Randomized Algorithms for Finding the Minimum Dominating Set of a Graph

Rodrigo Abreu
Mestrado em Engenharia Informática

***Abstract* – The Minimum Dominating Set (MDS) problem asks for the smallest set of vertices that dominates a graph and is NP-hard to solve exactly on large instances. This work studies two randomized algorithms for MDS. The first, is a random search that samples candidate dominating sets. The second, is a randomized greedy construction that mixes greedy choices with controlled randomness.**

Their complexity was analyzed and evaluated on random graphs with different densities, standard benchmark graphs, and instances from an external repository. Experiments show that randomized greedy produces solutions very close to optimal, with running times similar to a deterministic greedy heuristic, while random search is slower and usually less accurate, especially on dense graphs. Scalability estimates suggest that the randomized algorithms can handle graphs with tens of thousands of vertices in at most a few seconds on standard hardware.

***Keywords* – Minimum Dominating Set, Randomized Algorithm, Random Search, Randomized Greedy**

I. Introduction

The Minimum Dominating Set (MDS) problem aims to find, 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 analyzing the implementation of Randomized Algorithms to solve the MDS problem. 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].

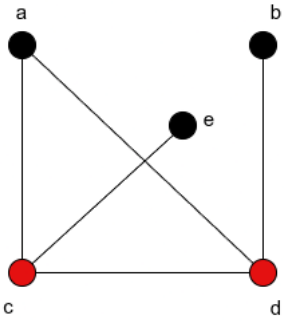

Fig. 1: Example of a 5 vertex graph that has the vertex of a possible minimum dominating set colored in red.

III. Algorithms

A. Randomized Algorithm

A Randomized Algorithm is an algorithm that employs a degree of randomness as part of its logic. Unlike a deterministic algorithm, which will always produce the same output and execute the same sequence of steps for a specific input, a randomized algorithm utilizes a source of random numbers to make decisions during execution.

This means that for the same input, the algorithm's execution path, running time, and sometimes even the output, may vary.

These algorithms are generally simpler and more efficient compared to deterministic alternatives [2]. By introducing random choices into the logic, they often can bypass the complex edge cases that force deterministic algorithms into worst case scenarios, resulting in significantly faster expected running times [3].

Two Randomized Algorithms were implemented in order to solve the MDS problem: Random Search Iterative and Randomized Greedy.

B. Random Search Iterative

In this approach, the algorithm attempts to find a minimum dominating set by exploring candidate solutions in increasing order of size. Starting with a candidate size $K = 1$, the algorithm stochastically generates

subsets of V with cardinality K. Each generated subset is analyzed to verify if it constitutes a dominating set.

To adhere to computational constraints and ensure efficiency, the algorithm implements a memory mechanism (a hash set) to track previously tested configurations, ensuring that no specific subset S is evaluated more than once.

The search for a solution of size K is bounded by a maximum number of attempts (N_{max}) and a global time limit. If a dominating set is found, the algorithm terminates immediately. Since the search iterates from the smallest size upwards, the first solution found is the smallest discovered within the execution limits. If the maximum number of attempts for size K is reached without success, the algorithm assumes no solution exists easily at that size, increments K to $K + 1$, and repeats the sampling process.

Algorithm 1 Randomized Iterative Search for Minimum Dominating Set

Input: Undirected graph $G = (V, E)$, Max iterations per size N_{max}
Output: Minimum dominating set found D^* and its size $|D^*|$

$Visited \leftarrow \emptyset$ **for** $k \leftarrow 1$ **to** $|V|$ **do**
  $attempts \leftarrow 0$ **while** $attempts < N_{max}$ **do**
    $S \leftarrow$ Generate random subset of V with size k
    if $S \notin Visited$ **then**
      $Visited \leftarrow Visited \cup \{S\}$ $attempts \leftarrow attempts + 1$
      $isDominating \leftarrow$ **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**
        return (S, k)
      end
    end
  end
end
return $(V, |V|)$; `// Fallback: The set of all vertices is always dominating`

C. Randomized Greedy Construction

This approach implements a constructive heuristic that builds a dominating set step-by-step rather than searching for complete sets blindly. At each iteration, the algorithm evaluates all vertices not yet in the solution to determine their coverage potential, which was defined as the number of *currently uncovered* vertices that would become dominated if the candidate vertex were added.

To prevent the algorithm from becoming deterministic and getting trapped in local optima, a randomization strategy is integrated into the selection process. Instead of always selecting the vertex with the maximum coverage (pure greedy), the algorithm employs a probabilistic tie-breaking mechanism controlled by a randomness parameter ρ (e.g., 0.3).

In each step, a random value is generated. If this value is below ρ, the algorithm creates a Restricted Candidate List (RCL) containing the top 30% of the best candidates and randomly selects one. Otherwise, it proceeds with the standard greedy choice (selecting the best candidate). This process repeats until the entire graph is dominated.

Algorithm 2 Randomized Greedy Construction with RCL

Input: Undirected graph $G = (V, E)$, Randomness factor $\rho \in [0, 1]$
Output: Dominating set D

$D \leftarrow \emptyset$;$Dominated \leftarrow \emptyset$;**while** $Dominated \neq V$ **do**
  $Candidates \leftarrow \emptyset$;**for** *each vertex* $v \in V \setminus D$ **do**
    $NewNodes \leftarrow (N(v) \cup \{v\}) \setminus Dominated$ $Gain \leftarrow |NewNodes|$ Add $(v, Gain)$ to $Candidates$
  end
  Sort $Candidates$ by $Gain$ descending
  $r \leftarrow$ **Random**$(0, 1)$ **if** $r < \rho$ ***and*** $|Candidates| > 1$ **then**
    $limit \leftarrow \max(1, \lfloor |Candidates| \times 0.3 \rfloor)$ $RCL \leftarrow Candidates[0 \dots limit]$ $v_{best} \leftarrow$ **RandomSelect**(RCL)
  end
  else
    $v_{best} \leftarrow Candidates[0]$
  end
  $D \leftarrow D \cup \{v_{best}\}$ $Dominated \leftarrow Dominated \cup N(v_{best}) \cup \{v_{best}\}$
end
return $(D, |D|)$

IV. Formal Analysis

A. Random Search Iterative

The Random Search Iterative algorithm explores subsets of size k in increasing order until a dominating set is found or all sizes up to $|V|$ are exhausted. For each k, the algorithm performs at most N_{max} random trials. For a generated subset S, dominance verification requires checking every vertex $v \in V$ to determine whether $v \in S$ or it has at least one neighbor in S. For an undirected graph, this verification step has cost

$$T_{\text{check}}(n, k) = \sum_{v \in V} 1 = O(n),$$

since for each vertex we test membership in S and, if needed, scan its adjacency list. Under the assumption of bounded average degree (typical in random graphs), neighborhood checks remain $O(1)$ on average, preserving linear complexity per verification.

Because for each k the algorithm performs at most N_{max} such verifications, the worst–case running time

is

$$T(n) = \sum_{k=1}^{|V|} N_{max} \cdot O(n) = O(n \cdot N_{max} \cdot |V|).$$

Since N_{max} is a fixed parameter and $|V| = n$, the global worst–case complexity is

$$T(n) = O(n^2).$$

However, the *expected* running time is typically much lower, because the algorithm stops as soon as it finds a dominating set, and smaller sets are tried first. In practice, performance tends to scale nearly linearly with n for moderate values of N_{max}.

B. Randomized Greedy Construction

In the Randomized Greedy algorithm, each iteration selects one vertex to add to the dominating set. At iteration t, the algorithm evaluates all currently unused vertices, computing each candidate's marginal coverage gain. Computing the gain of a vertex v requires inspecting its neighborhood, costing $O(\deg(v))$. Summed over all vertices, one full candidate evaluation phase costs

$$\sum_{v \in V} O(\deg(v)) = O\left(\sum_{v \in V} \deg(v)\right) = O(m),$$

where $m = |E|$.

Sorting candidates requires $O(n \log n)$ time, but in sparse graphs ($m = O(n)$), the candidate evaluation step dominates. The algorithm selects exactly one vertex per iteration, and at most n iterations occur, so the overall worst–case running time is

$$T(n, m) = O(nm + n^2 \log n).$$

For sparse graphs ($m = O(n)$), the complexity simplifies to

$$T(n) = O(n^2),$$

while for dense graphs ($m = \Theta(n^2)$), the worst case becomes

$$T(n) = O(n^3).$$

In practice, the marginal gains decrease rapidly as vertices become dominated, reducing the number of effective candidates and yielding performance closer to $O(n^2)$.

V. Experimental Setup

A. Device

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 Operating System used was Ubuntu 24.04.

B. Dataset and Graph Generation

The algorithms were tested on three distinct categories of graph instances to assess robustness across different topologies:

B.1 Random Graphs

A custom generator was implemented to create pseudo-random graphs. For a given number of vertices N and a density probability p, edges are added between any pair of distinct vertices (u, v) if a randomly generated number $r \in [0, 1)$ satisfies $r < p$.

- **Size (N):** Varied from 4 to 40 vertices.
- **Density (p):** Tested at four levels: 0.125 (sparse), 0.25, 0.50, and 0.75 (dense).

The size (N) varied from 4 to 40 vertices and density was tested at four levels: 0.125 (sparse), 0.25, 0.50, and 0.75 (dense).

This resulted in a varied dataset, allowing for the analysis of algorithmic performance relative to graph density.

B.2 Standard Benchmarks

To validate the algorithms against known structures, standard benchmark graphs were utilized, including:

- **Petersen Graph:** A well-known standard in graph theory.
- **Karate Club Graph:** A standard social network benchmark.
- **Cycle (C_{20}) and Complete (K_{10}) Graphs:** To test extreme boundary cases (minimum vs. maximum connectivity).

B.3 External Repository

From the graph repository provided by the professor on the moodle platform, all three undirected graphs were used for tests, but the largest one was removed because the algorithms were unable to solve it in a useful time.

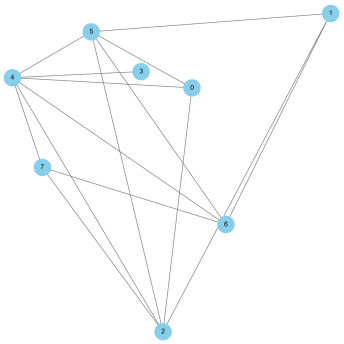

Fig. 2: Example of a graph generated, with 8 vertex and 50% edge density.

VI. Experimental Results

This section presents the empirical results obtained from the computational experiments. The two randomized algorithms (Random Search Iterative and Randomized Greedy) are the main focus. The exhaustive search algorithm and the deterministic greedy

heuristic are used only as baselines to compare running time and solution quality.

A. Execution Time Analysis

Figure 3 shows the execution time (in seconds, logarithmic scale) of all four algorithms on random graphs as a function of the number of vertices n, for the four tested densities $p \in \{0.125, 0.25, 0.50, 0.75\}$.

The Random Search Iterative algorithm is clearly the slowest of the two randomized. With a fixed iteration budget per value of k, its running time grows smoothly with n for all densities. For $n = 40$ the typical time lies between 10^{-4} and a few 10^{-3} seconds. The effect of density is visible: for $p = 0.50$ and $p = 0.75$ the curve is slightly higher, since each dominance test has to inspect larger neighbourhoods. This behaviour is aligned with the $O(n^2)$ bound derived in the formal analysis, but with a relatively large constant factor coming from repeatedly sampling full candidate sets.

The Randomized Greedy algorithm behaves much closer to the deterministic greedy heuristic. In all four panels its curve almost overlaps the greedy one. For all tested sizes and densities, the running time stays in the range 10^{-5} to 10^{-4} seconds, even for $n = 40$. The extra work associated with building the Restricted Candidate List and performing random tie-breaking therefore introduces only a small constant overhead. In practice, Randomized Greedy keeps the same polynomial growth in n as the greedy heuristic, while remaining one or two orders of magnitude faster than Random Search Iterative.

The exhaustive search algorithm displays the expected exponential trend, especially on sparse graphs ($p = 0.125$ and $p = 0.25$), where its running time can reach values close to one second at $n = 40$. In contrast, both randomized methods stay in the millisecond range and avoid the combinatorial blow-up seen in the first assignment.

For denser graphs ($p = 0.50$ and $p = 0.75$) the minimum dominating sets are very small, so the exhaustive search algorithm often finds a solution after exploring only a few small subsets and may run in a similar time to Random Search Iterative. This reflects different constant factors rather than a contradiction of the worst-case analysis.

Overall, the experiments show that the two randomized algorithms have practical, polynomial running times on the tested instances. Randomized Greedy in particular matches the speed of the greedy heuristic from the first assignment, while Random Search Iterative remains competitive but consistently slower.

B. Number of basic operations

Figure 4 reports the number of basic operations performed by each algorithm on random graphs. For the exhaustive and random search methods, one operation corresponds to checking whether a vertex is dominated when testing a candidate set. For the greedy and randomized greedy algorithms, it corresponds to evaluating a vertex as a candidate during one construction step.

The Random Search Iterative algorithm shows a fast increase in operations for small n, followed by a clear plateau. This is expected: the number of different candidate sets it can test is capped by the iteration budget, so after a certain size the cost grows almost linearly with n, but is bounded by roughly $\mathit{iterations} \times n$. For the settings used in the experiments (200 iterations), the total number of dominance checks stays between a few tens and a few hundreds for all densities and for n up to 40. This matches the formal bound $O(\mathit{iterations} \cdot n)$.

The Randomized Greedy algorithm behaves very similarly to the deterministic greedy heuristic. In all four panels, their curves almost overlap and grow smoothly with n. The growth is clearly polynomial: for each construction step the algorithm scans all remaining vertices, so the number of candidate evaluations is $O(n^2)$ in the sparse case and can approach $O(n^3)$ on very dense graphs. In practice, the curves look close to quadratic and remain below a few hundred operations at $n = 40$.

The exhaustive algorithm again shows the strongest growth. On sparse graphs ($p = 0.125$ and $p = 0.25$) the number of configurations tested increases by several orders of magnitude, reaching between 10^5 and 10^7 operations near $n = 40$. This behaviour is consistent with its exponential search space.

By contrast, the greedy and randomized greedy algorithms keep the operation count in the low hundreds, and Random Search Iterative remains bounded by the chosen iteration limit. Compared to the exhaustive method from the first assignment, the randomized approaches avoid the combinatorial explosion and empirically follow the polynomial bounds derived in the formal analysis.

C. Configurations tested

Figure 5 shows how many different configurations (candidate dominating sets) each algorithm tests on the random graphs. The y-axis is in logarithmic scale.

The Random Search Iterative algorithm tests at most a few hundred configurations per instance, almost independently of n. This is a direct effect of the iteration limit: once the algorithm has sampled its budget of random subsets, it stops, even though the number of possible subsets grows exponentially with n. The curves in Figure 5 rise quickly for small n and then flatten, staying close to the configured $N_{\max}$ value. This matches the analysis in Section IV: the search space is bounded by a parameter rather than by the combinatorial size of 2^n.

The Randomized Greedy algorithm is even more frugal. It does not explore whole configurations but builds a single dominating set step by step, adding one vertex per iteration. As a result, the number of configurations it visits is essentially the number of construction steps, which grows linearly with n. Across all densities

Fig. 3: Execution time vs. number of vertices n on random graphs, for different edge densities. The y-axis is in logarithmic scale.

it stays between a handful and a few tens of configurations even at $n = 40$.

The exhaustive method, in contrast, must inspect a large portion of the full search space. For sparse graphs, the number of configurations tested quickly reaches between 10^5 and 10^7, even for graphs with only 30–40 vertices. This again illustrates the exponential nature of the exhaustive approach.

Overall, these results show the main advantage of the randomized strategies: by bounding the number of configurations through an iteration limit (Random Search) or by constructing a single solution (Randomized Greedy), they avoid the combinatorial explosion of the exhaustive algorithm while still producing good solutions in practice.

D. Solution quality and approximation ratios

Figure 6 shows, for each density $p \in \{0.125, 0.25, 0.50, 0.75\}$, how the size of the dominating set produced by each algorithm evolves with the number of vertices n.

On sparse graphs ($p = 0.125$ and $p = 0.25$), the greedy heuristic closely tracks the optimal solution whenever the exhaustive algorithm is available, and remains near–optimal for larger n. Random Search Iterative and Randomized Greedy also find valid dominating sets, but typically with slightly larger cardinality, especially for $p = 0.125$.

As the density increases ($p = 0.50$ and $p = 0.75$), all heuristics converge to very small dominating sets (mostly of size 2 or 3). In these dense graphs, the differences between greedy, Random Search, and Randomized Greedy become negligible, and they almost always match the optimal size where it is known. Overall, the figure shows that the greedy heuristic and its randomized variant provide solutions of comparable quality to the exhaustive algorithm on the tested random graphs, while the pure random search tends to be slightly less accurate, particularly on sparser instances.

E. Comparison between theoretical and experimental complexity

The execution times of the randomized algorithms were fitted to simple models using the measurements on random graphs with density $p = 0.5$ (Figure 7). This allows a direct comparison between theoretical complexity and observed growth.

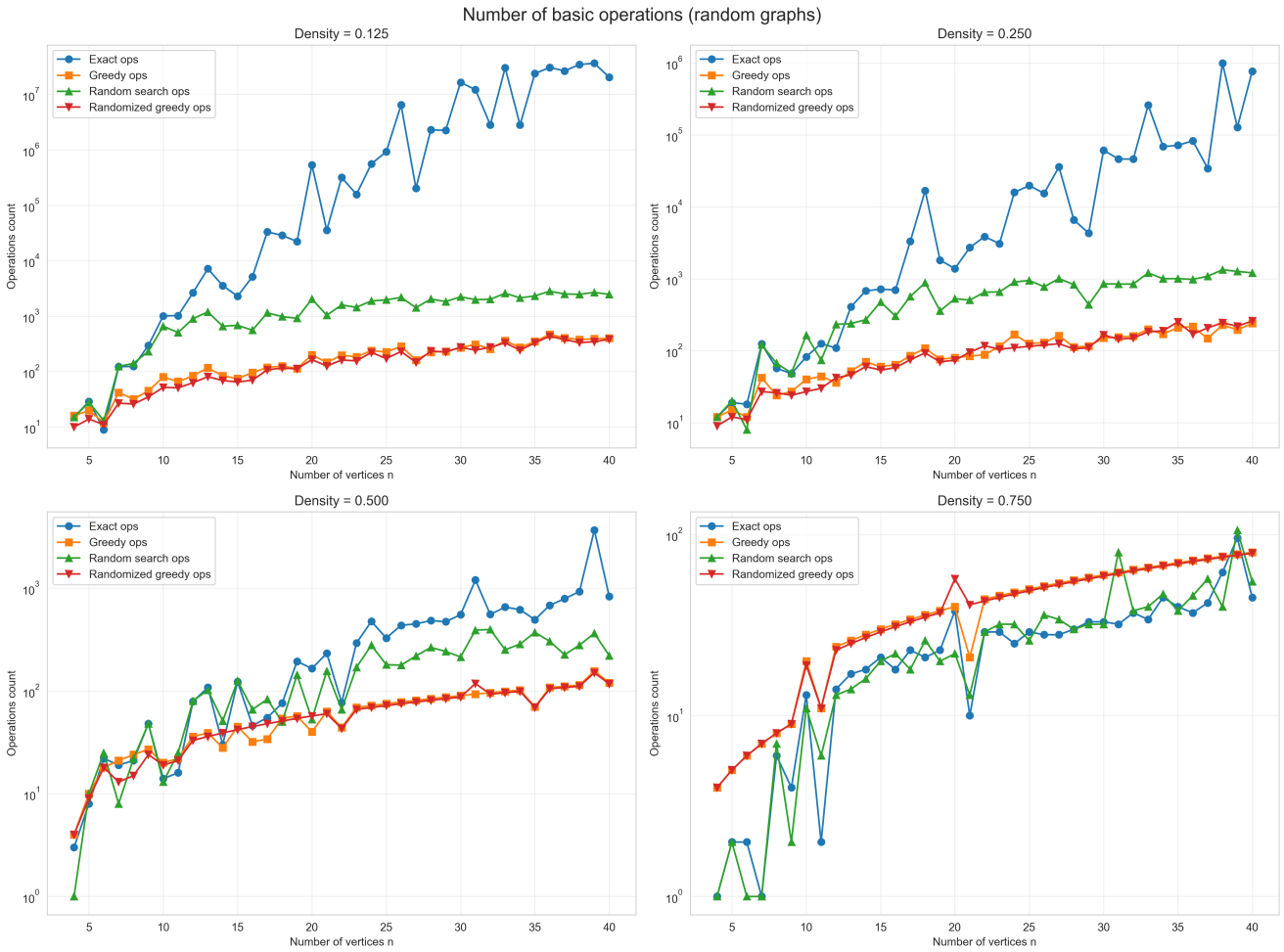

Fig. 4: Number of basic operations vs. number of vertices n on random graphs. The y-axis is in logarithmic scale.

E.1 Random Search Iterative

The fitted model for the Random Search Iterative algorithm is

$$T_{\text{rand}}(n) \approx 1.636 \times 10^{-5} \cdot n^{1.27}.$$

The exponent 1.27 is close to linear and clearly below 2, which is consistent with the $O(n^2)$ upper bound obtained in the formal analysis for a fixed number of iterations.

E.2 Randomized Greedy

For the Randomized Greedy algorithm, the regression yields

$$T_{\text{rg}}(n) \approx 9.155 \times 10^{-7} \cdot n^{1.42}.$$

This exponent lies between 1 and 2, matching the expected $O(n^2)$ behaviour on sparse graphs and remaining far from the $O(n^3)$ worst case for dense instances. The operation counts show the same polynomial trend.

E.3 Summary

In summary, the empirical models confirm the theoretical picture: both randomized algorithms run in polynomial time on the tested graphs, with subquadratic exponents, while still producing high-quality dominating sets.

F. External repository and benchmark graphs

The behaviour on structured graphs is consistent with the results on random instances. Greedy and randomized greedy usually match or come close to the best solution, while random search tends to be slower and less accurate. Detailed results for all benchmark and SW graphs are reported in Tables III and IV in Appendix B.

VII. Scalability and Practical Limits

For the randomized methods, Section VI-A fitted simple polynomial models using the random graphs with density $p = 0.5$. Using those fits, the estimated execution times for larger graphs are:

These values are consistent with the measurements for $n \leq 40$, where the worst observed times were about 15 ms for Random Search and 0.46 ms for Randomized Greedy. Even at $n = 10\,000$, both methods are predicted to run in well under a few seconds. At $n = 100\,000$, they remain under one minute.

Overall, the results indicate that the randomized algorithms scale well beyond the graph sizes used in the ex-

Fig. 5: Number of configurations tested vs. number of vertices n on random graphs. The y-axis is in logarithmic scale.

n	Rand. search $T_{\mathrm{RS}}(n)$	Rand. greedy $T_{\mathrm{RG}}(n)$
100	5.7 ms	0.63 ms
500	44 ms	6.1 ms
1000	0.11 s	16 ms
5000	0.83 s	0.16 s
10000	2.0 s	0.43 s
100000	37.7 s	11.2 s

TABLE I: Estimated execution times for the randomized algorithms as a function of n (density $p = 0.5$).

periments. In realistic settings, storing the graph and auxiliary data structures is likely to become a more serious limitation than CPU time, especially for the Randomized Greedy algorithm.

VIII. Conclusion

This work studied two randomized algorithms for the Minimum Dominating Set problem and compared them with the exhaustive search and the greedy heuristic from the first assignment. On random graphs, greedy and randomized greedy produced solutions very close to optimal, usually off by at most one vertex. Random search always found valid dominating sets, but with larger sizes on average. The exhaustive algorithm was only useful on small instances, where it confirmed optimality but showed the expected exponential growth.

In terms of running time, the randomized algorithms behaved polynomially. Randomized greedy was the best overall compromise: it kept almost the same speed and quality as the deterministic greedy heuristic. Random search was slower and less accurate, and on dense graphs it could even be slower than the exhaustive method, because the latter often finds a tiny dominating set early while random search still spends a fixed number of iterations checking large and costly candidate sets.

The scalability estimates indicate that both randomized algorithms can handle graphs with tens of thousands of vertices in at most a few seconds on standard hardware. For larger instances, memory usage is likely to be more limiting than CPU time, especially for randomized greedy.

Fig. 6: Dominating set sizes obtained on random graphs, for different densities p. The exhaustive algorithm is only executed up to $n = 18$.

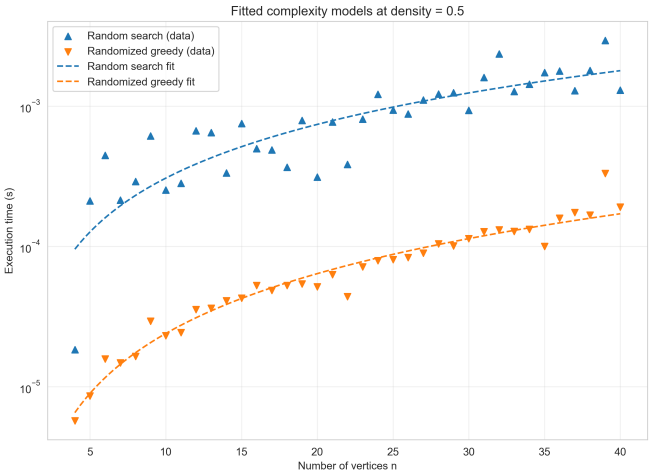

Fig. 7: Fitted complexity models for the Random Search Iterative and Randomized Greedy algorithms on random graphs with density $p = 0.5$ (logarithmic y-axis).

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] Rajeev Motwani and Prabhakar Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.

[3] Michael Mitzenmacher and Eli Upfal, *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*, Cambridge University Press, Cambridge, UK, 2nd edition, 2017.

Appendix

I. Detailed Results on Random Graphs

n	Exact time (s)	Greedy time (s)	Random time (s)	Rand. greedy time (s)
4	0.000017	0.000009	0.000276	0.000008
5	0.000021	0.000010	0.000355	0.000012
6	0.000022	0.000011	0.000229	0.000013
7	0.000065	0.000016	0.000773	0.000017
8	0.000062	0.000017	0.000500	0.000018
9	0.000122	0.000021	0.000664	0.000025
10	0.000314	0.000027	0.001339	0.000031
11	0.000293	0.000024	0.000763	0.000027
12	0.000604	0.000034	0.001346	0.000036
13	0.001759	0.000034	0.001606	0.000041
14	0.001194	0.000034	0.001031	0.000041
15	0.000775	0.000037	0.001245	0.000041
16	0.001851	0.000043	0.001114	0.000050
17	0.010902	0.000048	0.001988	0.000071
18	0.019173	0.000060	0.002126	0.000066
19	0.009999	0.000054	0.001636	0.000062
20	0.134613	0.000050	0.002366	0.000066
21	0.012730	0.000056	0.001665	0.000073
22	0.096072	0.000065	0.002370	0.000083
23	0.057888	0.000077	0.002802	0.000108
24	0.124778	0.000090	0.002849	0.000102
25	0.337211	0.000080	0.003294	0.000115
26	1.370556	0.000087	0.003406	0.000134
27	0.090797	0.000097	0.002728	0.000098
28	0.675433	0.000103	0.003776	0.000164
29	0.629242	0.000099	0.002828	0.000133
30	3.800115	0.000106	0.003068	0.000135
31	3.652994	0.000114	0.003232	0.000133
32	1.051370	0.000123	0.004551	0.000183
33	7.635317	0.000129	0.004724	0.000185
34	1.051595	0.000130	0.004354	0.000190
35	7.539630	0.000135	0.004770	0.000213
36	7.533194	0.000153	0.005199	0.000197
37	7.517601	0.000157	0.005237	0.000244
38	7.868858	0.000158	0.004159	0.000202
39	7.564523	0.000202	0.005213	0.000245
40	7.817544	0.000180	0.005319	0.000240

TABLE II: Average execution time by number of vertices n on random graphs.

II. Detailed results on structured graphs

Graph	n	m	$\lvert D^*\rvert$	$\lvert D_G\rvert$	t_G	$\lvert D_R\rvert$	t_R	$\lvert D_{RG}\rvert$
Petersen	10	15	3	3	$2.1 \cdot 10^{-5}$	3	$7.4 \cdot 10^{-4}$	3
Karate club	34	78	–	4	$1.0 \cdot 10^{-4}$	5	$5.4 \cdot 10^{-3}$	4
Cycle C_{20}	20	20	–	7	$6.4 \cdot 10^{-5}$	8	$5.9 \cdot 10^{-3}$	7
Complete K_{10}	10	45	1	1	$1.2 \cdot 10^{-5}$	1	$1.7 \cdot 10^{-5}$	1

TABLE III: Benchmark graphs (full results).

Graph	n	m	$\lvert D_G\rvert$	t_G	$\lvert D_R\rvert$	t_R	$\lvert D_{RG}\rvert$	t_{RG}
`SWtinyG`	13	13	4	$2.7 \cdot 10^{-5}$	4	$7.5 \cdot 10^{-3}$	4	$4.5 \cdot 10^{-5}$
`SWmediumG`	250	1273	31	$7.4 \cdot 10^{-3}$	62	1.47	34	$7.9 \cdot 10^{-3}$

TABLE IV: SW repository graphs (full results).