PACE Solver Description: UzL Solver for Dominating Set and Hitting Set

Max Bannach
European Space Agency, AI and Data Science Section, Noordwijk, The Netherlands

Florian Chudigiewitsch
Institute for Theoretical Computer Science, University of Lübeck, Germany

Marcel Wienöbst[1]
Institute for Theoretical Computer Science, University of Lübeck, Germany

Abstract

This document contains a short description of our solver for the dominating set and hitting set problems that we submitted to the exact tracks of the PACE Challenge 2025. The solver is based on a straightforward MaxSAT formulation supplemented by hitting-set-based reduction rules. It utilizes a clique solver if the reduced instance is a (small) input for the vertex cover problem and tries to match certain lower bounds by expressing the reduced instance as a SAT problem.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases exact algorithms, dominating set, hitting set

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.39

Supplementary Material *Software*: `https://github.com/mwien/PACE2025`
archived at `swh:1:dir:17aaee5f0fbc12856d7a55455c66a5f21801d327`

1 Introduction

Dominating set and hitting set are classical NP-hard problems [6]. In the *dominating set problem* we seek, for a given undirected graph $G = (V, E)$, a minimum-size set of vertices S such that for every vertex $u \in V$, the vertex u itself or one of its neighbors is in S. In the *hitting set problem*, the task is to find in a given hypergraph $H = (V, E)$ a minimum-size set of vertices S such that every edge $\{u_1, u_2, \dots\} \in E$ has a non-empty intersection with S. Clearly, any dominating set instance can be expressed as a hitting set instance with $E = \{ N[u] \mid u \in V \}$, where $N[u]$ is the closed neighborhood of u.

The solver we describe can be used to tackle both problems. Internally, any dominating set input is directly translated to the corresponding hitting set instance as described above. Thus, from now on, we solely focus on the hitting set problem.

2 Presolving with Reduction Rules

To improve the solver performance and reduce the instance complexity, we apply four reduction rules for the hitting set problem in a presolving step. These rules are designed to simplify the hypergraph structure while preserving the optimal solutions:

[1] Corresponding author.

1. If an edge consists only of a single vertex u, then add u to S and remove all edges that contain u.
2. If a vertex is in exactly one edge, and this edge has size at least two, then discard the vertex and remove it from the edge.
3. If an edge is a subset of another edge, keep only the smaller edge.
4. If there are two vertices u and v such that we have $u \in e$ for all edges e that contain v, then delete v.

The correctness of these rules is immediate: In Rule 1, the vertex u needs to be in any solution; in Rule 3, the discarded edge is hit by any solution; and for Rules 2 and 4, we can guarantee that there is always a solution without the discarded vertex by an exchange argument. For an efficient implementation, we store and update not only the edges E but also, for each vertex, the indices of the edges the vertex is part of (and pointers to its position in these edges). To check whether an edge e is a subset of another edge f, we only consider edges f containing the vertex $u \in e$ that is in the fewest number of edges. We use the same optimization when applying Rule 4.

Reduction Rules for the Dominating Set Track

It is straightforward to see that the four hitting set reduction rules presented above subsume several well-known reduction rules for the dominating set problem – for example, the so-called "Rule 1" introduced by Alber, Fellows, and Niedermeier [1]. This justifies our approach in the dominating set track, to directly reduce the problem to hitting set and then apply the aforementioned hitting set reduction rules, without relying on any dominating set-specific reductions at all.

Performance of the Reduction Rules in the Competition

For the instances in the PACE Challenge 2025, our presolving step executes in fractions of a second and significantly reduces the size of the hypergraphs. While this leads to faster solving times for many instances, we observed that, on the private test set, the number of instances solvable within 30 minutes did not increase.

3 MaxSAT Formulation of Hitting Set

The hitting set problem can be naturally expressed as the (partial) MAX-SAT problem [4]. This problem allows for both *soft* clauses and *hard* clauses, where the hard clauses must all be satisfied and the goal is to satisfy as many soft clauses as possible. Let $H = (V, E)$ be a hypergraph, then we formulate the hitting set problem as

Soft Clauses $\bigwedge_{u \in V} (\neg u)$,

Hard Clauses $\bigwedge_{\{u_1, u_2, \dots\} \in E} (u_1 \vee u_2 \vee \cdots)$.

We solve this formulation using the EvalMaxSAT solver [2]. For the dominating set track, we additionally preprocess the formula using maxpre2 [7, 9]. We do not perform this preprocessing for the hitting set instances because it did not yield performance gains on the public instances and was counterproductive in some cases. On the private instances, we observed no clear improvements due to the use of maxpre2.

Furthermore, we tweaked internal parameters of EvalMaxSAT to the problems at hand and, notably, bounded the time spent in the conflict minimization phase depending on the conflict size. One of the effects of this parameter tuning is that many easier instances are solved significantly faster compared to using the off-the-shelf EvalMaxSAT solver. On harder instances, the effect is less significant as the SAT solving dominates the computation time.

4 Using Maximum Clique and SAT Solvers

In the case that the reduced instance is a *vertex cover* instance (i.e., $|e| \leq 2$ for all $e \in E$) and has at most 350 vertices, we solve the problem in a different way. Inspired by the results of the PACE 2019 [5], we use a maximum clique solver, namely mcqd [8], on the complement graph of H. Every vertex not in the maximum clique must be in a minimum-size vertex cover. Note that we do not use this strategy for instances with more than 350 vertices, where the size of the usually very dense complement graph could be prohibitively large. Here, we use the MAX-SAT formulation stated above.

Before launching the MAX-SAT solver, we attempt to compute a strong lower bound that can potentially be matched using a SAT formulation. Specifically, we greedily search for a perfect matching in the hypergraph – i.e., we look for $|V|/2$ hyperedges of size two such that each vertex appears in exactly one of these edges. If such a matching is found, any feasible solution S must have size at least $|V|/2$. In this case, we use the SAT solver kissat [3] to verify whether this lower bound can be matched by a solution that selects exactly one vertex per matching edge. Our encoding simply introduces a Boolean variable for each matching edge, representing which of its two vertices is included in the solution. We then add clauses for all remaining hyperedges to ensure they are also hit by the selected vertices.

While the vertex cover and the matching trick seemed to be promising during preparation, we observed after the competition that they did not trigger on any of the private instances of the PACE 2025 exact tracks. Thus, they did not affect the results of the competition.

5 Sketch of Correctness

For a brief sketch of a correctness proof, observe that the reduction rules are sound and the MAX-SAT formulation for the hitting set problem is immediate (and well-known). Solving vertex cover with a clique solver in the complement graph is a textbook trick. If the SAT solver finds an assignment of size $|V|/2$ in the instances discussed above, then clearly this solution is an optimal hitting set as it matches the lower bound.

As external solvers, we rely on well-established and open-source software: EvalMaxSAT [2], kissat [3], maxpre2 [7, 9], and mcqd [8].

References

1 Jochen Alber, Michael R Fellows, and Rolf Niedermeier. Polynomial-time data reduction for dominating set. *Journal of the ACM (JACM)*, 51(3):363–384, 2004. doi:10.1145/990308.990309.

2 Florent Avellaneda. A short description of the solver EvalMaxSAT. *MaxSAT Evaluation*, 8:364, 2020.

3 Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL, Gimsatul, IsaSAT and Kissat entering the SAT Competition 2024. In *Proceedings of the SAT Competition 2024 – Solver, Benchmark and Proof Checker Descriptions*, pages 8–10, 2024.

4 Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*, volume 185. IOS press, 2009.

5 M. Ayaz Dzulfikar, Johannes K. Fichte, and Markus Hecher. The PACE 2019 Parameterized Algorithms and Computational Experiments Challenge: The Fourth Iteration. In *14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*, pages 25:1–25:23, 2019. doi:10.4230/LIPIcs.IPEC.2019.25.

6 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

7 Hannes Ihalainen, Jeremias Berg, and Matti Järvisalo. Clause redundancy and preprocessing in maximum satisfiability. In *Proceedings of the 11th International Joint Conference on Automated Reasoning (IJCAR '22)*, volume 13385 of *Lecture Notes in Computer Science*, pages 75–94. Springer, August 2022. doi:10.1007/978-3-031-10769-6_6.

8 Janez Konc and Dušanka Janezic. An improved branch and bound algorithm for the maximum clique problem. *proteins*, 4(5):590–596, 2007.

9 Tuukka Korhonen, Jeremias Berg, Paul Saikko, and Matti Järvisalo. Clause redundancy and preprocessing in maximum satisfiability. In *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing, (SAT '17)*, volume 10491 of *Lecture Notes in Computer Science*, pages 449–456. Springer, 2017.