph on the vertices in $S$ with edges in $E$ between vertices in $S$.

## 3 Related Work

Research results in the area can be found through work on the minimum vertex cover problem and its complementary maximum clique and independent set problems, and can often be categorized depending on the angle of attack. For exact exponential (theoretical) algorithms, the maximum independent set problem is canonically studied, for parameterized algorithms, the minimum vertex cover problem is studied, and the maximum clique problem is normally solved exactly in practice (though there are recent exceptions). However, these problems are only *trivially* different — techniques for solving one problem require only subtle modifications to solve the other two.

**Exponential-time Algorithms.** The maximum independent set problem is most often considered when designing exact (exponential-time) algorithms, and much research has be devoted to reducing the base of the exponential running time. A primary technique is to develop rules to modify the graph, removing or contracting subgraphs that can be solved simply, which reduces the graph to a smaller instance. These rules are referred to as *data reduction rules* (often simplified to *reduction rules* or *reductions*). Reduction rules have been used to reduce the running time of the brute force $O(n^2 2^n)$ algorithm to the $O(2^{n/3})$ time algorithm of Tarjan and Trojanowski [38], and to the current best polynomial space algorithm with running time of $O^*(1.1996^n)$ by Xiao and Nagamochi [44].

The reduction rules used for these algorithms are often staggeringly simple, including *pendant vertex removal*, *vertex folding* [10] and *twin* reductions [43], which eliminate nearly all vertices of degree three or less from the graph. These algorithms apply reductions during recursion, only branching when the graph can no longer be reduced [17], and are referred to as *branch-and-reduce* algorithms. Further techniques used to accelerate these algorithms include *branching rules* [25, 16] which eliminate unnecessary branches from the search tree, as well as faster exponential-time algorithms for graphs of small maximum degree [44].

**Parameterized Algorithms.** For parameterized algorithms, we now turn to the minimum vertex cover problem. The most efficient algorithms for computing a minimum vertex cover in both theory and practice repeatedly apply data reduction rules to obtain a (hopefully) much smaller problem instance. If this smaller instance has size bounded by a function of some parameter, it's called a *kernel*, and producing a polynomially-sized kernel gives a fixed-parameter tractable in the chosen parameter. Reductions are surprisingly effective for the minimum vertex cover problem. In particular, letting $k$ be the size of a minimum vertex cover, the well-known crown reduction rule produces a kernel of size $3k$ [13] and the LP-relaxation reduction due to Nemhauser and Trotter [30], produces a kernel of size $2k$ [10]. Chen et al. [11] developed the current best parameterized algorithm for minimum vertex cover, giving a branch-and-reduce algorithm with running time $O(1.2738^k + kn)$ and polynomial space. For more information on the history of vertex cover kernelization, see the recent survey by Fellows et al. [15].

**Exact Algorithms in Practice.** The most efficient maximum clique solvers use a branch-and-bound search with advanced vertex reordering strategies and pruning (typically using approximation algorithms for graph coloring, MaxSAT [27] or constraint satisfaction). The long-standing canonical algorithms for finding the maximum clique are the MCS algorithm by Tomita et al. [39] and the bit-parallel algorithms of San Segundo et al. [32, 33]. However, recently Li et al. [28] introduced the MoMC algorithm, which uses incremental MaxSAT logic to achieve speed ups of up to 1 000 over MCS. Experiments by Batsyn et al. [4] show that MCS can be sped up significantly by giving an initial solution found through local search. However, even with these state-of-the-art algorithms, graphs on thousands of vertices remain intractable. For example, a difficult graph on 4 000 required 39 wall-clock hours in a highly-parallel MapReduce cluster, and is estimated to require over a year of sequential computation [42]. Recent clique solvers for sparse graphs investigate applying simple data reduction rules, using an initial clique given by some inexact method [40, 34, 8]. However, these techniques rarely work on dense graphs, such as the complement graphs that we consider here. A thorough discussion of many results in clique finding can be found in the survey of Wu and Hao [41].

Data reductions have been successfully applied in practice to solve many problems that are intractable with general algorithms. Butenko et al. [5, 7] were the first to show that simple reductions could be used to compute exact maximum independent sets on graphs with hundreds vertices for graphs derived from error-correcting codes. Their algorithm works by first applying *isolated clique removal* reductions, then solving the remaining graph with a branch-and-bound algorithm. Later, Butenko and Trukhanov [6] introduced the *critical independent set* reduction, which was able to solve graphs produced by the Sanchis graph generator. Larson [26] later proposed an algorithm to find a *maximum* critical independent set, but in experiments it proved to be slow in practice [36]. Iwata et al. [24] then showed 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 minimum vertex cover problem, it has long been known that two such simple reductions, called *pendant vertex removal* and *vertex folding*, are particularly effective in practice. However, two seminal experimental works explored the efficacy of further reductions. Abu-Khzam et al. [1] showed that *crown reductions* are as effective (and sometimes faster) in practice than performing the LP relaxation reduction (which, as they show in the paper, removes crowns) on graphs. We briefly note that critical independent sets, together with their neighborhoods, are in fact crowns, and thus in some ways the work of Butenko and Trukhanov [6] replicates that by Abu-Khzam et al. [1], though their experiments are run on different graphs.

Later, Akiba and Iwata [2] showed that an extensive collection of advanced data reduction rules (together with branching rules and lower bounds for pruning search) are also highly effective in practice. Their algorithm finds exact minimum vertex covers on a corpus of large social networks with hundreds of thousands of vertices or more in mere seconds. More details on the reduction rules follow in Section 4.

We briefly note that we considered other reduction techniques that emphasize fast computation at the cost of a larger (irreducible) graph [9, 36, 22]; however, we did not find them as effective as Akiba and Iwata [2] for exactly solving difficult instances. This is somewhat expected, however, since these techniques are optimized to produce fast high-quality solutions when combined with inexact methods such as local search.

## 4 Techniques

We now describe techniques that we use in our solver.

**4.1 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 [43], a well-known linear-programming relaxation [24, 30] (which, consequently, removes crowns [1]), vertex folding [10], and twin, funnel, and desk reductions [43].

To be self-contained, we now give a brief description of those reductions, in order of increasing complexity. Each reduction allows us to choose vertices that are either in some minimum vertex cover, or for which we can locally choose a vertex in a minimum vertex cover after solving the remaining graph, by following simple rules. If a minimum vertex cover is found in the kernel, then each reduction may be undone, producing a minimum vertex cover in the original graph. Refer to Akiba and Iwata [2] for a more thorough discussion, including implementation details. Our implementation of the reductions is an adaptation of Akiba and Iwata's original code.

**Pendant vertices:** Any vertex $v$ of degree one, called a *pendant*, then its neighbor is in some minimum vertex cover, therefore $v$ and its neighbor $u$ can be removed from $G$.

**Vertex folding:** For a vertex $v$ with degree 2 whose neighbors $u$ and $w$ are not adjacent, either $v$ is in some minimum vertex cover, or both $u$ and $w$ are in some minimum vertex cover. Therefore, we can contract $u$, $v$, and $w$ to a single vertex $v'$ and decide which vertices are in the vertex cover after computing a minimum vertex cover on the reduced graph.

**Linear Programming Relaxation:** First introduced by Nemhauser and Trotter [30] for the vertex packing problem, they present a linear programming relaxation with a half-integral solution (i.e., using only values 0, 1/2, and 1) which can be solved using bipartite matching. Their relaxation may be formulated for the minimum vertex cover problem as follows: minimize $\sum_{v \in V} x_v$, such at for each edge $(u, v) \in E$, $x_u + x_v \geq 1$ and for each vertex $v \in V$, $x_v \geq 0$. There is a minimum vertex cover containing no vertices with value 1, and therefore their neighbors are added to the solution and removed together with the vertices from the graph. We use the further improvement from Iwata et al. [24], which computes a solution whose half-integral part is minimal.

**Unconfined [43]:** Though there are several definitions of an *unconfined* vertex in the literature, we use the simple one from Akiba and Iwata [2]. A vertex $v$ is *unconfined* when determined by the following simple algorithm. First, initialize $S = \{v\}$. Then find a $u \in N(S)$ such that $|N(u) \cap S| = 1$ and $|N(u) \setminus N[S]|$ is minimized. If there is no such vertex, then $v$ is confined. If $N(u) \setminus N[S] = \emptyset$, then $v$ is unconfined. If $N(u) \setminus N[S]$ is a single vertex $w$, then add $w$ to $S$ and repeat the algorithm. Otherwise, $v$ is confined. Unconfined vertices can be removed from the graph, since there always exists a minimum vertex cover that contains unconfined vertices.

**Twin [43]:** Let $u$ and $v$ be vertices of degree 3 with $N(u) = N(v)$. If $G[N(u)]$ has edges, then add $N(u)$ to the minimum vertex cover and remove $u$, $v$, $N(u)$, $N(v)$ from $G$. Otherwise, some $u$ and $v$ may belong to some minimum vertex cover. We still remove $u$, $v$, $N(u)$ and $N(v)$ from $G$, and add a new gadget vertex $w$ to $G$ with edges to $u$'s two-neighborhood (vertices at a distance 2 from $u$). If $w$ is in the computed minimum vertex cover, then $u$'s (and $v$'s) neighbors are in some minimum vertex cover, otherwise $u$ and $v$ are in a minimum vertex cover.

**Alternative:** Two sets of vertices $A$ and $B$ are set to be *alternatives* if $|A| = |B| \geq 1$ and there exists an minimum vertex cover $C$ such that $C \cap (A \cup B)$ is either $A$ or $B$. Then we remove $A$ and $B$ and $C = N(A) \cap N(B)$ from $G$ and add edges from each $a \in N(A) \setminus C$ to each $b \in N(B) \setminus C$. Then we add either $A$ or $B$ to $C$, depending on which neighborhood has vertices in $C$. Two structures are detected as alternatives. First, if $N(v) \setminus \{u\}$ induces a complete graph, then $\{u\}$ and $\{v\}$ are alternatives (a *funnel*). Next, if there is a cordless 4-cycle $a_1 b_1 a_2 b_2$ where each vertex has at least degree 3. Then sets $A = \{a_1, a_2\}$ and $B = \{b_1, b_2\}$ are alternatives (called a *desk*) when $|N(A) \setminus B| \leq 2$, $|N(A) \setminus B| \leq 2$, and $N(A) \cap N(B) = \emptyset$.

**4.2 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 as well as lower bounds to prune search.

**Branching.** When branching, a vertex of maximum degree is chosen for inclusion into the vertex cover. Mirrors and satellites are detected when branching, in order to eliminate branching on certain vertices. A *mirror* of a vertex $v$ is a vertex $u \in N^2(v)$ such that $N(v) \setminus N(u)$ is a clique or empty. Fomin et al. [16] show that either the mirrors of $v$ or $N(v)$ is in a minimum vertex cover, and we can therefore branch on all mirrors at once. This branching prevents branching on mirrors individually and decreases the size of the remaining graph (and thus the depth of the search tree). A *satellite* of a vertex $v$ is a vertex $u \in N^2(v)$ such that there exists a vertex $w \in N(v)$ such that $N(w) \setminus N[v] = \{u\}$. If a vertex has no mirrors, then either $v$ is in a minimum vertex cover or the neighbors of $v$'s satellites in a minimum vertex cover. Akiba and Iwata [2] further introduce *packing* branching, maintaining linear inequalities for each vertex included or excluded from the current vertex cover (called *packing constraints*) throughout recursion; when a constraint is violated, further branching can be eliminated.

**Lower Bounds.** We briefly remark that Akiba and Iwata [2] implement lower bounds to prune the search space. Their lower bounds are based on clique cover, the LP relaxation, and cycle covers (see their paper for further details). The final lower bound used for pruning is the maximum of these three.

**4.3 Branch-and-Bound.** Experiments by Strash [36] show that the full power of branch-and-reduce is only needed *very rarely* in real-world instances; kernelization followed by a 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. [28], 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.

**4.4 Iterated Local Search.** Batsyn et al. [4] 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 the iterated local search algorithm by Andrade et al. [3] to prime the *branch-and-reduce* solver with a high-quality initial solution. To the best of our knowledge, this has not been tried before. 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 produce a high-quality solution much faster than without kernelization [9, 14].

## 5 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 $S_{\text{init}}$ on the (hopefully smaller) kernel.

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

**Phase 3. (Branch-and-Bound, short)** If Phase 2 is unsuccessful, we run the MoMC [28] clique solver on the complement of the kernel, also using a short time limit[1]. 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.

[1]Note that repeatedly checking the time can slow down a highly optimized branch-and-bound solver considerably; we therefore simulate time checking by using a limit on the number of branches.

**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 $S_{\text{init}}$ 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 hardest-to-compute instances.

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

## 6 Experimental Results

We now look at the impact of the algorithmic components on the number of instances solved. Here, we focus on the public instances of the PACE 2019 Implementation Challenge, Vertex Cover Track A, obtained from `https://pacechallenge.org/files/pace2019-vc-exact-public-v2.tar.bz2`. This set contains 100 instances overall. We also summarize the results comparing against the second and third best competing algorithms on the private instances during the challenge (which can be found at `https://pacechallenge.org/2019/` and `https://www.optil.io/optilion/problem/3155`). Note that further comparisons are not yet possible, as the private instances have not yet been released.

**6.1 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 32 768 KB of 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`. Our source code is publicly available under the MIT license at [23]. Each algorithm was run sequentially with a time limit of 30 minutes—the time allotted to solve a single data set in the PACE

2019 Implementation Challenge. Our primary focus is on the total number of instances solved.

**6.2 Evaluation.** We now explain the main configuration that we use in our experimental setup. In the following, MoMC runs the MoMC clique solver by Li et al. [28] on the complement of the input graph; RMoMC applies reductions to the input graph exhaustively, and then runs MoMC on the complement of the resulting kernel; LSBnR applies reductions exhaustively, then runs local search to obtain a high-quality solution on the kernel which is used as a initial bound in the branch-and-reduce algorithm that is run on the kernel; BnR applies reductions and then runs the branch-and-reduce algorithm on the kernel (no local search is used to improve an initial bound); FullA is the full algorithm as described in the previous section, using a short time limit of one second and a long time limit of thirty seconds.

Tables 1 and 2 give an overview of the instances that each of the solver solved, including the kernel size, and the minimum vertex cover size for those instances solved by any of the four algorithms. Overall, MoMC can solve 30 out of the 100 instances. Applying reductions first enables RMoMC to solve 68 instances. However, curiously, there are two instances (instances 131 and 157) that MoMC solves, but that RMoMC can not solve. In these cases, kernelization reduced the number of nodes, but *increased* the number of edges. This is due to the *alternative* reduction, which in some cases can create more edges than initially present. This is why we choose to also run MoMC on the unkernelized input graph in FullA (in order to solve those instances as well).

LSBnR solves 55 of the 100 instances. Priming the branch-and-reduce algorithm with an initial solution computed by local search has a significant impact: LSBnR solves 13 more instances than BnR, which solve 42 instances. In particular, using local search to find an initial bound helps to solve large instances in which the initial kernelization step does not reduce the graph fully. Surprisingly, RMoMC solves 26 instances that BnR does not (and even LSBnR is only able to solve one of these instances). To the best of our knowledge, this is the first time that kernelization followed by branch-and-bound is shown to significantly outperform branch-and-reduce. Our full algorithm FullA solves 82 of the 100 instances and, as expected, dominates each of the other configurations. This can be further seen from the plot in Figure 1, which shows how many instances each algorithm solves over time (this includes all 100 public and 100 private instances of the challenge). Note that LSBnR and RMoMC solve more instances in narrow time gaps, due to FullA's set up cost and running multiple algorithms. However, FullA quickly makes up for this

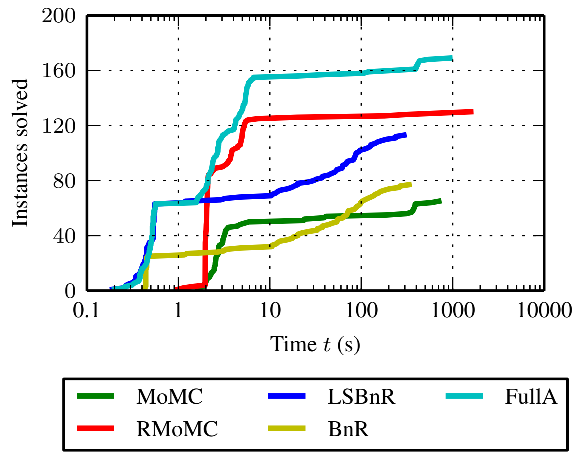


Figure 1: Number of instances solved over time by each algorithm over *all* instances. At each time step $t$, we count each instance solved by the algorithm in at most $t$ seconds.

and overtakes all algorithms at approximately eight seconds. In addition to the 100 public instances, the PACE Implementation Challenge tests all submissions on 100 private instances. Tables 3 and 4 give detailed per instances results on those instances. The results are similar to the results on the private instances. On the private instances, MoMC can solve 35 out of the 100 instances, RMoMC solves 62, LSBnR solves 58 and BnR solves 35 instances. Our full algorithm FullA solved 87 of the 100 instances, which is 10 more instances than the second-place submission (peaty [31], solving 77), and 11 more than the third-place submission (bogdan [46]), solving 76). Our solver dominates these other solvers: with the exception of one graph, our algorithm solves all instances that peaty and bogdan can solve combined.

We briefly describe these two solvers. The peaty solver uses reductions to compute a problem kernel of the input followed by an unpublished maximum weight clique solver on the complement of each of the connected components of the kernel to assemble a solution. The clique solver is similar to MaxCLQ by Li and Quan [29], but is more general. Local search is used to obtain an initial solution. On the other hand, bogdan implemented a small suite of simple reductions (including vertex folding, isolated clique removal, and degree-one removal) together with a recent maximum clique solver by Szabó and Zavalnij [37].

Lastly, we note that our choice of using MoMC as our chosen branch-and-bound solver is significant on the private instances. Eight instances solved exclusively by our solver are solved in Phase 5, where MoMC is run until the end of the challenge time limit.

## 7 Conclusion

We presented the winning solver of the PACE 2019 Implementation Challenge Vertex Cover Track. 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, our experiments show the impact of the different solver techniques on the number of instances solved during the challenge. In particular, the results emphasize that data reductions play an important tool to boost the performance of branch-and-bound, and local search is highly effective to boost the performance of branch-and-reduce.

**Acknowledgments.** We wish to thank the organizers of the PACE 2019 Implementation Challenge for providing us with the opportunity and means to test our algorithmic ideas. We also are indebted to Takuya Akiba and Yoichi Iwata for sharing their original branch-and-reduce source code[2], and to Chu-Min Li, Hua Jiang, and Felip Manyà for not only sharing—but even open sourcing—their code for MoMC at our request[3]. Their solver was of critical importance to our algorithm's success.

## 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.

[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] 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.

[5] S. Butenko, P. Pardalos, I. Sergienko, V. Shylo, and P. Stetsyuk. Finding maximum independent sets in graphs arising from coding theory. In *Proc. 2002 ACM Symposium on Applied Computing (SAC'02)*, pages 542–546. ACM, 2002. doi:10.1145/508791.508897.

[6] S. Butenko and S. Trukhanov. Using critical sets to solve the maximum independent set problem. *Oper. Res. Lett.*, 35(4):519–524, 2007. doi:10.1016/j.orl.2006.07.004.

[7] Sergiy Butenko, Panos Pardalos, Ivan Sergienko, Vladimir Shylo, and Petro Stetsyuk. Estimating the size of correcting codes using extremal graph problems. In Charles Pearce and Emma Hunt, editors, *Optimization*, volume 32 of *Springer Optimization and Its Applications*, pages 227–243. Springer, 2009. doi:10.1007/978-0-387-98096-6_12.

[8] Lijun Chang. Efficient maximum clique computation over large sparse graphs. In *Proc. 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 529–538. ACM, 2019. doi:10.1145/3292500.3330986.

[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] Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40):3736–3756, 2010. doi:10.1016/j.tcs.2010.06.026.

[12] Tammy M. K. Cheng, Yu-En Lu, Michele Vendruscolo, Pietro Lio', and Tom L. Blundell. Prediction by graph theoretic measures of structural effects in proteins arising from non-synonymous single nucleotide polymorphisms. *PLOS Computational Biology*, 4(7):1–9, 07 2008. doi:10.1371/journal.pcbi.1000135.

[13] Benny Chor, Mike Fellows, and David Juedes. Linear kernels in linear time, or how to save k colors in $o(n^2)$ steps. In Juraj Hromkovič, Manfred Nagl, and Bernhard Westfechtel, editors, *Graph-Theoretic Concepts in Computer Science*, pages 257–269. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. doi:10.1007/978-3-540-30559-0_22.

[14] 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.

[15] Michael R. Fellows, Lars Jaffke, Aliz Izabella Király, Frances A. Rosamond, and Mathias Weller. *What Is Known About Vertex Cover Kernelization?*, pages 330–356. Springer, 2018. doi:10.1007/978-3-319-98355-4_19.

[2] https://github.com/wata-orz/vertex_cover

[3] https://home.mis.u-picardie.fr/~cli/EnglishPage.html

[16] Fedor V Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *Journal of the ACM*, 56(5):25, 2009. doi:10.1145/1552285.1552286.

[17] F.V. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Springer, 2010. doi:10.1007/978-3-642-16533-7.

[18] Eleanor J. Gardiner, , Peter Willett, and Peter J. Artymiuk. Graph-theoretic techniques for macromolecular docking. *Journal of Chemical Information and Computer Science*, 40(2):273–279, 2000. doi:10.1021/ci990262o.

[19] A. Gemsa, B. Niedermann, and M. Nöllenburg. Trajectory-Based Dynamic Map Labeling. In *Proc. 24th Int. Symp. on Algorithms and Computation (ISAAC'13)*, LNCS, pages 413–423. Springer, 2013. URL: http://dx.doi.org/10.1007/978-3-642-45030-3_39.

[20] L. Gottlieb, A. Kontorovich, and R. Krauthgamer. Efficient classification for metric data. *IEEE Transactions on Information Theory*, 60(9):5750–5759, Sep. 2014. doi:10.1109/TIT.2014.2339840.

[21] F. Harary and I. C. Ross. A Procedure for Clique Detection Using the Group Matrix. *Sociometry*, 20(3):pp. 205–215, 1957. URL: http://www.jstor.org/stable/2785673.

[22] 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. doi:10.1137/1.9781611975055.19.

[23] Demian Hespe, Sebastian Lamm, Christian Schulz, and Darren Strash. WeGotYouCovered, May 2019. doi:10.5281/zenodo.2816116.

[24] 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.

[25] Joachim Kneis, Alexander Langer, and Peter Rossmanith. A Fine-grained Analysis of a Simple Independent Set Algorithm. In Ravi Kannan and K. Narayan Kumar, editors, *Proc. 29th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2009)*, volume 4 of *LIPIcs*, pages 287–298, 2009. doi:10.4230/LIPIcs.FSTTCS.2009.2326.

[26] 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.

[27] C. Li, Z. Fang, and K. Xu. Combining MaxSAT Reasoning and Incremental Upper Bound for the Maximum Clique Problem. In *Proceedings of 25th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 939–946, Nov 2013. doi:10.1109/ICTAI.2013.143.

[28] 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.

[29] Chu Min Li and Zhe Quan. An efficient branch-and-bound algorithm based on MaxSAT for the maximum clique problem. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, 2010. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1611.

[30] 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.

[31] Patrick Prosser and James Trimble. Peaty: an exact solver for the vertex cover problem, May 2019. doi:10.5281/zenodo.3082356.

[32] P. San Segundo, F. Matia, D. Rodriguez-Losada, and M. Hernando. An improved bit parallel exact maximum clique algorithm. *Optim. Lett.*, 7(3):467–479, 2013. doi:10.1007/s11590-011-0431-y.

[33] P. San Segundo, D. Rodríguez-Losada, and A. Jiménez. An exact bit-parallel algorithm for the maximum clique problem. *Comput. Oper. Res.*, 38(2):571–581, 2011. doi:10.1016/j.cor.2010.07.019.

[34] Pablo San Segundo, Alvaro Lopez, and Panos M. Pardalos. A new exact maximum clique algorithm for large and massive sparse graphs. *Computers & Operations Research*, 66:81–94, 2016. doi:10.1016/j.cor.2015.07.013.

[35] Pedro V. Sander, Diego Nehab, Eden Chlamtac, and Hugues Hoppe. Efficient traversal of mesh edges using adjacency primitives. *ACM Trans. Graph.*, 27(5):144:1–144:9, December 2008. doi:10.1145/1409060.1409097.

[36] 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.

[37] S. Szabó and B. Zaválnij. A different approach to maximum clique search. In *2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 310–316. IEEE, Sep. 2018. doi:10.1109/SYNASC.2018.00055.

[38] R. E. Tarjan and A. E. Trojanowski. Finding a maximum independent set. *SIAM J. Comput.*, 6(3):537–546, 1977. doi:10.1137/0206038.

[39] E. Tomita, Y. Sutani, T. Higashi, S. Takahashi, and M. Wakatsuki. A simple and faster branch-and-bound algorithm for finding a maximum clique. In Md. Saidur Rahman and Satoshi Fujita, editors, *Algorithms and Computation (WALCOM'10)*, volume 5942 of *LNCS*, pages 191–203. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-11440-3_18.

[40] Anurag Verma, Austin Buchanan, and Sergiy Butenko. Solving the maximum clique and vertex coloring problems on very large sparse networks. *INFORMS Journal on Computing*, 27(1):164–177, 2015. doi:10.1287/ijoc.2014.0618.

[41] Q. Wu and J. Hao. A review on algorithms for maximum clique problems. *European Journal of Operational Research*, 242(3):693 – 709, 2015. doi:10.1016/j.ejor.2014.09.064.

[42] J. Xiang, C. Guo, and A. Aboulnaga. Scalable maximum clique computation using mapreduce. In *Proc. IEEE 29th International Conference on Data Engineering (ICDE'13)*, pages 74–85, April 2013. doi:10.1109/ICDE.2013.6544815.

[43] 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.

[44] Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. *Information and Computation*, 255:126–146, 2017. doi:10.1016/j.ic.2017.06.001.

[45] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New Algorithms for Fast Discovery of Association Rules. In *3rd International Conference on Knowledge Discovery and Data Mining*, pages 283–286. AAAI Press, 1997.

[46] Bogdan Zavalnij. zbogdan/pace-2019 a, May 2019. doi:10.5281/zenodo.3228802.

Table 1: Detailed per instance results for public instances. The columns $n$ and $m$ refer to the number of nodes and edges of the input graph, $n'$ and $m'$ refer to the number of nodes and edges of the kernel graph after reductions have been applied exhaustively, and $|VC|$ refers to the size of the minimum vertex cover of the input graph. We list a '✓' when a solver successfully solved the given instance in the time limit, and '-' otherwise.

| inst# | $n$ | $m$ | $n'$ | $m'$ | MoMC | RMoMC | LSBnR | BnR | FullA | $\|VC\|$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 001 | 6 160 | 40 207 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 2 586 |
| 003 | 60 541 | 74 220 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 12 190 |
| 005 | 200 | 819 | 192 | 800 | ✓ | ✓ | ✓ | ✓ | ✓ | 129 |
| 007 | 8 794 | 10 130 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 4 397 |
| 009 | 38 452 | 174 645 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 21 348 |
| 011 | 9 877 | 25 973 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 4 981 |
| 013 | 45 307 | 55 440 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 8 610 |
| 015 | 53 610 | 65 952 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 10 670 |
| 017 | 23 541 | 51 747 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 12 082 |
| 019 | 200 | 884 | 194 | 862 | ✓ | ✓ | ✓ | ✓ | ✓ | 130 |
| 021 | 24 765 | 30 242 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 5 110 |
| 023 | 27 717 | 133 665 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 16 013 |
| 025 | 23 194 | 28 221 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 4 899 |
| 027 | 65 866 | 81 245 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 13 431 |
| 029 | 13 431 | 21 999 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 6 622 |
| 031 | 200 | 813 | 198 | 818 | ✓ | ✓ | ✓ | ✓ | ✓ | 136 |
| 033 | 4 410 | 6 885 | 138 | 471 | - | ✓ | ✓ | ✓ | ✓ | 2 725 |
| 035 | 200 | 884 | 189 | 859 | ✓ | ✓ | ✓ | ✓ | ✓ | 133 |
| 037 | 198 | 824 | 194 | 810 | ✓ | ✓ | ✓ | ✓ | ✓ | 131 |
| 039 | 6 795 | 10 620 | 219 | 753 | - | ✓ | ✓ | ✓ | ✓ | 4 200 |
| 041 | 200 | 1 040 | 200 | 1 023 | ✓ | ✓ | ✓ | ✓ | ✓ | 139 |
| 043 | 200 | 841 | 198 | 844 | ✓ | ✓ | ✓ | ✓ | ✓ | 139 |
| 045 | 200 | 1 044 | 200 | 1 020 | ✓ | ✓ | ✓ | ✓ | ✓ | 137 |
| 047 | 200 | 1 120 | 198 | 1 080 | ✓ | ✓ | ✓ | ✓ | ✓ | 140 |
| 049 | 200 | 957 | 198 | 930 | ✓ | ✓ | ✓ | ✓ | ✓ | 136 |
| 051 | 200 | 1 135 | 200 | 1 098 | ✓ | ✓ | ✓ | ✓ | ✓ | 140 |
| 053 | 200 | 1 062 | 200 | 1 026 | ✓ | ✓ | ✓ | ✓ | ✓ | 139 |
| 055 | 200 | 958 | 194 | 938 | ✓ | ✓ | ✓ | ✓ | ✓ | 134 |
| 057 | 200 | 1 200 | 197 | 1 139 | ✓ | ✓ | ✓ | ✓ | ✓ | 142 |
| 059 | 200 | 988 | 193 | 954 | ✓ | ✓ | ✓ | ✓ | ✓ | 137 |
| 061 | 200 | 952 | 198 | 914 | ✓ | ✓ | ✓ | ✓ | ✓ | 135 |
| 063 | 200 | 1 040 | 200 | 1 011 | ✓ | ✓ | ✓ | ✓ | ✓ | 138 |
| 065 | 200 | 1 037 | 200 | 1 011 | ✓ | ✓ | ✓ | ✓ | ✓ | 138 |
| 067 | 200 | 1 201 | 200 | 1 174 | ✓ | ✓ | ✓ | ✓ | ✓ | 143 |
| 069 | 200 | 1 120 | 196 | 1 077 | ✓ | ✓ | ✓ | ✓ | ✓ | 140 |
| 071 | 200 | 984 | 200 | 952 | ✓ | ✓ | ✓ | ✓ | ✓ | 136 |
| 073 | 200 | 1 107 | 200 | 1 078 | ✓ | ✓ | ✓ | ✓ | ✓ | 139 |
| 075 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 077 | 200 | 988 | 193 | 954 | ✓ | ✓ | ✓ | ✓ | ✓ | 137 |
| 079 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 081 | 199 | 1 124 | 197 | 1 087 | ✓ | ✓ | ✓ | ✓ | ✓ | 141 |
| 083 | 200 | 1 215 | 198 | 1 182 | ✓ | ✓ | ✓ | ✓ | ✓ | 144 |
| 085 | 11 470 | 17 408 | 3 539 | 25 955 | - | - | - | - | - | |
| 087 | 13 590 | 21 240 | 441 | 1 512 | - | ✓ | - | - | ✓ | 8 400 |
| 089 | 57 316 | 77 978 | 16 834 | 54 847 | - | - | - | - | - | |
| 091 | 200 | 1 196 | 200 | 1 163 | ✓ | ✓ | ✓ | ✓ | ✓ | 145 |
| 093 | 200 | 1 207 | 200 | 1 162 | ✓ | ✓ | ✓ | ✓ | ✓ | 143 |
| 095 | 15 783 | 24 663 | 510 | 1 746 | - | ✓ | - | - | ✓ | 9 755 |
| 097 | 18 096 | 28 281 | 579 | 1 995 | - | ✓ | - | - | ✓ | 11 185 |
| 099 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |

Table 2: Detailed per instance results for public instances. The columns $n$ and $m$ refer to the number of nodes and edges of the input graph, $n'$ and $m'$ refer to the number of nodes and edges of the kernel graph after reductions have been applied exhaustively, and $|VC|$ refers to the size of the minimum vertex cover of the input graph. We list a '✓' when a solver successfully solved the given instance in the time limit, and '-' otherwise.

| inst# | $n$ | $m$ | $n'$ | $m'$ | MoMC | RMoMC | LSBnR | BnR | FullA | $\|VC\|$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 101 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 103 | 15 783 | 24 663 | 513 | 1 752 | - | ✓ | - | - | ✓ | 9 755 |
| 105 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 107 | 13 590 | 21 240 | 435 | 1 500 | - | ✓ | - | - | ✓ | 8 400 |
| 109 | 66 992 | 90 970 | 20 336 | 66 350 | - | - | - | - | - | |
| 111 | 450 | 17 831 | 450 | 17 831 | ✓ | ✓ | - | - | ✓ | 420 |
| 113 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 115 | 18 096 | 28 281 | 573 | 1 986 | - | ✓ | - | - | ✓ | 11 185 |
| 117 | 18 096 | 28 281 | 582 | 2 007 | - | ✓ | - | - | ✓ | 11 185 |
| 119 | 18 096 | 28 281 | 588 | 2 016 | - | ✓ | - | - | ✓ | 11 185 |
| 121 | 18 096 | 28 281 | 579 | 1 998 | - | ✓ | - | - | ✓ | 11 185 |
| 123 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 125 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 127 | 18 096 | 28 281 | 582 | 2 001 | - | ✓ | - | - | ✓ | 11 185 |
| 129 | 15 783 | 24 663 | 507 | 1 752 | - | ✓ | - | - | ✓ | 9 755 |
| 131 | 2 980 | 5 360 | 2 179 | 6 951 | ✓ | - | - | - | ✓ | 1 920 |
| 133 | 15 783 | 24 663 | 507 | 1 746 | - | ✓ | - | - | ✓ | 9 755 |
| 135 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 137 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 139 | 18 096 | 28 281 | 579 | 1 995 | - | ✓ | - | - | ✓ | 11 185 |
| 141 | 18 096 | 28 281 | 576 | 1 995 | - | ✓ | - | - | ✓ | 11 185 |
| 143 | 18 096 | 28 281 | 582 | 2 001 | - | ✓ | - | - | ✓ | 11 185 |
| 145 | 18 096 | 28 281 | 576 | 1 989 | - | ✓ | - | - | ✓ | 11 185 |
| 147 | 18 096 | 28 281 | 567 | 1 974 | - | ✓ | - | - | ✓ | 11 185 |
| 149 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 151 | 15 783 | 24 663 | 501 | 1 728 | - | ✓ | - | - | ✓ | 9 755 |
| 153 | 29 076 | 45 570 | 2 124 | 16 266 | - | - | - | - | - | |
| 155 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 157 | 2 980 | 5 360 | 2 169 | 6 898 | ✓ | - | - | - | ✓ | 1 920 |
| 159 | 18 096 | 28 281 | 582 | 2 004 | - | ✓ | - | - | ✓ | 11 185 |
| 161 | 138 141 | 227 241 | 41 926 | 202 869 | - | - | - | - | - | |
| 163 | 18 096 | 28 281 | 582 | 2 004 | - | ✓ | - | - | ✓ | 11 185 |
| 165 | 18 096 | 28 281 | 576 | 1 995 | - | ✓ | - | - | ✓ | 11 185 |
| 167 | 15 783 | 24 663 | 510 | 1 746 | - | ✓ | - | - | ✓ | 9 755 |
| 169 | 4 768 | 8 576 | 3 458 | 11 014 | - | - | - | - | - | |
| 171 | 18 096 | 28 281 | 576 | 1 989 | - | ✓ | - | - | ✓ | 11 185 |
| 173 | 56 860 | 77 264 | 17 090 | 55 568 | - | - | - | - | - | |
| 175 | 3 523 | 6 446 | 2 723 | 8 570 | - | - | - | - | - | |
| 177 | 5 066 | 9 112 | 3 704 | 11 797 | - | - | - | - | - | |
| 179 | 15 783 | 24 663 | 504 | 1 740 | - | ✓ | - | - | ✓ | 9 755 |
| 181 | 18 096 | 28 281 | 573 | 1 989 | - | ✓ | ✓ | - | ✓ | 11 185 |
| 183 | 72 420 | 118 362 | 30 340 | 133 872 | - | - | - | - | - | |
| 185 | 3 523 | 6 446 | 2 723 | 8 568 | - | - | - | - | - | |
| 187 | 4 227 | 7 734 | 3 264 | 10 286 | - | - | - | - | - | |
| 189 | 7 400 | 13 600 | 5 802 | 18 212 | - | - | - | - | - | |
| 191 | 4 579 | 8 378 | 3 539 | 11 137 | - | - | - | - | - | |
| 193 | 7 030 | 12 920 | 5 510 | 17 294 | - | - | - | - | - | |
| 195 | 1 150 | 81 068 | 1 150 | 81 068 | - | - | - | - | - | |
| 197 | 1 534 | 127 011 | 1 534 | 127 011 | - | - | - | - | - | |
| 199 | 1 534 | 126 163 | 1 534 | 126 163 | - | - | - | - | - | |

Table 3: Detailed per instance results for private instances. The columns $n$ and $m$ refer to the number of nodes and edges of the input graph, $n'$ and $m'$ refer to the number of nodes and edges of the kernel graph after reductions have been applied exhaustively, and $|VC|$ refers to the size of the minimum vertex cover of the input graph. We list a '✓' when a solver successfully solved the given instance in the time limit, and '-' otherwise.

| inst# | $n$ | $m$ | $n'$ | $m'$ | MoMC | RMoMC | LSBnR | BnR | FullA | $\lvert VC\rvert$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 002 | 51 795 | 63 334 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 10 605 |
| 004 | 8 114 | 26 013 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 2 574 |
| 006 | 200 | 751 | 188 | 716 | ✓ | ✓ | ✓ | ✓ | ✓ | 126 |
| 008 | 7 537 | 72 833 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 3 345 |
| 010 | 199 | 774 | 189 | 756 | ✓ | ✓ | ✓ | ✓ | ✓ | 127 |
| 012 | 53 444 | 68 044 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 10 918 |
| 014 | 25 123 | 31 552 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 5 111 |
| 016 | 153 | 802 | 153 | 802 | - | - | - | - | - | |
| 018 | 49 212 | 63 601 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 10 201 |
| 020 | 57 287 | 71 155 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 11 648 |
| 022 | 12 589 | 33 129 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 6 749 |
| 024 | 7 620 | 47 293 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 4 364 |
| 026 | 6 140 | 36 767 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 2 506 |
| 028 | 54 991 | 67 000 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 11 211 |
| 030 | 62 853 | 79 557 | 0 | 0 | - | ✓ | ✓ | ✓ | ✓ | 13 338 |
| 032 | 1 490 | 2 680 | 1 081 | 3 426 | ✓ | - | - | - | ✓ | 960 |
| 034 | 1 490 | 2 680 | 1 090 | 3 467 | ✓ | ✓ | - | - | ✓ | 960 |
| 036 | 26 300 | 41 500 | 500 | 3 000 | - | ✓ | ✓ | ✓ | ✓ | 16 300 |
| 038 | 786 | 14 024 | 460 | 6 623 | ✓ | ✓ | ✓ | ✓ | ✓ | 605 |
| 040 | 210 | 625 | 210 | 625 | ✓ | ✓ | - | - | ✓ | 145 |
| 042 | 200 | 974 | 200 | 952 | ✓ | ✓ | ✓ | ✓ | ✓ | 136 |
| 044 | 200 | 1 186 | 200 | 1 147 | ✓ | ✓ | ✓ | ✓ | ✓ | 142 |
| 046 | 200 | 812 | 200 | 812 | ✓ | ✓ | ✓ | ✓ | ✓ | 137 |
| 048 | 200 | 1 052 | 198 | 1 022 | ✓ | ✓ | ✓ | ✓ | ✓ | 138 |
| 050 | 200 | 1 048 | 200 | 1 025 | ✓ | ✓ | ✓ | ✓ | ✓ | 140 |
| 052 | 200 | 1 019 | 198 | 1 000 | ✓ | ✓ | ✓ | ✓ | ✓ | 138 |
| 054 | 200 | 985 | 198 | 951 | ✓ | ✓ | ✓ | ✓ | ✓ | 137 |
| 056 | 200 | 1 117 | 200 | 1 089 | ✓ | ✓ | ✓ | ✓ | ✓ | 141 |
| 058 | 200 | 1 202 | 200 | 1 171 | ✓ | ✓ | ✓ | ✓ | ✓ | 142 |
| 060 | 200 | 1 147 | 200 | 1 118 | ✓ | ✓ | ✓ | ✓ | ✓ | 141 |
| 062 | 199 | 1 164 | 199 | 1 128 | ✓ | ✓ | ✓ | ✓ | ✓ | 141 |
| 064 | 200 | 1 071 | 198 | 1 040 | ✓ | ✓ | ✓ | ✓ | ✓ | 138 |
| 066 | 200 | 884 | 198 | 875 | ✓ | ✓ | ✓ | ✓ | ✓ | 134 |
| 068 | 200 | 983 | 198 | 961 | ✓ | ✓ | ✓ | ✓ | ✓ | 135 |
| 070 | 200 | 887 | 198 | 856 | ✓ | ✓ | ✓ | ✓ | ✓ | 133 |
| 072 | 200 | 1 204 | 198 | 1 176 | ✓ | ✓ | ✓ | ✓ | ✓ | 140 |
| 074 | 200 | 820 | 194 | 785 | ✓ | ✓ | ✓ | ✓ | ✓ | 132 |
| 076 | 26 300 | 41 500 | 500 | 3 000 | - | ✓ | ✓ | - | ✓ | 16 300 |
| 078 | 11 349 | 17 739 | 357 | 1 245 | - | ✓ | - | - | ✓ | 7 015 |
| 080 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 082 | 200 | 978 | 196 | 956 | ✓ | ✓ | ✓ | ✓ | ✓ | 138 |
| 084 | 13 590 | 21 240 | 435 | 1 503 | - | ✓ | - | - | ✓ | 8 400 |
| 086 | 26 300 | 41 500 | 500 | 3 000 | - | ✓ | ✓ | - | ✓ | 16 300 |
| 088 | 26 300 | 41 500 | 500 | 3 000 | - | ✓ | ✓ | - | ✓ | 16 300 |
| 090 | 11 349 | 17 739 | 357 | 1 245 | - | ✓ | - | - | ✓ | 7 015 |
| 092 | 450 | 17 794 | 450 | 17 794 | ✓ | ✓ | - | - | ✓ | 420 |
| 094 | 5 960 | 10 720 | 4 217 | 13 456 | - | - | - | - | - | |
| 096 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 098 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 100 | 26 300 | 41 500 | 500 | 3 000 | - | ✓ | ✓ | ✓ | ✓ | 16 300 |

Table 4: Detailed per instance results for private instances. The columns $n$ and $m$ refer to the number of nodes and edges of the input graph, $n'$ and $m'$ refer to the number of nodes and edges of the kernel graph after reductions have been applied exhaustively, and $|VC|$ refers to the size of the minimum vertex cover of the input graph. We list a '✓' when a solver successfully solved the given instance in the time limit, and '-' otherwise.

| inst# | $n$ | $m$ | $n'$ | $m'$ | MoMC | RMoMC | LSBnR | BnR | FullA | $\|VC\|$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 102 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 104 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 106 | 2 980 | 5 360 | 2 136 | 6 809 | ✓ | - | - | - | ✓ | 1 920 |
| 108 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 110 | 98 128 | 161 357 | 29 168 | 140 392 | - | - | - | - | - | |
| 112 | 18 096 | 28 281 | 576 | 1 992 | - | ✓ | - | - | ✓ | 11 185 |
| 114 | 15 783 | 24 663 | 504 | 1 740 | - | ✓ | - | - | ✓ | 9 755 |
| 116 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 118 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 120 | 70 144 | 116 378 | 6 029 | 38 285 | - | - | - | - | - | |
| 122 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 124 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 126 | 18 096 | 28 281 | 582 | 2 001 | - | ✓ | - | - | ✓ | 11 185 |
| 128 | 26 300 | 41 500 | 500 | 3 000 | - | - | - | - | - | |
| 130 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 132 | 15 783 | 24 663 | 513 | 1 755 | - | ✓ | - | - | ✓ | 9 755 |
| 134 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 136 | 18 096 | 28 281 | 585 | 2 007 | - | ✓ | - | - | ✓ | 11 185 |
| 138 | 18 096 | 28 281 | 576 | 1 992 | - | ✓ | - | - | ✓ | 11 185 |
| 140 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 142 | 2 980 | 5 360 | 2 180 | 6 946 | ✓ | - | - | - | ✓ | 1 920 |
| 144 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 146 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 148 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 150 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 152 | 13 590 | 21 240 | 438 | 1 506 | - | ✓ | ✓ | - | ✓ | 8 400 |
| 154 | 15 783 | 24 663 | 504 | 1 737 | - | ✓ | - | - | ✓ | 9 755 |
| 156 | 450 | 17 809 | 450 | 17 809 | ✓ | ✓ | - | - | ✓ | 420 |
| 158 | 15 783 | 24 663 | 507 | 1 746 | - | ✓ | - | - | ✓ | 9 755 |
| 160 | 18 096 | 28 281 | 576 | 1 989 | - | ✓ | - | - | ✓ | 11 185 |
| 162 | 50 635 | 83 075 | 13 066 | 63 758 | - | - | - | - | - | |
| 164 | 29 296 | 46 040 | 1 210 | 8 666 | - | - | - | - | - | |
| 166 | 3 278 | 5 896 | 2 400 | 7 643 | ✓ | - | - | - | - | 2 112 |
| 168 | 2 980 | 5 360 | 2 180 | 6 943 | ✓ | - | - | - | ✓ | 1 920 |
| 170 | 15 783 | 24 663 | 507 | 1 746 | - | ✓ | - | - | ✓ | 9 755 |
| 172 | 4 025 | 7 435 | 3 158 | 9 863 | - | - | - | - | - | |
| 174 | 2 980 | 5 360 | 2 180 | 6 955 | ✓ | - | - | - | ✓ | 1 920 |
| 176 | 15 783 | 24 663 | 501 | 1 734 | - | ✓ | - | - | ✓ | 9 755 |
| 178 | 18 096 | 28 281 | 573 | 1 995 | - | ✓ | - | - | ✓ | 11 185 |
| 180 | 15 783 | 24 663 | 501 | 1 731 | - | ✓ | - | - | ✓ | 9 755 |
| 182 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 184 | 6 290 | 11 560 | 4 904 | 15 397 | - | - | - | - | - | |
| 186 | 26 300 | 41 500 | 500 | 3 000 | - | - | ✓ | - | ✓ | 16 300 |
| 188 | 6 660 | 12 240 | 5 220 | 16 375 | - | - | - | - | - | |
| 190 | 3 875 | 7 090 | 2 997 | 9 424 | - | - | - | - | - | |
| 192 | 2 980 | 5 360 | 2 180 | 6 941 | ✓ | - | - | - | ✓ | 1 920 |
| 194 | 1 150 | 80 851 | 1 150 | 80 851 | ✓ | ✓ | - | - | ✓ | 1 100 |
| 196 | 1 534 | 126 082 | 1 534 | 126 082 | - | - | - | - | - | |
| 198 | 1 150 | 80 072 | 1 150 | 80 072 | ✓ | ✓ | - | - | ✓ | 1 100 |
| 200 | 1 150 | 80 258 | 1 150 | 80 258 | ✓ | ✓ | - | - | ✓ | 1 100 |