Algoritmos Avançados
2nd Project - Finding minimum weight edge dominating set

Miguel Miragaia
Student 108317, DETI
Universidade de Aveiro
Aveiro, Portugal
miguelmiragaia@ua.pt

***Abstract*—The main objective of this article is to analyze the Minimum Weight Edge Dominating Set Problem with randomized algorithms and for different sizes of the problem. Formal analysis of the computational complexity of each algorithm is made and complemented with an experimental analysis based on their execution times and number of basic operations.**

***Index Terms*—Minimum Weight Edge Dominating Set, Randomized Algorithms**

I. Introduction

The Minimum Weight Edge Dominating Set (MWEDS) problem has significant applications in network optimization and resource minimization problems. In a given undirected graph $G = (V, E)$, where V is a set of vertices and E is a set of edges, an edge dominating set (EDS) is defined as a subset of edges such that every edge not in this subset shares at least one vertex with an edge in the subset. The MWEDS problem focuses on identifying the edge dominating set with the minimum total weight, where the weight of an EDS is calculated as the sum of the weights of the edges within this subset [1].

This assignment aims to design, implement and evaluate random algorithms for the MWEDS problem. The focus is not only on the algorithms ability to find an optimal or near-optimal solution, but also on its efficiency, scalability, and reliability.

II. Problem Solution

To address the Minimum Weight Edge Dominating Set (MWEDS) problem, two algorithmic approaches were implemented: a dynamic randomized search algorithm and a dynamic combined search algorithm (improved version of dynamic randomized). These methods offer strengths related to the exhaustive and greedy search algorithms, and are suited to different scales of graph complexity. Here are detailed their methodologies, benefits, and limitations.

A. Dynamic Randomized Search Algorithm

The *dynamic randomized algorithm* is an efficient approximation method for solving the Minimum Weight Edge Dominating Set (MWEDS) problem. Instead of evaluating all possible subsets of edges, like exhaustive search, it employs randomness and dynamic adjustments to explore the solution space effectively.

The algorithm iteratively evaluates subsets of edges, adjusting the subset size based on progress:

- If limited improvement is observed (*base threshold*), the search size increases to explore larger subsets.
- If a promising solution is found (*refine threshold*), the search size decreases to focus on refinement.

At each step, randomly sampled subsets are evaluated to determine if they form a valid Edge Dominating Set (EDS), with the subset of minimum weight being tracked. Random sampling ensures diverse exploration, while dynamic thresholds guide the search toward optimal solutions.

This approach balances exploration and refinement, significantly reducing computational costs compared to exhaustive methods. Although it does not guarantee exact solutions, it delivers approximations efficiently, making it suitable for larger graphs where exact methods are computationally prohibitive.

Algorithms 1 and 3 represent the Dynamic Randomized Search Algorithm.

Algorithm 1 Check Edge Dominating Set

```
1: Procedure IsEdgeDominatingSet($G$, $edge\_set$)
2: for each edge $(u, v)$ in $G$ do
3:     if no edge $(u1, v1, w)$ in $edge\_set$ has $u$ or $v$ as an endpoint then
4:         return False
5:     end if
6: end for
7: return True
8: End Procedure
```

Algorithm 2 Dynamic Combined Heuristic for MWEDS

```
1: Procedure DynamicCombinedMWEDS($G$, $iteration\_factor$, $initial\_search\_size$, $base\_threshold$, $refine\_threshold$, $early\_stopping\_threshold$, $min\_subset\_size$, $max\_subset\_size$)
2: $num\_edges \leftarrow |G.edges|$
3: $max\_iterations \leftarrow iteration\_factor \cdot num\_edges$
4: $edges \leftarrow$ list of edges in $G$ with weights
5: $min\_weight \leftarrow \sum_{(u,v,w)\in edges} w$
6: $search\_size \leftarrow initial\_search\_size$
7: $max\_subset\_size \leftarrow max\_subset\_size$ or $num\_edges$
8: $seen\_subsets \leftarrow$ deque with max length 500
9: $last\_improvement \leftarrow \infty$, $improvement\_count \leftarrow 0$
10: for $iteration \leftarrow 1$ to $max\_iterations$ do
11:   $progress \leftarrow iteration/max\_iterations$
12:   if $progress > base\_threshold$ and $best\_solution = edges$ then
13:     $search\_size \leftarrow \min(search\_size + 1, max\_subset\_size)$
14:   else if $progress > refine\_threshold$ and $best\_solution \neq edges$ then
15:     $search\_size \leftarrow \max(search\_size - 1, min\_subset\_size)$
16:   end if
17:   $upper\_bound \leftarrow \min(search\_size, max\_subset\_size) - 1$
18:   $candidate\_size \leftarrow$ random integer between $min\_subset\_size$ and $upper\_bound$
19:   $candidate\_size \leftarrow \max(candidate\_size, min\_subset\_size)$
20:   $candidate\_set \leftarrow$ random sample of size $candidate\_size$ from $edges$
21:   $candidate\_set\_key \leftarrow$ sorted tuple of $candidate\_set$
22:   if $candidate\_set\_key \in seen\_subsets$ then
23:     continue
24:   end if
25:   Append $candidate\_set\_key$ to $seen\_subsets$
26:   $(is\_dominating, operations) \leftarrow$ IsEdgeDominatingSet($G$, $candidate\_set$)
27:   if $is\_dominating$ then
28:     $weight \leftarrow \sum_{(u,v,w)\in candidate\_set} w$
29:     if $weight < min\_weight$ then
30:       $min\_weight \leftarrow weight$, $best\_solution \leftarrow candidate\_set$
31:       $search\_size \leftarrow \max(min\_subset\_size, search\_size - 1)$
32:       if $last\_improvement - min\_weight < early\_stopping\_threshold$ then
33:         $improvement\_count \leftarrow improvement\_count + 1$
34:         if $improvement\_count > 5$ then
35:           break
36:         end if
37:       else
38:         $improvement\_count \leftarrow 0$
39:       end if
40:       $last\_improvement \leftarrow min\_weight$
41:     end if
42:   end if
43: end for
44: return $best\_solution, min\_weight, basic\_operations, num\_configurations$
```

Algorithm 3 Dynamic Randomized Heuristic for MWEDS

```
1: Procedure DynamicRandomizedMWEDS($G$, $iteration\_factor$, $initial\_search\_size$, $base\_threshold$, $refine\_threshold$)
2: $num\_edges \leftarrow$ number of edges in $G$
3: $max\_iterations \leftarrow iteration\_factor \cdot num\_edges$
4: $edges \leftarrow$ list of edges in $G$ with their weights
5: $best\_solution \leftarrow edges$
6: $min\_weight \leftarrow$ sum of weights of all edges
7: $search\_size \leftarrow initial\_search\_size$
8: $seen\_subsets \leftarrow \emptyset$
9: for $iteration \leftarrow 1$ to $max\_iterations$ do
10:   $progress \leftarrow iteration/num\_edges$
11:   if $progress > base\_threshold$ and $best\_solution = edges$ then
12:     $search\_size \leftarrow \min(search\_size + 1, num\_edges)$
13:   else if $progress > refine\_threshold$ and $best\_solution \neq edges$ then
14:     $search\_size \leftarrow \max(search\_size - 1, 1)$
15:   end if
16:   $candidate\_set \leftarrow$ random subset of $edges$ of size up to $search\_size$
17:   $candidate\_set\_key \leftarrow$ sorted tuple of $candidate\_set$
18:   if $candidate\_set\_key \in seen\_subsets$ then
19:     continue
20:   end if
21:   $seen\_subsets \leftarrow seen\_subsets \cup \{candidate\_set\_key\}$
22:   $is\_dominating, operations \leftarrow$ IsEdgeDominatingSet($G$, $candidate\_set$)
23:   if $is\_dominating$ then
24:     $weight \leftarrow$ sum of weights in $candidate\_set$
25:     if $weight < min\_weight$ then
26:       $min\_weight \leftarrow weight$
27:       $best\_solution \leftarrow candidate\_set$
28:       $search\_size \leftarrow \max(2, search\_size - 1)$
29:     end if
30:   end if
31: end for
32: return $best\_solution, min\_weight, basic\_operations, num\_configurations$
33: End Procedure
```

B. Dynamic Combined Search Algorithm

The *dynamic combined heuristic algorithm* enhances the dynamic randomized search algorithm for the MWEDS problem with the following key improvements:

- **Adaptive Subset Bounds:** Dynamically adjusts subset size between `min_subset_size` and `max_subset_size`.
- **Early Stopping:** Halts if solution improvement falls below `early_stopping_threshold` for consecutive iterations, reducing computation time.
- **Memory of Subsets:** Tracks recently evaluated subsets

using a deque to avoid redundant evaluations.
- **Dynamic Progress Evaluation:** Adjusts search size based on iteration progress, balancing exploration and refinement.

These improvements improve scalability and efficiency, allowing the algorithm to handle larger graphs where exact methods are infeasible. By integrating adaptive mechanisms and progress-based adjustments, it refines the search for minimum-weight edge dominating sets while retaining the computational efficiency of the randomized approach.

Algorithms 1 and 2 represent the Dynamic Combined Randomized Search Algorithm.

C. Comparative Evaluation of Algorithms

To assess the effectiveness of these algorithms, we will compare them with the exhaustive and greedy solutions developed previously. Computational experiments will be conducted on varied graph instances. The exhaustive search solution serves as a benchmark, confirming the optimal edge dominating set and its weight. Since the exhaustive results are limited to graphs with up to 8 vertices, we will also compare the new algorithms against each other for different graph densities. The comparison will focus on metrics such as accuracy, number of basic operations, execution times, solution size, and total weight. All evaluations are made with an iteration factor of 100.

- **Accuracy**: The closeness of the solutions obtained by the new random algorithms to the optimal solution found by exhaustive search. Figure 1 and 2 visualize the accuracy/precision of the new random algorithms compared to the Greedy Algorithm and the optimal solution provided by the Exhaustive Search Algorithm.

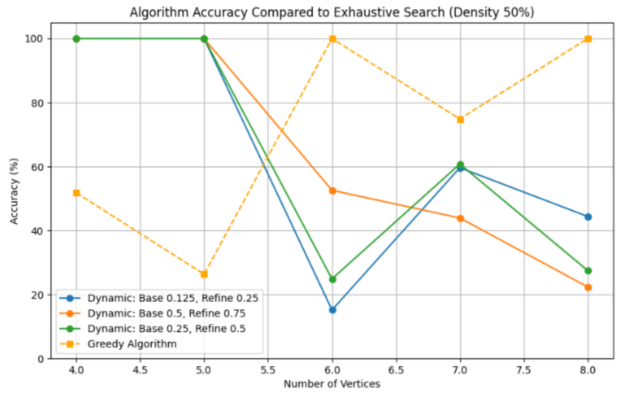

Fig. 1: Comparison of the Dynamic Random Algorithm to the optimal solution in graphs with 50% edge density

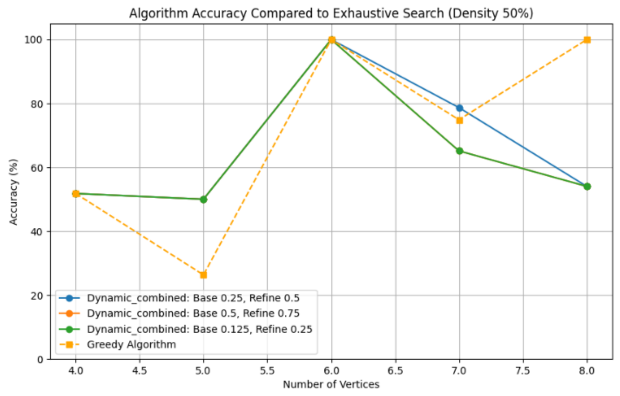

Fig. 2: Comparison of the Dynamic Combined Random Algorithm to the optimal solution in graphs with 50% edge density

- **Solution Weight Comparison**: The comparison of the total weight of the solutions produced by the new randomized algorithms and the optimal solution found by the exhaustive search. Figure 3 and 4 shows the solution weight comparison between the new random algorithms the Greedy Algorithm and the optimal solution for an edge density of 50%.

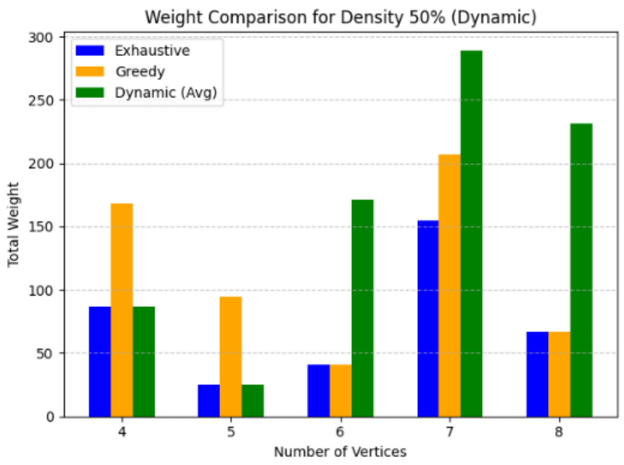

Fig. 3: Solution Weight comparison of the Dynamic Randomized Algorithm and Optimal Solution - (50%) Edge Density

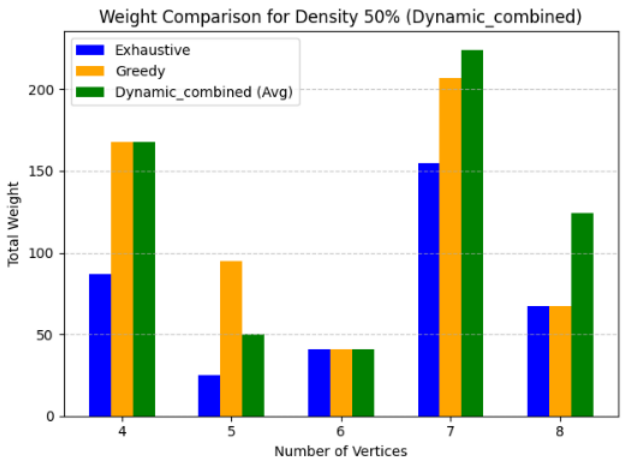

Fig. 4: Solution Weight of the Dynamic Combined Random Algorithm and Optimal Solution - (50%) Edge Density

- **Basic Operations**: The number of basic operations required by each algorithm. The exhaustive search algorithm performs an exponential number of basic operations, as it must evaluate all possible subsets to find the optimal solution, resulting in a rapid increase in operations as graph size grows. In contrast, the new randomized algorithms, as shown in Figure 5 and 6, generally require fewer basic operations compared to the exhaustive search. Additionally, the results for some graphs taken from online sources are presented in Table I.

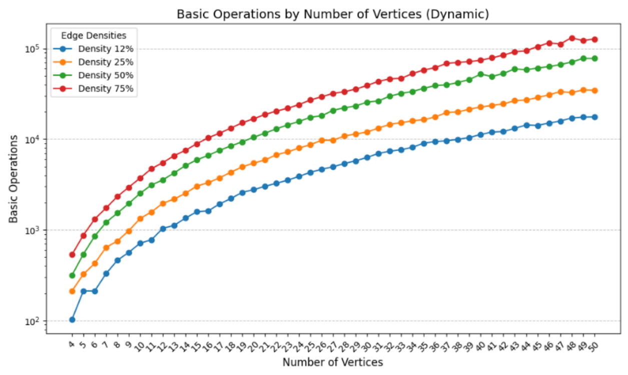

Fig. 5: Comparison of Basic Operations in Dynamic Randomized Algorithm for each density

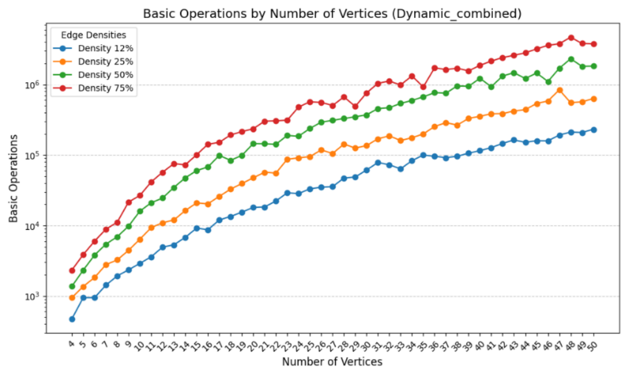

Fig. 6: Comparison of Basic Operations in Dynamic Combined Randomized Algorithm for each density

Num Vertices	Num Edges	Dynamic Rand Basic Ops	Dynamic Combined Basic Ops
8	15	1462	6051
250	2546	192148	7966919

TABLE I: Basic Operations Count for Dynamic Random Algorithm and Dynamic Combined Algorithm of the Internet Graphs

- **Execution Time**: The time required by the new randomized algorithms to complete their computations. This analysis evaluates how execution times scale with graph size and density, highlighting the differences between the dynamic random algorithm and the dynamic combined algorithm. Figure 7 and 8 illustrates these comparisons for various densities. Additionally, results for graphs taken from online sources are summarized in Table II.

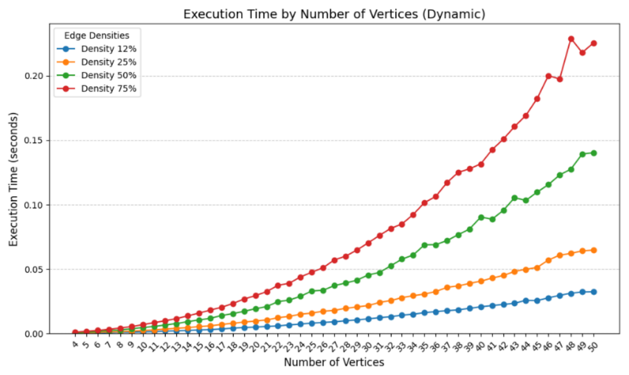

Fig. 7: Execution Time comparison of Dynamic Randomized Algorithm for each density

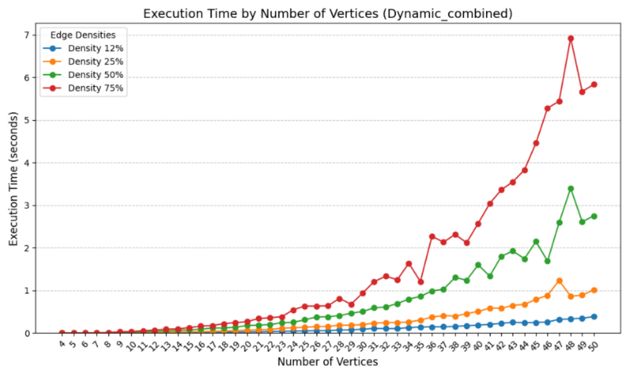

Fig. 8: Execution Time comparison of Dynamic Combined Randomized Algorithm for each density

Num Vertices	Num Edges	Dynamic Randomized Time (s)	Dynamic Combined Time (s)
8	15	0.002741	0.003652
250	2546	0.555256	36.254582

TABLE II: Execution Time Comparison for Randomized Algorithms of the Internet Graphs

- **Solution Size (EDS Size)**: The size of the Edge Dominating Set (EDS) obtained by the randomized algorithms. This metric reflects the effectiveness of the algorithms in finding more compact EDS solutions than larger solutions. Figure 9 and 10 illustrates the EDS size for each algorithm. Additionally, results for graphs taken from online sources are summarized in Table III.

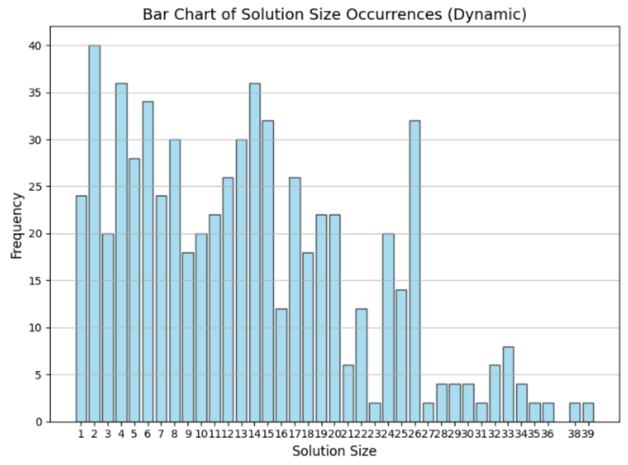

Fig. 9: Solution Size Comparison (EDS Size) of Dynamic Randomized Algorithm for each density - Graph with 30 Vertices

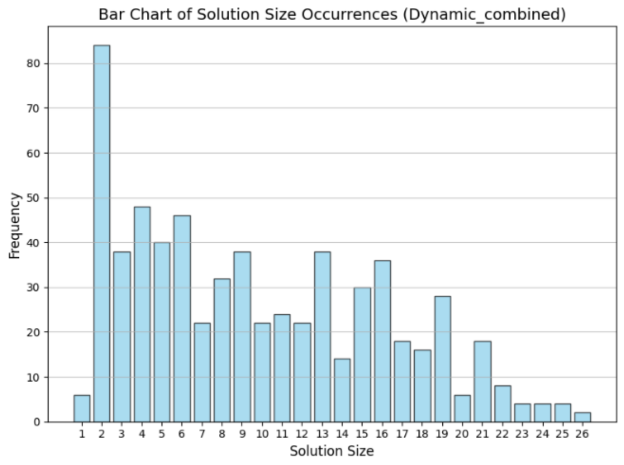

Fig. 10: Solution Size Comparison (EDS Size) of Dynamic Combined Random Algorithm for each density - Graph with 30 Vertices

Num Vertices	Num Edges	Dynamic Rand EDS Size	Dynamic Combined EDS Size
8	15	5	3
250	2546	431	301

TABLE III: Solution Size (EDS Size) Comparison for Randomized Algorithms of the Internet Graphs

- **Weight Comparison**: The total weight of the Edge Dominating Set (EDS) generated by the randomized algorithms is compared to evaluate which algorithm provides a more efficient solution in terms of edge weights. Figure 11, compares the EDS weight produced by the dynamic random and dynamic combined algorithms.

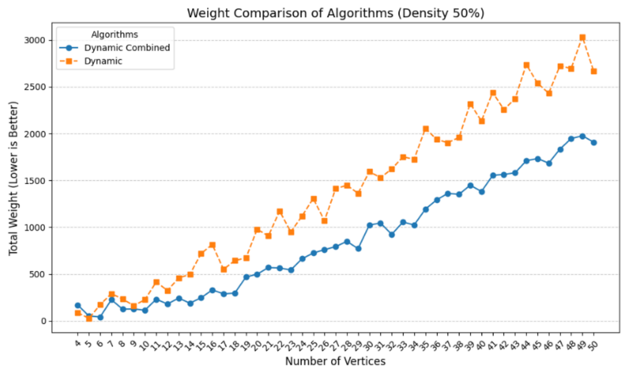

Fig. 11: Weight Comparison of the EDS between Randomized Algorithms at 50% Density

III. Algorithm Development

The Dynamic Combined Algorithm is an improved version of the Dynamic Randomized Algorithm, and for this reason we will only analyze the improved algorithm.

A. *Dynamic Combined MWEDS Algorithm Development*

This algorithm was developed to address the limitations of exhaustive approaches by integrating dynamic search parameters and efficient subset evaluation. The development process emphasizes modularity, configurability, and scalability. Key stages in the algorithm's development are as follows:

- **Dynamic Iterations Setup**: The number of iterations is determined dynamically based on the graph size (`iteration_factor`). The calculation of `max_iterations` ensures scalability by adapting to larger graphs:

```
num_edges = len(G.edges)
max_iterations = iteration_factor
    * num_edges
```

- **Subset Selection Logic**: A randomized selection process is implemented to generate candidate edge subsets of varying sizes. The size of each subset is constrained dynamically by the parameters `min_subset_size` and `max_subset_size`, which adjust based on thresholds:

```
upper_bound = min(search_size,
    max_subset_size) - 1
candidate_size = random.randint(
    min_subset_size, upper_bound)
candidate_set = random.sample(
    edges, candidate_size)
```

- **Threshold-Based Search Adjustment**: The search size is adjusted dynamically using thresholds to balance exploration and refinement:
  - If progress is limited (`base_threshold`), the search size increases to explore larger subsets.
  - If promising solutions are found (`refine_threshold`), the search size decreases to focus on refinement.

```
if progress is > base_threshold
    and best_solution == edges:
     search_size = min(search_size
         + 1, max_subset_size)
elif progress is >
    refine_threshold and
    best_solution != edges:
     search_size = max(search_size
         - 1, min_subset_size)
```

- **Subset Validation and Optimization**: Each candidate subset is validated using the `is_edge_dominating_set` function. Subsets are skipped if they have been previously evaluated, using a deque to track recent configurations:

```
candidate_set_key = tuple(sorted(
    candidate_set))
if candidate_set_key in
    seen_subsets:
     continue
seen_subsets.append(
    candidate_set_key)
is_dominating, operations =
    is_edge_dominating_set(G,
    candidate_set)
```

- **Early Stopping Mechanism**: To prevent unnecessary computations, the algorithm halts if improvement in the solution weight remains below the `early_stopping_threshold` for several iterations:

```
if last_improvement - min_weight <
     early_stopping_threshold:
    improvement_count += 1
    if improvement_count > 5:
        break
else:
    improvement_count = 0
```

- **Result Compilation**: After all iterations or an early stop, the algorithm outputs the best solution, its weight, the total basic operations, and the number of configurations evaluated:

```
return best_solution, min_weight,
    basic_operations,
    num_configurations
```

This modular development process ensures that the algorithm is flexible, allowing adjustments to parameters for different graph structures and sizes. Additionally, tracking metrics such as basic operations and configurations aids in analyzing the algorithm's performance.

IV. Formal Analysis of the Algorithm

A. Dynamic Combined MWEDS Algorithm

This approach combines the benefits of random sampling with adaptive search size and early stopping mechanisms, making it more computationally feasible than exhaustive search algorithms, while still being more accurate than basic greedy approaches for larger graphs. In contrast to the exponential complexity of exhaustive search approach, the dynamic combined MWEDS algorithm offers a balanced trade-off between exploration and refinement of the solution space.

B. Time Complexity

The time complexity of the dynamic combined MWEDS algorithm depends on several factors:

- **Iteration Factor**: The number of iterations is dynamically determined by the size of the graph, specifically the number of edges $|E|$. The maximum number of iterations is proportional to $|E|$, scaled by the `iteration_factor`. Thus, the time complexity can be influenced by the number of edges:

$$O(|E| \cdot \texttt{iteration\_factor})$$

- **Search Size Adjustment**: The algorithm adjusts the search size dynamically during execution based on the progress. While this adds complexity, the adaptive search helps in reducing unnecessary computations compared to exhaustive search, as the search space is explored in smaller steps when necessary.
- **Subset Sampling**: Each iteration involves random sampling of subsets of edges. For each subset sampled,

the algorithm checks whether it forms a valid dominating set and computes its weight. Given that the sampling process involves choosing subsets of size between `min_subset_size` and `max_subset_size`, the number of potential subsets to sample is bounded by $O(2^s)$, where s is the current search size.

- **Worst-Case Time Complexity**: The worst-case time complexity is proportional to the number of iterations, the graph size, and the sampling process. Therefore, the overall worst-case time complexity can be expressed as:

$$O(|E| \cdot V)$$

where V represents the number of vertices in the graph, and $|E|$ is the number of edges.

C. Space Complexity

The space complexity of the dynamic combined MWEDS algorithm is influenced by the storage requirements of the algorithm during execution:

- **Edge Subsets Storage**: The algorithm maintains a `deque` of up to 500 subsets of edges. Each subset is stored as a key, which is a sorted tuple of edges, leading to space usage of $O(2^s)$, where s is the maximum search size.
- **Other Memory Usage**: The algorithm also requires storage for the graph, the current best solution, and temporary variables such as progress, last improvement, and basic operation counts. These contribute a constant space complexity.

Thus, the overall space complexity is dominated by the storage of previously evaluated subsets:

$$O(2^s)$$

where s is the dynamically adjusted search size, and 2^s represents the number of unique subsets the algorithm can track at once.

D. Summary

- **Time Complexity**: $O(|E| \cdot V)$, where $|E|$ is the number of edges and V is the number of vertices.
- **Space Complexity**: $O(2^s)$, dominated by the storage of previously evaluated edge subsets.

This dynamic approach significantly reduces time and space complexity compared to exhaustive search, while still maintaining a quality solution for large graphs.

E. Time Complexity Prediction

The time complexity of the algorithm grows with the number of edges ($|E|$) and the number of vertices (V). The execution time is modeled based on the formula:

$$T_{\text{algorithm}}(|E|, V) = C_{\text{algorithm}} \cdot \texttt{iteration\_factor} \cdot |E| \cdot V$$

where:

- $C_{\text{algorithm}}$ is an empirical coefficient derived from experimental data.
- `iteration_factor` is a scaling parameter determining the number of iterations.

1) Example Calculation: Using the data point for $V = 10$ and $D = 50\%$, the number of edges can be calculated as:

$$|E| = \frac{50}{100} \cdot \frac{10 \cdot (10-1)}{2} = \frac{1}{2} \cdot 45 = 22.5 \quad \text{(rounded to 22 edges)}$$

For $T_{\text{measured}} = 0.017191171646118164$ seconds, the coefficient is estimated as:

$$\begin{aligned} C_{\text{algorithm}} &= \frac{T_{\text{measured}}}{\texttt{iteration\_factor} \cdot |E| \cdot V} \\ &= \frac{0.017191171646118164}{100 \cdot 22 \cdot 10} \\ &\approx 7.814 \times 10^{-8} \end{aligned}$$

This coefficient models the growth and can be used to predict execution times for larger graphs.

2) General Prediction Model: For a general number of edges ($|E|$) and vertices (V), the predicted execution time is:

$$T^{\text{predicted}}_{\text{algorithm}}(|E|, V) = C_{\text{algorithm}} \cdot \texttt{iteration\_factor} \cdot |E| \cdot V$$

This model allows the execution time to be approximated based on the graph's edge and vertex count. As it can be seen by the comparison between the Measured Time and the Predicted Time, in Figure 12.

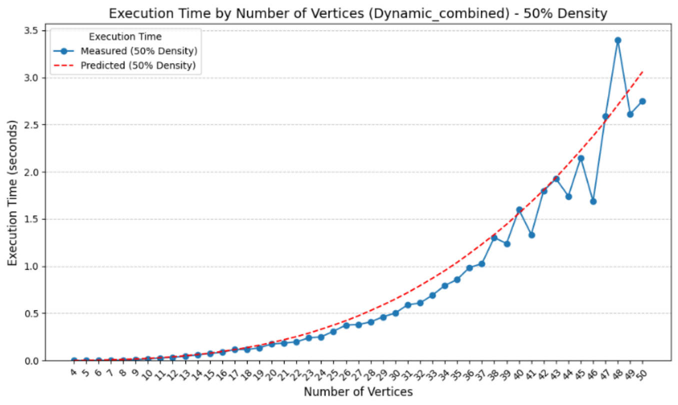

Fig. 12: Predicted Time Complexity - 50% Edge Density

V. Algorithm Parameters Analysis

It's important to analyze the key parameters of the algorithm, including the **iteration factor**, **base threshold**, and **refine threshold**. These parameters significantly influence the algorithm's performance, execution time, and the quality of the solutions obtained. By systematically varying these parameters and observing the results, we gain insights into their roles and importance.

A. Iteration Factor

The **iteration factor** directly determines the maximum number of iterations for the algorithm. A higher iteration factor increases the search space, allowing for more thorough exploration of candidate solutions, but it also increases execution time. Conversely, a smaller iteration factor reduces computation time but might compromise the quality of the solution.

Table IV presents the impact of the iteration factor on execution time and the weight of the EDS for various graph sizes.

Iteration Factor	Execution Time (s)	Weight of EDS	Graph Size (V, E)
50	0.424890	1066	(30, 217)
100	0.6063	963	(30, 217)
1000	7.3285	851	(30, 217)

TABLE IV: Impact of Iteration Factor on Execution Time and Solution Quality

As shown in the table, increasing the iteration factor improves the precision of the solution at the cost of longer execution times. For larger graphs, the iteration factor must be reduced to maintain computational feasibility.

B. Base Threshold and Refine Threshold

The **base threshold** and **refine threshold** dynamically adjust the search size during execution, balancing exploration and refinement. A high base threshold increases the exploration of larger candidate subsets, while a low refine threshold focuses on refining promising solutions.

Table V compares the results for different pairs of base and refine thresholds. Iteration factor is 100, and edge density is 50%.

Base Threshold	Refine Threshold	Execution Time (s)	Weight of EDS	Graph Size (V, E)
0.125	0.25	0.3658	1028	(30, 217)
0.25	0.50	0.4453	905	(30, 217)
0.50	0.75	0.3917	928	(30, 217)

TABLE V: Impact of Base and Refine Thresholds on Execution

From the results, it is evident that a carefully chosen pair of thresholds can improve execution efficiency and solution quality.

C. Insights

1. The **iteration factor** should be adjusted dynamically based on the graph size and density to balance execution time and solution quality. Larger graphs benefit from a smaller iteration factor, while smaller graphs can utilize a higher value. For this problem, an iteration factor of 100 has been observed to provide a good balance between accuracy and execution time, offering reliable performance across a range of graph sizes.

2. The **base threshold** and **refine threshold** should be tuned based on the desired trade-off between exploration and refinement. High thresholds are recommended for the initial stages of optimization, transitioning to lower thresholds for fine-tuning. For this problem, thresholds of 0.50 (base) and 0.75 (refine) yielded the most accurate results, effectively balancing exploration and refinement to minimize the weight of the EDS.

3. To ensure practical applicability, it is crucial to determine the largest graph that can be processed on the available computational resources without excessive runtime. Using a suitable parameter configuration (iteration factor of 10, base threshold of 0.50, and refine threshold of 0.75). A graph with **1000 vertices** and **16866 edges** can be processed within a time frame of 364.7 seconds. This graph was the biggest graph I had at my disposal to test, that was feasible.

Table VI represents some time predictions for much larger graphs.

Num Vertices (V)	Num Edges ($\|E\|$)	Execution Time (s)
1,000	16866	370.695
10,000	123462	12346.2

TABLE VI: Expected Execution Times for Specific Graphs

VI. Conclusion

This report explored the performance of a randomized algorithm for solving the Minimum Weight Edge Dominating Set (EDS) problem on undirected graphs. The randomized approach enables efficient exploration of the solution space by dynamically generating candidate subsets, balancing between exploration and exploitation through parameter tuning.

Key algorithmic parameters, such as the **iteration factor**, **base threshold**, and **refine threshold**, significantly influence both execution time and solution quality. The **iteration factor** directly impacts the computational effort. The thresholds, on the other hand, govern the adaptability of the search, adjusting subset sizes to refine or expand the exploration based on intermediate results.

The randomized nature of the algorithm provides flexibility and robustness across diverse graph instances.

In summary, this study highlights the power of randomized algorithms combined with dynamic parameter tuning, solving graph-based optimization problems while maintaining adaptability to diverse input characteristics.

VII. Code Organization

To go along with this report which results were developed in a GitHub repository, there will be a set of files organized in a set of folders, namely:

- **graphics:** contains the folders with the different visualizations
- **graphs:** contains the original graphs created ad the internet graphs
- **graphs_solution:** contains the graphs with the solution marked in red
- **results:** contains multiple csv's with the metrics analyzed in this problem
- **main.py:** main function file
- **graph_creation.py:** graph creation file
- **algorithms.py:** file that contains the Rndomized Algorithms
- **analysis.py:** file that performs the different needed visualizations.

References

[1] S. Bouamama and C. Blum, "The Minimum Weight Dominating Set Problem," Mayra Albuquerque, 2018. Retrieved from: https://arxiv.org/pdf/1808.09809