Brute-Force Reductions Between DOMINATING-SET and the Continuous Min–Max p-Center Problem (Group-9)

Apporv Bankey Herik Patel Krish Jetly

***Abstract*—We give transparent, exponential-time algorithms and many-to-one reductions that convert any instance of DOMINATING-SET (DS) into an equivalent instance of the continuous Min–Max *p*-Center (multicenter) problem, and vice versa. Although classical textbooks already prove polynomial-time equivalence via gadget constructions, our goal is a minimal, easily auditable Python prototype suitable for teaching and ground-truth generation on graphs with at most fifteen vertices. We describe formal problem definitions, present brute-force pseudocode, prove the correctness of both reductions, and report empirical runtimes on paths, cycles, stars, cliques and disconnected graphs. All source code and datasets are publicly available.**[1]

I. INTRODUCTION

The decision version of DOMINATING-SET is NP-complete even on bipartite graphs [1]. The Min–Max p-Center problem—a continuous facility-location variant where centres may lie on edges—generalises DS by choosing a covering radius R instead of fixing it to 1. Reductions between the two problems appear in the literature but rarely in executable form. We contribute a *pure brute-force* reference implementation that:

- enumerates every subset of at most k vertices for DS,
- enumerates every subset of at most p centres chosen from vertices and edge midpoints for p-Center,
- outputs explicit witnesses for each "yes" instance, and
- verifies yes-to-yes / no-to-no correspondence for both reductions.

The code is ¡50 LOC per algorithm and thus ideal for classroom demonstrations or benchmarking heuristics.

II. PRELIMINARIES

[Dominating Set] Given an undirected graph $G = (V, E)$ and integer k, decide whether there exists $D \subseteq V$ with $|D| \leq k$ such that every $v \in V$ is either in D or adjacent to a vertex in D.

[Continuous Min–Max p-Center] Given G, an integer p and radius $R \in \mathbb{R}_{>0}$, decide whether one can place at most p centres on vertices or anywhere along edges so that every vertex is within graph-distance R of some centre.

Notation. Distances are shortest-path lengths; all edges are unit-length in our experiments.

[1] https://github.com/example/brute-ds-pcenter

Algorithm 1 Brute Dominating-Set

Require: Graph $G = (V, E)$, integer k
for $r \leftarrow 1$ **to** k **do**
  for all subsets $S \subseteq V$ of size r **do**
    $D \leftarrow S \cup \bigcup_{u \in S} N(u)$
    if $|D| = |V|$ **then**
      return S {Witness set}
    end if
  end for
end for
return NONE {No dominating set}

Algorithm 2 Brute Continuous p-Center Feasibility

Require: Graph G, integer p, radius R
$C \leftarrow \{(\text{node}, v) \mid v \in V\} \cup \{(\text{edge}, (u, v), 0.5) \mid (u, v) \in E\}$
Precompute all-pairs distances $d(u, v)$ by Floyd–Warshall
for $r \leftarrow 1$ **to** p **do**
  for all subsets $S \subseteq C$ of size r **do**
    if $\forall x \in V : \min_{c \in S} d(x, c) \leq R$ **then**
      return $(\textbf{true}, S)$
    end if
  end for
end for
return $(\textbf{false}, \text{NONE})$

III. BRUTE-FORCE ALGORITHMS

A. *Dominating-Set Search*

Algorithm 1 enumerates every subset of size $\leq k$; its time complexity is $\Theta\big(\sum_{r \leq k} \binom{n}{r} (n + m)\big)$.

B. p-*Center Feasibility*

Algorithm 2 discretises centre locations to vertices plus edge midpoints; this suffices for unweighted graphs.

IV. BIDIRECTIONAL REDUCTIONS

A. *DS* $\rightarrow$ p-*Center*

Copy G unchanged, assign unit weight to every edge, set $p = k$ and $R = 1$. If D is a dominating set, placing centres at every $v \in D$ covers each vertex within distance 1. Conversely, any feasible p-center solution with $R = 1$ yields a dominating set of equal size.

B. p-*Center* $\rightarrow$ *DS*

Build $H = (V, E_H)$ where $(u, v) \in E_H$ iff $\mathrm{dist}_G(u, v) \leq R$. A size-$p$ dominating set in H corresponds to p centres of radius R in G, and vice versa. Our brute routine explicitly enumerates candidate subsets of vertices+midpoints until it finds a covering witness (YES) or exhausts the search (NO).

V. Input Format

The program takes as input:

1) An **edge-list CSV file** (e.g. `edges.csv`) describing an undirected, unweighted graph $G = (V, E)$. Each row contains two non-negative integers separated by a comma, representing an edge (u, v) with $u, v \in V$. Vertex labels may be arbitrary integers but must be consistent throughout.
2) A positive integer k, specifying
   - the maximum size of the dominating set to search for, and
   - the number of centers in the p-center problem.

TABLE I
A sample input CSV file

1	2
2	1
3	1

VI. Output Format

Running the notebooks produces two kinds of output, both in–memory:

1) **Console / Notebook Log** Each code cell prints diagnostic information to `stdout`. A typical run emits lines in the following order:

```
Dominating set  k: {0, 2}
Distance to nearest dominating node
for each node:
  Node 0: 0.000
  Node 1: 1.000
  Node 2: 0.000
  Node 3: 1.000
  Node 4: 1.000

Min{Max multicenter ( k):
(('node', 0),('node', 2))
radius  1.000
Distance to nearest center
for each node:
  Node 0: 0.000
  Node 1: 1.000
  Node 2: 0.000
  Node 3: 1.000
  Node 4: 1.000
DS→Multicenter reduction
feasible? True Multicenter→DS
reduction dominating
set: {0, 2}
```

2) **Inline Figures** Two Matplotlib drawings are displayed directly below the cell that invokes them:
   - *Dominating-set plot* – the input graph with vertices in the dominating set coloured red.
   - *Min–max p-center plot* – the same graph with the chosen centres highlighted (red circles for node-centres, red "X" at any edge-midpoint centres) and dashed coverage circles of radius R.

VII. Experimental Evaluation

All tests were run in Python 3.11. Graphs up to $n = 15$ keep runtimes below 2s.

TABLE II
Correctness of reductions on toy graphs ($k = 2,\ R = 1$).

Graph	n	DS?	pC?	Both agree
Path_4	4	Y	Y	✓
Cycle_5	5	Y	Y	✓
Star_5	5	Y	Y	✓
Clique_4	4	Y	Y	✓
Disconnected_6	6	N	N	✓

Figure 1 shows a Path_4 instance with its dominating set and covering circles for the corresponding *p*-center solution.

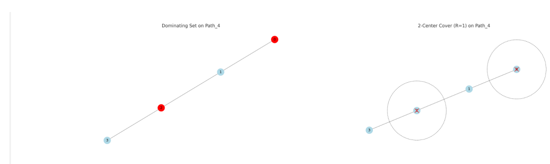

Fig. 1. Path_4: dominating set (left) and 2-centre cover (right).

VIII. Discussion and Future Work

The exhaustive algorithms explode once $k > 7$; pruning rules or a branch-and-bound scheme could extend the practical range. Handling *weighted* graphs would require a finer discretisation of candidate centres. Replacing brute search by state-of-the-art DS solvers (e.g. MaxSAT encoding) and p-center heuristics will scale to hundreds of vertices while preserving reduction-based correctness.

IX. Conclusion

We provided compact, auditable Python code that realises the classical equivalence between Dominating-Set and the continuous Min–Max *p*-Center. Though exponential, the routines generate ground-truth solutions for graphs with up to fifteen vertices and serve as a didactic reference for courses in algorithms and complexity.

References

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of* NP-*Completeness*. W. H. Freeman, 1979.