PACE Solver Description: OBLX Exact Solver for the Dominating Set Problem

Jona Dirks
Université Clermont-Auvergne, CNRS, Mines de Saint-Étienne, Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France

Enna Gerhard
University of Bremen, Bibliothekstraße 5, 28359 Bremen, Germany

Victoria Kaial
Université Clermont-Auvergne, CNRS, Mines de Saint-Étienne, Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France

Lucas Lorieau
Université Clermont-Auvergne, CNRS, Mines de Saint-Étienne, Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France

Abstract

We present and describe the solver *OBLX* for the Dominating Set problem on graphs. This solver was developed during the PACE challenge 2025 for the Exact track. It first applies several data reduction rules and performs a polynomial time reduction to Max Sat. The resulting Max Sat instance is in turn solved using the *EvalMaxSat* solver by Florent Avellaneda.

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

Keywords and phrases complexity theory, parameterized complexity, linear programming, java, dominating set, PACE 2025

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.33

Supplementary Material *Software (Source Code)*: `https://gitlab.limos.fr/oblx/public` [4]
archived at `swh:1:dir:2863e5f060c85b6d04b04c053ab39135abee8f0d`

Funding *Lucas Lorieau*: This author has received financial support from the CNRS through the MITI inter-disciplinary programs and the IRL ReLaX.

1 Introduction

We present a brief description of the solver *OBLX* submitted for PACE Challenge 2025 (`https://pacechallenge.org/2025/`). It is an exact solver for the Dominating Set problem, that is, given a graph select a set of vertices such that every vertex that is not in the set itself has a neighbor in it.

In Section 2, we present the notation and definitions that will be used in the following. We summarize the architecture of our solver in Section 3. Section 4 is dedicated to a description of the reduction rules and their correctness. Sketches of proofs are given for non-trivial rules.

The implementation and software design builds on the solver presented in Enna Gerhard's master thesis [6], which is an improved version of the PACE submissions [2] and [5].

2 Notation and preliminary definitions

We refer to [3] for common notation and terminology on graphs and parameterized complexity.

For our reduction rules, we define a new Extended Dominating Set problem, that is identical to the common version but differentiates between three types of vertices, each of them corresponding to a status with regards to a partial solution. The different considered

20th International Symposium on Parameterized and Exact Computation (IPEC 2025).
Editors: Akanksha Agrawal and Erik Jan van Leeuwen; Article No. 33; pp. 33:1–33:4
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

vertices' types are the following. *Covered vertices* are vertices that are dominated by the current partial solution built by reduction rules; *Excluded vertices* are vertices that are forbidden in any solution we will consider. This type of vertex is typically used when a reduction rule detects that a specific vertex won't be present in any optimal solution; *Plain vertices* are not covered and not excluded. An EXTENDED DOMINATING SET instance containing only plain vertices is equivalent to a DOMINATING SET instance of the same vertices and edges.

This labeling of vertices based on the current partial solution is inspired by [7] where a similar partition is used for their own set of reduction rules. In contrast, we keep track of vertices added to the solution and remove them within the graph, which reduces the complexity of creating reduction rules.

3 Overview of the solving pipeline

Figure 1 presents the overview of the solving pipeline that we have created. We first iteratively apply a set of data reduction rules (see Section 4) in order to reduce the size of the instance to solve. We apply these rules exhaustively and always start again with the first rules once a rule has been applied successfully. When no reduction rule can be applied anymore, we can treat connected components independently. We perform two polynomial time reductions:

- We reduce our EXTENDED DOMINATING SET instance to the HITTING SET problem on hypergraphs. In the new hypergraph, we use the same vertex set of our original graph. For each non-covered vertex of the original graph we create a new hyperedge for each closed neighborhood, connecting all the vertices of said neighborhood. This follows the well known straightforward classic reduction of the DOMINATING SET problem.
- We then encode our instance of HITTING SET as a MAX SAT instance that will be finally solved using the *EvalMaxSat* solver [1]. We add soft constraints trying to exclude all vertices individually. For each hyperedge we create a hard constraint requiring it to be included in the solution.

The set of variables assigned as true in the solution of the MAX SAT instance are exactly a solution to the HITTING SET instance, and in turn for the EXTENDED DOMINATING SET instance that we had obtained after reduction rules. We now have to apply remaining instructions of reduction rules that depend on the solution of the reduced instance in backward order to obtain the solution for the original DOMINATING SET instance.

4 Description of the reduction rules

We present the different reduction rules used in the solver in Tables 1 and 2. We provide a quick description and justification of their correctness. The rules are applied iteratively: We attempt to apply the rules in their order, moving on the next rule if the rules pattern could not be matched. As soon as a rule has been applied successfully, we return to trying to applying rules beginning at the start. Simple rules that are easy to check are placed first in the order with expensive rules being applied at the end. After the last rule did not match, all reduction rules have been applied exhaustively.

DOMINATING SET instance ⟶ Exhaustive application of data reduction rules ⟶ Splitting into components → HITTING SET instances → MAX SAT instances

Figure 1 Solving pipeline.

Table 1 Data reduction rules (I).

Name and Description
1. Dominated covered Remove a covered vertex v if it has a non excluded neighbor that covers a superset of the closed neighborhood of v. Implied by Rule 2 of [7].
2. Isolated vertices Remove vertices v if $\deg(v) \leq 1$. Add v or its neighbor to the solution if necessary.
3. Exclude dominated Remove an excluded vertex v if a non-covered neighbor w exists that only adjacent to other neighbors of v. Any vertex selected to cover w also covers v.
4. All neighbors are excluded Directly include a non-covered vertex v if all of its neighbors are excluded.
5. Remove edges between excluded vertices (obviously possible)
6. Excluded covers Mark a vertex v as covered, if there exists an excluded vertex u such that $N(u) \subseteq N(v)$. If a vertex is adjacent to all neighbors of an excluded vertex, it will automatically be covered. This indirectly resolves domination between excluded vertices.
7. Dominated excluded Remove an excluded vertex v if it has a non-covered neighbor that needs to be covered by a subset of the neighborhood of v.
8. Diadems Let $v_1 \dots v_5$ be a chordless cycle such that (i) v_1, v_2 and v_3 are not covered and do not have any neighbors outside of the cycle, (ii) v_2, v_4 and v_5 are not excluded. Remove v_1, v_2 and v_3. Further add v_2 to the solution.
9. Remove edges between covered vertices (obviously possible)
10. Diamonds Assume $u_1, u_2, \dots, u_\ell$ ($\ell \geq 2$) are false twins with only two neighbors v_1, v_2 that are additionally required to not be excluded. Remove $u_2, \dots, u_\ell$ and exclude u_1. Note that any solution for the reduced graph is also a solution for the original graph. The size of an optimal solution does also not increase. Moving to a new solution of equal size: If a solution contains only one such vertex it has to be either v_1 or v_2 and we can keep that solution. Otherwise, we move to the solution containing v_1 and v_2 and none of the vertices $u_1, \dots u_\ell$.
11. Contract squares If there is a chordless cycle $v_1, \dots, v_4$ of non-covered non-excluded vertices with only v_1 and v_2 having outside neighbors and v_2 exactly one, u_2, then remove v_2, v_3, v_4 and add an edge v_1u_2, decreasing the solution size by one. For obtaining the original solution, we can add exactly one vertex: If $v_1 \in S$ then add v_2; if $u_2 \in S$ add v_4, otherwise add v_3, restoring the original coverage.
12. Ladders Let $v_{1,1}, \dots v_{3,2}$ be the vertices of an induced 3×2-grid without covered or excluded vertices. (a) If no vertices apart from $v_{1,2}$ and $v_{3,1}$ have neighbors outside of the grid, we add $v_{1,2}$ and $v_{3,1}$ to the solution. A minimum of two vertices are required to cover the grid, choosing the ones with (possible) external neighbors may cover additional vertices. (b) If no vertices besides $v_{1,1}$ and $v_{3,1}$ have neighbors outside of the grid, remove $v_{1,2}, v_{2,1}, v_{2,2}, v_{3,2}$ and add the edge $v_{1,1}v_{3,1}$. If both $v_{1,1}$ and $v_{3,1}$ are in S, add $v_{2,2}$ to the solution; if $v_{1,1} \in S$ but $v_{3,1} \notin S$, add $v_{3,2}$ to the solution; if $v_{3,1} \in S$ but $v_{1,1} \notin S$, add $v_{1,2}$ to the solution; and if both $v_{1,1}, v_{3,1} \notin S$, add $v_{2,2}$ to the solution. In all of the cases, the solution size decreases by one in the new reduced graph. Depending on $v_{1,1}$ and $v_{3,1}$ being covered in the solution in the original graph, a minimum solution would exactly contain the vertex we now add manually apart from the case where both are in the solution, in which an arbitrary additional vertex may be selected.

Table 2 Data reduction rules (II).

Name and Description
13. Small cat ears
If for two adjacent edges $\{v_1, v_2\}, \{v_2, v_3\}$, v_2 not covered or excluded, excluded vertices v_x, v_y with edges $\{v_x, v_1\}, \{v_x, v_2\}, \{v_y, v_2\}, \{v_y, v_3\}$ exist and v_2, v_x, v_y are otherwise isolated, we introduce a new covered vertex v_{13} and connect it to all vertices adjacent to v_1 or v_3. We then delete v_x, v_y, v_1, v_2, v_3. If v_{13} is included in the solution, we add both v_1, v_3, otherwise v_2. Since both v_x, v_y need to be covered, as soon as v_1 is chosen for the solution, we can push out a possibly selected v_2 to v_3 (the other way by symmetry), also covering the other side, otherwise both can be covered by v_2. This behavior is performed exactly by the newly created vertex.
14. Covered excluded sibling
If for two excluded vertices v_1, v_2, $N(v_1) \subseteq N(v_2)$, we can remove v_2 as it will be covered by every vertex used to cover v_1.
15. House
Special case of Rule 18 where $V(C) - m = \{v_1, v_2\}$, $N(m) = \{s, v_1, v_2\}$, $N(v_1) = \{s, t, m\}$ and $N(v_2) = \{m, t\}$.
16. Native ignorable vertices
More efficient implementation of a special case of Rule 3 of [7].
17. Extended native ignorable vertices
Implementation of Rule 3 of [7].
18. General s t
If there exists a connected component C of $G - \{s, t\}$ such that (i) $\{s, t\}$ dominate C, (ii) neither s nor t individually dominates C, and (iii) $\exists m \in V(C)$ that dominates C but not $C \cup \{s, t\}$, then we add a dominated vertex x with $N(x) = N(s) \cup N(t)$ and remove C, s and t. If for the new graph x is picked into the solution then pick m otherwise pick s and t. This works because in both cases the solutions size gets reduced by one and if s and t or m is in the solution C is dominated.

References

1. Florent Avellaneda. A short description of the solver EvalMaxSAT, 2020. URL: `https://hdl.handle.net/10138/318451`.
2. Moritz Bergenthal, Jona Dirks, Thorben Freese, Jakob Gahde, Enna Gerhard, Mario Grobler, and Sebastian Siebertz. PACE Solver Description: GraPA-JAVA. In *IPEC 2022*, volume 249 of *LIPIcs*, pages 30:1–30:4, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.IPEC.2022.30`.
3. Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2016. `doi:10.1007/978-3-319-21275-3`.
4. Jona Dirks, Enna Gerhard, Victoria Kaial, and Lucas Lorieau. OBLX. Software, swhId: `swh:1:dir:2863e5f060c85b6d04b04c053ab39135abee8f0d` (visited on 2025-12-04). URL: `https://gitlab.limos.fr/oblx/public`, `doi:10.4230/artifacts.25239`.
5. Jona Dirks, Mario Grobler, Roman Rabinovich, Yannik Schnaubelt, Sebastian Siebertz, and Maximilian Sonneborn. PACE Solver Description: PACA-JAVA. In *IPEC 2021*, volume 214 of *LIPIcs*, pages 30:1–30:4, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.IPEC.2021.30`.
6. Enna Gerhard and Sebastian Siebertz. Solving the Directed Feedback Vertex Set Problem in Theory and Practice, 2024. `doi:10.26092/elib/4916`.
7. Ziliang Xiong and Mingyu Xiao. Exactly Solving Minimum Dominating Set and Its Generalization. *IJCAI-24*, 2024. `doi:10.24963/ijcai.2024/780`.