# Exhaustive Search & Greedy Heuristic Algorithms for Finding the Minimum Dominating Set of a Graph

Rodrigo Abreu
Mestrado em Engenharia Informática

***Abstract* – Given an undirected graph $G = (V, E)$, this work implements and analyzes two algorithms for the Minimum Dominating Set problem: an exact exhaustive search and an approximate greedy heuristic. A formal complexity analysis is contrasted with a systematic experimental analysis on graphs ($n = 4$ to $n = 40$) of varying size and edge density. The experimental results confirm the theoretical exponential complexity of the exhaustive search, but critically reveal that its practical performance is dictated by graph density, with performance varying by over 1,000,000x for the same $n$. The greedy heuristic demonstrated exceptional efficiency and solution quality, achieving a mean precision of 95.9% and finding the provably optimal solution in 85.1% of instances. A key finding is the significant, explained discrepancy between the greedy algorithm's $O(n^3)$ formal worst-case complexity and its experimentally-fitted $O(n^{1.42})$ average-case performance.**

***Keywords* – Minimum Dominating Set, Exhaustive Search, Greedy Heuristic, Computational Complexity, Formal Analysis, Experimental Analysis, Average-Case vs Worst-Case**

## I. Introduction

The Minimum Dominating Set (MDS) problem seeks to determine, in an undirected graph, the smallest subset of vertices such that every vertex in the graph is either included in this subset or adjacent to at least one vertex from it. This paper focuses on comparing two approaches to solving the MDS problem: an exhaustive search algorithm, which guarantees the optimal solution by exploring all possible subsets but suffers from exponential growth in computational cost, and a greedy heuristic algorithm, which constructs a dominating set iteratively based on a local selection criterion, offering a faster but potentially suboptimal solution. The objective is to analyze their performance experimentally and discuss how graph size and density affect solution quality and execution time.

## II. The Problem – Minimum Dominating Set

The *Minimum Dominating Set (MDS)* problem consists of finding, in a given undirected graph $G(V, E)$ with $n$ vertices and $m$ edges, the smallest subset of vertices $D \subseteq V$ such that every vertex in the graph is either a member of $D$ or adjacent to at least one vertex in $D$. In other words, every vertex in $V \setminus D$ must have at least one neighbor belonging to $D$. A subset of vertices satisfying this property is called a *dominating set*, and among all possible dominating sets, the goal is to identify the one with the *minimum cardinality*.

The MDS problem is a classical example of an *NP-hard* combinatorial optimization problem, meaning that the number of possible vertex subsets grows exponentially with the size of the graph [1]. Consequently, *exact algorithms* such as exhaustive search can guarantee the optimal solution but are only feasible for small graphs due to their high computational complexity. In contrast, *heuristic methods*, such as greedy algorithms, aim to find near-optimal dominating sets more efficiently by making locally optimal choices at each step, enabling the treatment of larger graph instances at the expense of potential solution quality [2], [3].

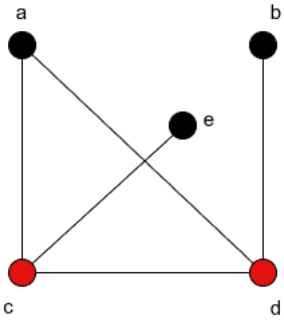


Fig. 1: Example of a 5 vertex graph that has the vertex of a possible minimum dominating set colored in red.

## III. Algorithms and Formal Analysis

### A. Exhaustive Search

The first version of the exhaustive search algorithm explores all possible subsets of vertices in the graph to identify the smallest dominating set. Since every possible subset is tested, this approach always finds the optimal solution. However, as the number of vertices increases, the algorithm quickly becomes impractical due to the exponential number of subsets that must be examined.

**Algorithm 1** Pure Exhaustive Search for the Minimum Dominating Set

**Input:** Undirected graph $G = (V, E)$
**Output:** Minimum dominating set $D^*$ and its size $|D^*|$

$D^* \leftarrow \emptyset$, $minSize \leftarrow \infty$ **for** $r \leftarrow 1$ **to** $|V|$ **do**
  **for** *each subset* $S \subseteq V$ *of size* $r$ **do**
    $isDominating \leftarrow$ **True** **for** *each vertex* $v \in V$ **do**
      **if** $v \notin S$ ***and*** $N(v) \cap S = \emptyset$ **then**
        $isDominating \leftarrow$ **False** **break**
      **end**
    **end**
    **if** *isDominating* **then**
      $D^* \leftarrow S$ $minSize \leftarrow |S|$ **return** $(D^*, minSize)$
    **end**
  **end**
**end**
**return** $(D^*, minSize)$

This approach serves as a baseline for comparison with more efficient algorithms. It provides a guaranteed optimal solution, which can later be used to assess the quality of approximate solutions generated by heuristic methods such as the greedy algorithm.

*Time Complexity Analysis*

The time complexity analysis depends critically on the size of the *actual* minimum dominating set, let's call it $k = |D^*|$.

The implemented algorithm (Algorithm 1) is optimized: it iterates through subset sizes $r = 1, 2, \ldots, k$ and stops as soon as the first (and therefore minimum) dominating set is found.

For each size $r$, it tests $\binom{n}{r}$ candidate subsets. The verification of each subset $S$ (the 'is_dominating_set' check) requires examining each of the $n$ vertices and, in the worst case, scanning its adjacency list. This check costs $O(n \cdot d_{\text{avg}})$ (or $O(n^2)$ for dense graphs).

The total number of operations is dominated by the last step (when $r = k$):

$$T_{\text{exh}}(n, k) = O\left(\sum_{r=1}^{k} \binom{n}{r} \cdot (n \cdot d_{\text{avg}})\right)$$

Since the sum is dominated by its last and largest term, $\binom{n}{k}$, a tighter bound is:

$$T_{\text{exh}}(n, k) = O\left(\binom{n}{k} \cdot n \cdot d_{\text{avg}}\right)$$

Using the approximation $\binom{n}{k} \leq n^k$, this gives:

$$T_{\text{exh}}(n, k) = O(n^k \cdot n \cdot d_{\text{avg}}) = O(n^{k+1} \cdot d_{\text{avg}})$$

This analysis, while still exponential if $k$ is not constant (e.g., $k = n/2$), more accurately reflects the algorithm's optimized performance. A looser, "pessimistic" upper bound, which assumes all $2^n$ subsets are checked, would be $O(2^n \cdot n^2)$.

*B. Greedy Heuristic for the Minimum Dominating Set*

The greedy heuristic provides an efficient approximate solution to the Minimum Dominating Set problem by iteratively selecting vertices based on their coverage potential. At each step, the algorithm chooses the vertex that dominates the largest number of currently undominated vertices. This process continues until all vertices in the graph are dominated. Although it does not guarantee an optimal solution, it significantly reduces computational cost compared to exhaustive search while producing near-optimal results for many graph instances.

**Algorithm 2** Greedy Heuristic

**Input:** Undirected graph $G = (V, E)$
**Output:** Dominating set $D$ and its size $|D|$

$D \leftarrow \emptyset$ $dominated \leftarrow \emptyset$ $U \leftarrow V$ ; `// Set of all vertices`
**while** $dominated \neq U$ **do**
  $best\_vertex \leftarrow$ **None** $max\_new\_dominated \leftarrow -1$
  **for** $v \in U \setminus D$ **do**
    $new\_dominated \leftarrow (N(v) \cup \{v\}) - dominated$
    **if** $|new\_dominated| > max\_new\_dominated$ **then**
      $best\_vertex \leftarrow v$ $max\_new\_dominated \leftarrow |new\_dominated|$
    **end**
  **end**
  $D \leftarrow D \cup \{best\_vertex\}$ $dominated \leftarrow dominated \cup N(best\_vertex) \cup \{best\_vertex\}$
**end**
**return** $(D, |D|)$

*Time Complexity Analysis*

Let $n = |V|$, $d_{\text{avg}}$ be the average degree, and $k$ be the size of the final dominating set found by the heuristic (where $k \leq n$).

The algorithm proceeds in $k$ iterations of the outer `while` loop (one for each vertex added to $D$).

In each iteration, the algorithm must find the 'best vertex'. This is done by an inner `for` loop that iterates over $O(n)$ vertices.

The critical operation inside this loop is calculating the number of newly dominated vertices: `new_dominated ← (N(v) ∪ {v}) - dominated`.

The cost of this operation is not constant. The set-difference operation `new_dominated - dominated` depends on the size of the 'dominated' set, which grows from 0 to $O(n)$. Therefore, the cost of this line is $O(d_v + |dominated|)$, which is $O(d_{\text{avg}} + n)$ on average.

The total cost of the inner loop (to find one 'best vertex') is $n$ iterations $\times$ $O(d_{\text{avg}} + n)$ per iteration.

$$T_{\text{inner\_loop}} = O(n \cdot (d_{\text{avg}} + n)) = O(n \cdot d_{\text{avg}} + n^2)$$

The total complexity is the cost of the inner loop multiplied by the $k$ iterations of the outer loop:

$$T_{\text{greedy}}(n, k) = O(k \cdot (n \cdot d_{\text{avg}} + n^2))$$

In the worst case, $k = n$ (e.g., a path graph where each vertex added only dominates one new vertex). This leads to a total time complexity of:

$$T_{\text{greedy}}(n) = O(n \cdot (n \cdot d_{\text{avg}} + n^2)) = O(n^2 \cdot d_{\text{avg}} + n^3)$$

For both sparse graphs ($d_{\text{avg}} = O(1)$) and dense graphs ($d_{\text{avg}} = O(n)$), this complexity is dominated by the $O(n^3)$ term.

## IV. Computational Tests

### A. Experimental Setup

All computational tests were performed on a Lenovo LOQ Essential 15IAX9E with Intel Core I7 of the 12th generation CPU and 16 GB RAM. The Operative System used was Ubuntu 24.04.

### B. Graph Generation

The test graphs used in the experiments were generated using a custom Python script developed with the NetworkX library. Each graph is undirected, unweighted, and embedded in a two-dimensional plane. The generation process was designed to produce reproducible and structurally diverse instances by varying both the number of vertices and the edge density.

Each vertex was assigned random coordinates $(x, y)$ within a $500 \times 500$ area, with a minimum Euclidean distance of ten units enforced between any two vertices to prevent overlap. The number of edges in each graph depended on a target density value $d \in \{12.5\%, 25\%, 50\%, 75\%\}$, representing the fraction of all possible edges to be included. Edges were then selected uniformly at random from all possible vertex pairs until the desired density was reached.

After the edge assignment phase, the algorithm verified whether the generated graph was connected. If disconnected, additional edges were added between components until a single connected component was obtained. This step ensured that each instance contained a valid and meaningful dominating set. To guarantee reproducibility, all random number generators were initialized with the student's identification number (`113626`) as the seed.

For each density level, graphs with increasing numbers of vertices were generated, ranging from $n = 4$ to $n = 40$. Each instance was stored both as a human-readable `.json` file, containing the vertex coordinates and edge list, and as a visualization image for reference. These graph instances were later used as inputs for both the exhaustive search and greedy heuristic algorithms in the experimental evaluation.

### C. Experimental Results

The following sections present the experimental results obtained from systematic testing of both the exhaustive search and greedy heuristic algorithms on randomly generated graphs ranging from $n = 4$ to $n = 40$ vertices, with four different edge density levels (12.5%, 25%, 50%, and 75%).

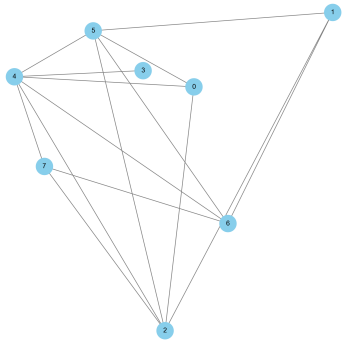


Fig. 2: Example of a graph generated, with 8 vertex and 50% edge density.

### D. Execution Time Analysis

Figure 3 presents the execution time comparison between exhaustive search and greedy heuristic algorithms across all tested configurations. The results are displayed on a logarithmic scale to accommodate the dramatic differences in performance between the two approaches.

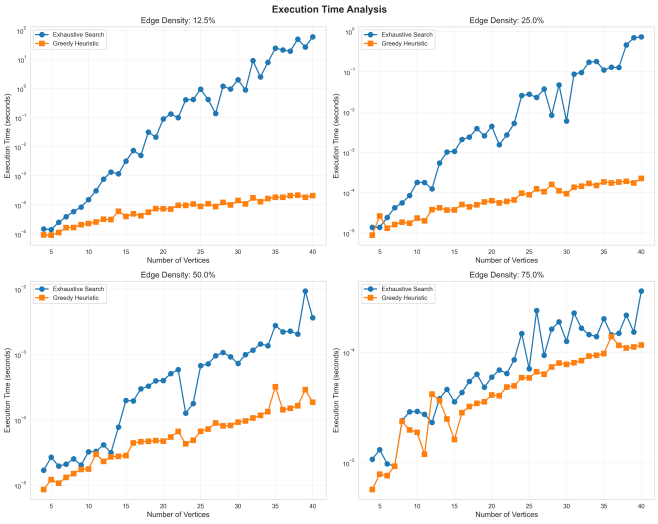


Fig. 3: Execution time comparison between exhaustive search and greedy heuristic algorithms for different graph sizes and edge densities. Note the logarithmic scale on the y-axis, highlighting the exponential growth of exhaustive search versus the polynomial growth of the greedy approach.

#### D.1 Exhaustive Search Performance

The exhaustive search algorithm exhibits clear exponential growth in execution time as the number of vertices increases. While the algorithm remains practical for small graphs ($n \leq 10$), with execution times under 0.2 ms, its performance degrades rapidly. For medium graphs, such as $n = 20$ (12.5% density), the time grows to 89.5 ms. This trend culminates in impractical execution times for larger instances, reaching 59.8 seconds

for $n = 40$ at the same 12.5% density, demonstrating its impracticality for large-scale problems.

Interestingly, the execution time is heavily influenced by edge density, with higher densities (75%) resulting in faster execution times than sparse graphs (12.5%). This counter-intuitive result occurs because denser graphs tend to have smaller dominating sets, which the optimized algorithm finds earlier in its search. This effect is dramatic: at $n = 30$, the 75% density graph completed in just 0.125 ms, while the 12.5% density graph required 1.99 seconds, a difference of approximately $16,000\times$.

### *D.2 Greedy Heuristic Performance*

In stark contrast, the greedy heuristic maintains consistently low execution times across all graph sizes. The algorithm exhibits a smooth, polynomial growth pattern, which is consistent with its polynomial-time formal complexity. This demonstrates its excellent scalability; even for the largest tested graph ($n = 40$), the execution time remained under 1 ms (at 0.225 ms). Unlike the exhaustive search, edge density had minimal impact on the greedy algorithm's performance, with all four density levels producing similar performance curves. The total execution time ranged from approximately 5.7 µs at $n = 4$ to only 225 µs at $n = 40$, representing a modest 39× increase across the entire tested range.

### *D.3 Comparative Analysis*

Table I presents selected execution time comparisons at key graph sizes for the 50% edge density configuration, illustrating the diverging performance characteristics of both algorithms.

TABLE I: Execution time comparison at selected graph sizes (50% edge density)

| n | Exhaustive (s) | Greedy (s) | Speedup |
|---|---|---|---|
| 10 | $3.24 \times 10^{-5}$ | $1.76 \times 10^{-5}$ | 1.84× |
| 20 | $3.99 \times 10^{-4}$ | $4.72 \times 10^{-5}$ | 8.45× |
| 30 | $7.30 \times 10^{-4}$ | $9.20 \times 10^{-5}$ | 7.93× |
| 40 | $3.64 \times 10^{-3}$ | $1.85 \times 10^{-4}$ | 19.68× |

The data confirms that while both algorithms are practical for small graphs, only the greedy heuristic remains viable for larger problem instances. The speedup factor, while modest for small graphs, demonstrates accelerating growth as problem size increases, consistent with the exponential-polynomial complexity gap between the approaches.

### *E. Number of Operations Analysis*

Figure 4 illustrates the number of basic operations performed by each algorithm, providing a hardware-independent measure of computational work. Crucially, the definition of an "operation" differs for each algorithm, reflecting their distinct computational structures.

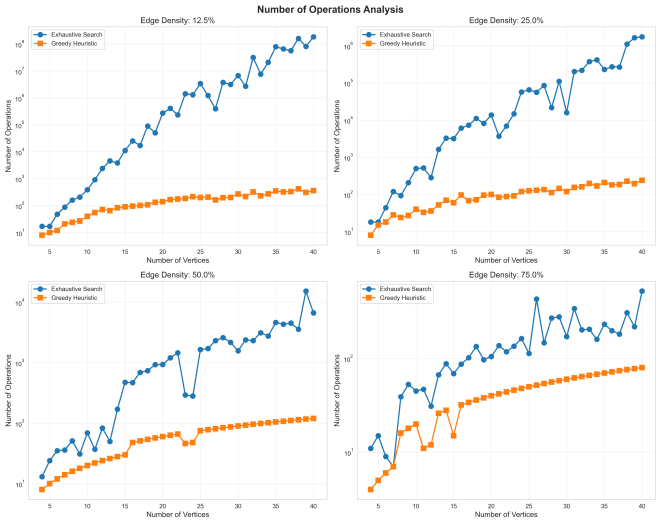


Fig. 4: Number of basic operations for exhaustive search and greedy heuristic algorithms. The logarithmic scale emphasizes the exponential versus polynomial growth patterns.

### *E.1 Exhaustive Search Operations*

For the exhaustive search, an "operation" is defined as a single vertex check within the `is_dominating_set` function. This provides a fine-grained measure of the total work performed to validate all candidate sets. The plot of these operations shows clear exponential growth, mirroring the execution time. For the 12.5% density configuration, the operation count grew from 17 ($n = 4$) to a total of 185,881,809 operations at $n = 40$. This operation count (total work) is distinct from the number of configurations tested. For instance, at $n = 40$ (12.5%), 185.9 million operations were required to check 48.4 million candidate sets. This implies that, on average, each invalid candidate was rejected very quickly (after $\approx 3.84$ vertex checks), while only the final, valid dominating set required all $n = 40$ checks. The impact of density on this metric is dramatic. At $n = 38$, the sparse 12.5% graph (with a large MDS, $k = 8$) required 162.1 million operations. In contrast, the dense 75% graph (with a small MDS, $k = 2$) required only 308 operations. This difference of over $500,000\times$ confirms that the size of the optimal solution $k$ is the primary driver of practical performance, as it dictates how many $\binom{n}{r}$ levels the algorithm must explore.

### *E.2 Greedy Heuristic Operations*

For the greedy heuristic, an "operation" is defined as the evaluation of a single vertex's "coverage potential" within the inner loop (i.e., one iteration of the `for v in all_nodes` loop). This counter measures the product of the final solution size ($k$) and the number of vertices ($n$), as the algorithm checks all $n$ vertices for each of the $k$ times it adds a vertex to the solution. This $O(k \cdot n)$ relationship is clearly visible in the data. For validation, at $n = 40$ (12.5% density), the algo-

rithm found a solution of size $k = 9$ and performed $9 \times 40 = 360$ operations. Similarly, for the 75% density graph, it found $k = 2$ and performed $2 \times 40 = 80$ operations, perfectly matching the $k \times n$ formula. Since $k$ grows very slowly (sub-linearly) with $n$ in these random graphs, the operation count grows in a near-linear or mild polynomial fashion. This must be contrasted with the $O(n^3)$ *time* complexity. The discrepancy is explained because the "operation" being counted here (vertex evaluation) is not an $O(1)$ constant-time task; its cost, which involves a set-difference operation, grows with $n$, accounting for the higher polynomial in the execution time.

### *E.3 Comparative Analysis*

The operational disparity revealed by this hardware-independent measure is immense. At $n = 40$ (12.5%), the exhaustive search performed 185.9 million vertex-check operations, while the greedy heuristic performed only 360 candidate-evaluation operations. This confirms the vast asymptotic superiority of the greedy approach, showing it performs over 500,000 times fewer "basic operations" for this specific instance.

### *F. Configurations Tested (Exhaustive Search)*

Figure 5 shows the number of candidate configurations (subsets) examined by the exhaustive search algorithm before finding the optimal solution. This metric provides a direct measure of the search space explored.

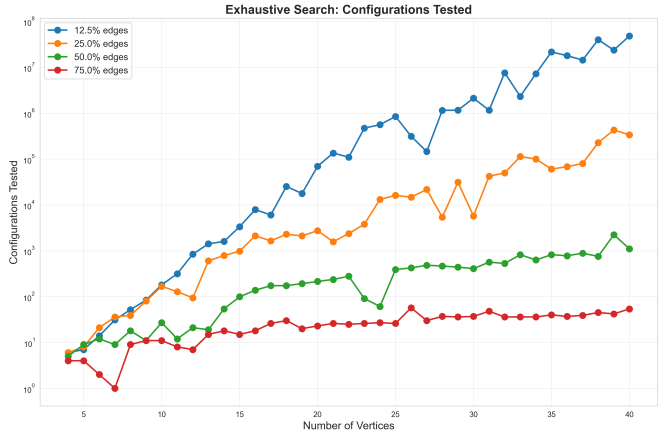


Fig. 5: Number of configurations tested by exhaustive search across different edge densities. The exponential growth is evident, with sparse graphs requiring significantly more evaluations than dense graphs.

### *F.1 Analysis of Configuration Space Exploration*

As predicted by the formal analysis, the number of configurations tested grows exponentially with $n$. However, the plot clearly shows that this growth is dominated by the graph's edge density. Dense graphs exhibit dramatically fewer configurations tested. This is because dense connectivity results in smaller minimum dominating sets (a smaller domination number, $k$), as vertices have higher coverage. Since the algorithm tests subsets in increasing size order ($r = 1, 2, \ldots, k$), finding a solution with a small $k$ requires examining far fewer configurations.

This effect is most pronounced at $n = 40$: for the 12.5% (sparse) density, the solution size was $k = 8$ and the algorithm tested 48,433,527 configurations, whereas for the 75.0% (dense) density, the solution size was $k = 2$ and it tested only 54 configurations. This difference of nearly 900,000-fold demonstrates the profound impact of graph structure on exhaustive search performance.

The number of configurations tested can be quantified as $\sum_{r=1}^{k-1} \binom{n}{r} + C_k$, where $C_k$ is the number of size-$k$ subsets tested before the first solution is found ($1 \leq C_k \leq \binom{n}{k}$). This sum is substantially smaller than the theoretical maximum of $2^n$ subsets, explaining why dense graphs with small $k$ values are processed orders of magnitude faster.

### *G. Greedy Heuristic Precision Analysis*

Precision quantifies the solution quality of the greedy heuristic by comparing its output, $|D_{\text{greedy}}|$, to the optimal solution size, $\gamma(G)$. The ratio is defined as:

$$\text{Precision} = \frac{\gamma(G)}{|D_{\text{greedy}}|} \tag{1}$$

A score of 1.0 indicates a perfect, optimal solution. Figure 6 presents the analysis across all tested instances.

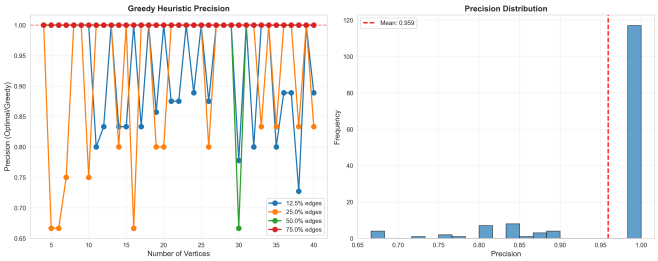


Fig. 6: Greedy heuristic precision analysis. Left: Precision versus number of vertices for different edge densities. Right: Overall precision distribution showing that 95.9% average precision was achieved.

### *G.1 Analysis of Results*

The greedy heuristic demonstrated excellent approximation quality, achieving a mean precision of **0.959 (95.9%)** across all 148 test instances. As shown in Figure 6 (right), the algorithm found the provably optimal solution (precision = 1.0) in **85.1%** of cases (126 out of 148).

A key finding was the strong correlation between precision and edge density, as summarized in Table II. The heuristic was **100% optimal** for every graph with 75% density and near-perfect (97.3% optimal) for 50% density. This suggests that for dense graphs, the

locally optimal choice is almost always the globally optimal one.

TABLE II: Greedy heuristic precision by edge density

| Density | Mean | Optimal Found | Min |
|---|---|---|---|
| 12.5% | 0.935 | 25/37 (67.6%) | 0.727 |
| 25.0% | 0.953 | 28/37 (75.7%) | 0.667 |
| 50.0% | 0.982 | 36/37 (97.3%) | 0.667 |
| 75.0% | 1.000 | 37/37 (100%) | 1.000 |

Conversely, the few sub-optimal solutions occurred on sparser graphs (12.5% and 25.0%), which present a more challenging landscape for the greedy strategy. The worst-observed precision was 0.667 (a solution of size 3 vs. an optimal of 2), confirming that even the worst-case approximations remained practically reasonable.

### *H. Theoretical versus Experimental Complexity*

This section compares the formal complexity bounds developed in Section 3 with the experimental data. Figure 7 presents complexity models fitted to the 50% edge density data, which is used as a representative case to derive empirical growth functions.

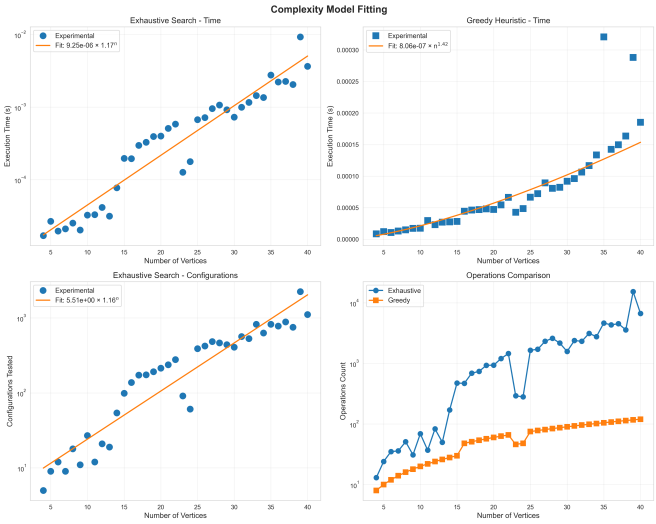


Fig. 7: Complexity model fitting for both algorithms using 50% edge density data. The fitted models are used to compare empirical performance against theoretical bounds.

### *H.1 Exhaustive Search: A Clear Exponential Match*

The formal analysis identified a pessimistic bound of $O(2^n \cdot n^2)$ and a tighter, implementation-specific bound of $O(n^{k+1} \cdot d_{\text{avg}})$, where $k$ is the size of the minimum dominating set. An exponential model $T(n) = a \cdot b^n$ was fitted to the experimental data, yielding $T_{\text{exh}}(n) \approx (9.25 \times 10^{-6}) \cdot 1.17^n$. The experimental data strongly confirms the formal analysis, as both show the algorithm is in the exponential complexity class.

The experimental base $b \approx 1.17$ is substantially smaller than the pessimistic theoretical base $b = 2$. This is perfectly explained by our optimized algorithm, which stops at the first solution $k$. For 50% density graphs, $k$ is very small (e.g., $k = 2$ or $k = 3$ for most instances). The search space explored, $\sum_{r=1}^{k} \binom{n}{r}$, is therefore vastly smaller than $2^n$ when $k$ is a small constant, resulting in a much smaller effective growth base.

### *H.2 Greedy Heuristic: A Key Discrepancy*

This analysis provides the most critical finding of the report. The formal worst-case complexity of our implementation was found to be $O(n^2 \cdot d_{\text{avg}} + n^3)$, which simplifies to $O(n^3)$ for 50% dense graphs. However, a polynomial model $T(n) = a \cdot n^c$ fitted to the experimental data yielded $T_{\text{greedy}}(n) \approx (8.06 \times 10^{-7}) \cdot n^{1.42}$. The experimental result $O(n^{1.42})$ does not match the formal worst-case analysis $O(n^3)$. This significant discrepancy is not an error, but rather highlights the crucial difference between worst-case and average-case behavior. The $O(n^3)$ bound assumes a worst-case where the solution size $k$ approaches $n$. Our 2D random graphs represent an average case, where the algorithm finds a very small solution $k$ (e.g., $k = 3$ at $n = 40$). In this scenario, the true complexity $O(k \cdot n^2)$ behaves more like $O(n^2)$ since $k$ is small. Furthermore, the experimental exponent $c \approx 1.42$ is even lower because the test range ($n \leq 40$) is too small for the true asymptotic behavior to dominate constant factors, making the $O(n^{1.42})$ a local fit for this small domain.

### *H.3 Complexity Comparison Summary*

Tables III and IV summarize the comparison between theoretical predictions and the new experimentally fitted models.

TABLE III: Exhaustive Search complexity comparison.

| Metric | Formal (Worst-Case) | Experimental Fit |
|---|---|---|
| Time | $O(2^n \cdot n^2)$ | $O(1.17^n)$ |
| Configs | $O(\binom{n}{k})$ | $O(1.16^n)$ |
| **Match?** | (Complexity Class) | |

TABLE IV: Greedy Heuristic complexity comparison.

| Metric | Formal (Worst-Case) | Experimental Fit |
|---|---|---|
| Time | $O(n^3)$ | $O(n^{1.42})$ |
| **Match?** | **No (Discrepancy)** | |

The results for the exhaustive search (Table III) confirm its exponential nature, with the small base $b \approx 1.17$ being a direct consequence of the algorithm's optimization and the small $k$-values of dense graphs.

For the greedy heuristic (Table IV), the experiment shows that its average-case performance on these graphs is excellent (sub-quadratic), even though its theoretical worst-case is cubic. This divergence be-

tween theoretical worst-case and practical average-case performance is a key takeaway.

### *I. Determining the Practical Limit of Exhaustive Search*

A key objective was to determine the largest graph size $n$ that the exhaustive search algorithm could process "without taking too much time." We define this practical threshold as an execution time of 60 seconds.

Based on the experimental results, this limit is not a single number but is instead dictated entirely by the graph's edge density. Our analysis script fitted an exponential model to the 12.5% density (sparse) graphs, which represent the worst-case scenario in our experiment, yielding the growth model $T_{\text{worst}}(n) \approx (1.17 \times 10^{-3}) \cdot 1.309^n$.

Using this model to find the largest $n$ that remains under our 60-second threshold, the script determined the practical limit to be $n = 40$ vertices, with an estimated time at that limit of 56.04 seconds. This confirms that for any graph of unknown or sparse structure, $n = 40$ is the practical processing limit on the test hardware. In contrast, the model for 75% density graphs ($T(n) \approx (2.07 \times 10^{-5}) \cdot 1.065^n$) shows that dense graphs could be processed in milliseconds, even at this size.

### *J. Estimation for Larger Problem Instances*

We use the empirical models fitted by the analysis script to estimate the execution time for much larger problem instances.

#### *J.1 Exhaustive Search Extrapolation*

The impact of density is profound. To demonstrate this, we fitted exponential models $T(n) = a \cdot b^n$ to our best, average, and worst-case data. As shown in Table VI, the exponential base $b$ for the sparse graph ($b \approx 1.309$) is dramatically larger than for the dense graph ($b \approx 1.065$), leading to a catastrophic failure of scalability.

For a graph with $n = 60$, the estimated time ranges from 0.89 milliseconds in the best-case (dense) to 3.41 hours in the worst-case (sparse), illustrating that performance is entirely dependent on graph structure.

TABLE V: Fitted Exponential Models ($T(n) = a \cdot b^n$) for Exhaustive Search

| Case | Fitted Model |
|---|---|
| Worst (12.5%) | $T(n) \approx (1.17 \times 10^{-3}) \cdot 1.309^n$ |
| Average (50.0%) | $T(n) \approx (4.59 \times 10^{-6}) \cdot 1.194^n$ |
| Best (75.0%) | $T(n) \approx (2.07 \times 10^{-5}) \cdot 1.065^n$ |

#### *J.2 Greedy Heuristic Extrapolation*

For the greedy heuristic, we use the average-case fitted model $T_{\text{greedy}}(n) \approx (8.06 \times 10^{-7}) \cdot n^{1.42}$, which was derived from the 50% density data. As shown in Table VII, these estimates demonstrate the algorithm's exceptional scalability.

TABLE VI: Extrapolated Execution Time for Exhaustive Search

| Time | Wst. Case | Avg. Case | Best Case |
|---|---|---|---|
| $T(45)$ | 3.59 min | 13.39 ms | 0.35 ms |
| $T(50)$ | 13.82 min | 32.51 ms | 0.48 ms |
| $T(60)$ | 3.41 hours | 191.45 ms | 0.89 ms |

It is critical to note that this is an average-case estimation based on the $O(n^{1.42})$ local fit. The theoretical worst-case complexity is $O(n^3)$, which would be significantly slower. However, for the class of random graphs tested, the greedy heuristic proves to be a highly efficient and scalable algorithm, capable of solving a problem with one million nodes in under 5 minutes.

TABLE VII: Extrapolated execution time for Greedy Heuristic (Average-Case)

| Vertices (n) | Estimated Time |
|---|---|
| 1,000 | 14.99 milliseconds |
| 10,000 | 397.13 milliseconds |
| 100,000 | 10.52 seconds |
| 1,000,000 | 4.64 minutes |

## V. Conclusion

This work successfully implemented and comprehensively analyzed two distinct algorithms for solving the Minimum Dominating Set problem. The exhaustive search, while guaranteeing optimality, was confirmed to be exponential, $O(b^n)$. Our key experimental finding was that its practical performance is not governed by $n$ alone, but is overwhelmingly dictated by graph density. Sparse graphs (12.5%) pushed the runtime to its 60-second practical limit at $n = 40$, while dense graphs (75%) solved in milliseconds due to their small domination number $k$. This demonstrates its complete non-viability for problems of non-trivial size.

The greedy heuristic, in contrast, proved to be a highly effective and scalable solution. It exhibited excellent solution quality, achieving a 95.9% average precision and, remarkably, finding the 100% optimal solution for all dense graphs tested.

The most significant analytical insight was the identified discrepancy between the greedy algorithm's formal and experimental complexity. While our formal analysis derived a $O(n^3)$ worst-case bound, the experimental data was fitted to a much faster $O(n^{1.42})$ model. This divergence highlights the critical distinction between theoretical worst-case (which assumes pathological graph structures) and the highly efficient average-case performance observed in practice on random graphs.

For practical applications, the greedy heuristic is the clear and reliable choice, offering an outstanding balance of high-speed polynomial performance and near-optimal solution quality.

## References

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

[2] Teresa W. Haynes, Stephen T. Hedetniemi, and Peter J. Slater, *Fundamentals of Domination in Graphs*, Marcel Dekker, New York, 1998.

[3] David S. Johnson, "Approximation algorithms for combinatorial problems", *Journal of Computer and System Sciences*, vol. 9, no. 3, pp. 256–278, 1974.