PACE Solver Description

Aslam Ameen
National Institute of Technology Calicut, Kozhikode, India

Ashwin Jacob
National Institute of Technology Calicut, Kozhikode, India

Sahil Muhammed
National Institute of Technology Calicut, Kozhikode, India

Nithin R
National Institute of Technology Calicut, Kozhikode, India

Pankaj Kumar R
National Institute of Technology Calicut, Kozhikode, India

Edwin Thomas
National Institute of Technology Calicut, Kozhikode, India

Abstract

This is a short description of our exact and heuristic solver submitted to the PACE 2025 challenge on the Dominating Set problem [4]. Our approach applies data reduction rules such as isolated vertex and degree-one pruning, followed by exact solving on small components using a combination of brute-force enumeration and ILP formulation.

For larger components, we employ a greedy heuristic that prioritizes vertices based on the number of undominated neighbors in their closed neighborhood. After constructing an initial dominating set, we apply a redundancy removal phase that attempts to eliminate non-critical vertices while preserving coverage. This two-stage approach enables us to produce high-quality dominating sets while maintaining efficiency across a variety of graph instances.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Dominating Set, Data Reduction, Greedy Algorithms, Integer Linear Programming, Exact Algorithms, Heuristics, Graph Decomposition, Brute-force Enumeration, Redundancy Elimination

Supplementary Material Software: `https://github.com/nithinraj04/pace2025_heuristic`

1 Introduction

This document presents our solver submitted to the heuristic track of the 2025 Parameterized Algorithms and Computational Experiments (PACE) challenge on the Dominating Set problem [4]. Our solver combines effective data reduction techniques, exact methods for small components via brute-force and Integer Linear Programming (ILP), and a customization of the standard Greedy Heuristic tailored for larger instances.

In the following, we first define the problem and notation used in Section 2. Then, we describe the reduction rules and the exact strategies used to simplify and solve small components in Section 3. Finally, we detail our heuristic approach in Section 4 and conclude in Section 5.

2 Preliminaries

We use standard graph theoretic terminologies from Diestel's book [1]. We address the classical Dominating Set problem on undirected, unweighted graphs. Given a graph $G = (V, E)$ with vertex set V and edge set $E \subseteq V \times V$, a subset $D \subseteq V$ is called a *dominating* *set* if every vertex $u \in V$ is either in D or has at least one neighbor in D. The objective is to compute a dominating set of minimum cardinality.

Our solver assumes that input graphs are provided in the standard DIMACS format [2] and parses them accordingly. The graph is stored using adjacency lists.

Terminology.

- DOMINATED VERTEX: A vertex is said to be *dominated* if it is either included in the dominating set or has a neighbor in the dominating set.
- COMPONENT: A *component* refers to a maximal connected subgraph of the input graph.
- COVERED NEIGHBORS: For a given vertex u, the *covered neighbors* are the subset of its adjacent vertices that are already dominated.

3 Reduction Rules

We apply a series of graph reductions and exact strategies on each connected component of the input graph before applying our main heuristic. The goal is to simplify the instance, reduce the search space, and extract exact solutions for small subgraphs when feasible.

REDUCTION 1 (ISOLATED VERTEX). Let $v \in V$ be a vertex such that $N(v) = \emptyset$, where $N(v)$ denotes the neighborhood of v. We add v to the solution and remove it from the graph.

The correctness follows since no other vertex can dominate v, it must be included in any valid DOMINATING SET.

REDUCTION 2 (DEGREE-1 LEAF RULE). Let $u \in V$ be a vertex of degree one with neighbor v, i.e., $N(u) = \{v\}$. We include v, mark its closed neighborhood as dominated, and remove it and all its edges from the graph.

In this case, the correctness follows as any dominating set containing u can be replaced by another dominating set of the same size by removing u and (potentially) adding v. The vertex u dominates only u and v while v also dominates u, v and potentially other neighbours of v.

Exact Solving on Small Components. After applying the reduction rules, we look at each connected component of the graph. Each "small-sized" component is solved exactly using different strategies, depending on its size.

EXACT STRATEGY 1 (BRUTE-FORCE ENUMERATION). Let $C \subseteq V$ be a connected component of size at most 30. We enumerate all subsets $D_C \subseteq C$ such that for every $u \in C$, either $u \in D_C$ or $N(u) \cap D_C \neq \emptyset$. Among all such valid DOMINATING SETS, we choose one of minimum size. If some vertices in C were previously fixed to be in the solution, these are included in every candidate.

EXACT STRATEGY 2 (INTEGER LINEAR PROGRAMMING). Let $C \subseteq V$ be a connected component with $30 < |C| \leq 256$. We define a binary variable $x_v \in \{0, 1\}$ for each $v \in C$, where $x_v = 1$ indicates inclusion in the DOMINATING SET. For each vertex $u \in C$, we add the constraint:

$$x_u + \sum_{v \in N(u)} x_v \geq 1,$$

ensuring that u is either selected or dominated by a neighbor. The objective is to minimize $\sum_{v \in C} x_v$. This ILP formulation is solved using the GLPK solver [3]. Vertices already forced into the solution are fixed via constraints $x_v = 1$.

The size thresholds (30 for brute-force and 256 for ILP) were determined empirically to strike a balance between runtime efficiency and exactness of solutions.

4 Heuristic

Following reductions and exact solving for small components, the remaining parts of the graph are handled using a greedy heuristic designed to construct a high-quality dominating set quickly. The heuristic proceeds in two main phases: vertex selection via a dynamic coverage-based priority scheme, and a postprocessing phase to eliminate redundant vertices from the solution.

Greedy Vertex Selection. We iteratively build the dominating set by selecting vertices that offer the most coverage of undominated vertices. To this end, we associate with each vertex $u \in V$ a dynamic score reflecting the number of currently undominated vertices in its closed neighborhood $N[u]$. Specifically, the score of u is defined as:

$$\text{score}(u) = \mathbf{1}_{\{\text{not\_dominated}(u)\}} + |\{v \in N(u) \mid \text{not\_dominated}(v)\}|,$$

where $\mathbf{1}_{\{\cdot\}}$ denotes the indicator function. A vertex $v \in V$ is considered *not_dominated* if neither $v \in D$ nor $N(v) \cap D \neq \emptyset$, where D is the current dominating set.

All vertices with non-zero scores are maintained in a max-priority queue. In each iteration, we extract the vertex with the highest score. If the score is stale due to recent updates in domination status, it is recomputed and reinserted. A vertex u is selected in the dominating set if either:

$$\text{score}(u) \geq 2 \quad \text{or} \quad u \text{ is still undominated.}$$

Upon selection, all vertices in $N[u]$ are marked as dominated. This process continues until the queue is exhausted or all vertices are covered.

This dynamic and coverage-aware selection strategy prioritizes impactful vertices and naturally adapts as the graph becomes progressively dominated.

Redundancy Removal. Once a dominating set has been constructed by the greedy procedure, we perform a redundancy elimination pass to further reduce its size without compromising coverage.

First, for every vertex $v \in V$, we compute the *coverage count*, i.e., the number of vertices in the current dominating set that dominate v. Formally,

$$\text{coverage\_count}(v) = |\{u \in D \mid v \in N[u]\}|,$$

where $D \subseteq V$ is the current dominating set, and $N[u]$ denotes the closed neighborhood of u, i.e., $\{u\} \cup N(u)$.

Then, we assess the necessity of each vertex $u \in D$ based on how critical it is to maintain the coverage of its neighbors.

For each vertex $u \in D$, we calculate an *importance score*, which heuristically reflects how easily its coverage can be compensated by others. Vertices that do not uniquely dominate any other vertex are considered candidates for removal. Among these, we prioritize those whose neighbors are highly redundant (e.g., already covered by many other dominating set members), using the following scoring heuristic:

$$\text{importance}(u) = \sum_{v \in N(u)} \phi(\text{coverage\_count}(v)), \quad \text{where} \quad \phi(k) = \begin{cases} 0 & \text{if } k = 1 \\ 0.5 & \text{if } k = 2 \\ \frac{1}{k} & \text{if } k > 2 \end{cases}$$

Vertices with the lowest importance scores are considered first for removal. A vertex u is removed from the dominating set only if its removal does not leave any vertex $v \in N[u]$ with coverage count ≤ 1, ensuring that all vertices remain dominated. The coverage counts are then updated accordingly.

This postprocessing step effectively eliminates redundancies and fine-tunes the solution toward minimality.

5 Conclusion

In summary, our solver combines effective reduction rules with exact and heuristic methods to tackle the Dominating Set problem efficiently. By applying local degree-based reductions and solving small connected components exactly via brute-force or integer linear programming, we are able to significantly simplify the input before applying our main heuristic.

The heuristic employs a dynamic, coverage-based vertex selection strategy, followed by a redundancy elimination phase to refine the solution. This combination of reductions, exact subroutines, and scalable heuristics allows the solver to maintain a strong balance between solution quality and computational efficiency across a wide range of instance sizes and structures.

References

1 Reinhard Diestel. Graph theory, volume 173 of. *Graduate texts in mathematics*, 593, 2012.

2 DIMACS. DIMACS graph format specification. `http://dimacs.rutgers.edu/archive/Challenges/`, 1993. Accessed: 2025-06-21.

3 GNU Project. GLPK (gnu linear programming kit). `https://www.gnu.org/software/glpk/`, 2023. Accessed: 2025-06-21.

4 PACE Steering Committee. The PACE 2025 parameterized algorithms and computational experiments challenge: Dominating set. `https://pacechallenge.org/2025/`, 2025. Accessed: 2025-06-21.