AA - Maximum Weight Independent Set Exhaustive and Greedy Algorithms

Alexandre Ribeiro - 108122

***Abstract* – In recent years, breakthroughs in algorithm development have led to faster methods for tackling NP-hard problems, with some approaches achieving quasi-polynomial runtimes. This work focuses on the Maximum Weighted Independent Set (MWIS) problem, developing both exhaustive and greedy algorithms to deliver solutions that are not only precise but also scalable and efficient.**

My experiments show that while greedy algorithms significantly reduce computation time, the accuracy of the results tends to decline for higher edge densities. To mitigate this, I explore combining heuristics, which enhances accuracy without compromising the solving speed, providing a balanced approach to obtaining an approximate solution.

***Keywords* – Maximum weighted independent set, exhaustive, greedy, heuristic**

I. Introduction

The Maximum Independent Set problem is one of the most important problems in graph algorithms and has been widely studied since its formulation in the early 20th century. Its weighted variation (MWIS), where each node is attributed with a non-negative weight, has also received a lot of attention due to the potential applications in areas such as social networks [1] and computer graphics [2].

Recent research has introduced innovative algorithms that solve MWIS accurately with remarkable speeds. Notably, a study published in June [3] consolidated prior conjectures and theorems to achieve a quasi-polynomial-time solution. However, the search for a polynomial solution to NP-hard problems still continues to this day.

The MWIS problem starts with an undirected graph $G(V, E)$, where $|V|$ is the number of nodes (vertices) and $|E|$ the number of edges. Each vertex has an attributed positive weight $w : V(G) \to \mathbb{Z}_{\geq 0}$. The goal is to find a subset of vertices $S \subseteq V$ with maximum total weight, such that for every V_1, V_2 in S, there is no edge connecting the vertices.

In this work, I explored two types of algorithms in order to solve this problem: exhaustive and greedy.

A. Exhaustive Algorithm

Exhaustive search algorithms are known to be exact but highly inefficient, meaning they always reach the optimal solution but their execution time grows exponentially for larger problems, due to its nature of testing every possibility.

In the MWIS problem, the exhaustive search method consists in generating every subset of nodes of a given graph and testing them, checking if they constitute an independent set and then determining the total weight. The algorithm doesn't finish until it has tested every subset and determined the best solution.

B. Greedy Algorithm

A greedy algorithm offers a faster solution to computational problems, though often at the cost of precision. However, it performs well on complex instances, and its accuracy can be enhanced by refining the algorithm and its heuristics.

For the MWIS problem, there are various possible heuristics that one can use to develop a greedy algorithm. For this work, I chose the following: WMax (Maximum Weighted Node First), where the nodes are sorted by weight and sequentially picked; DMin (Least Degree Node First), similar to WMax but the first nodes to be picked are the ones with the least connected edges, and lastly a mix of both, WDMix, where a ratio of weight and degree is calculated and used to find the best solution.

II. Formal Analysis

A. Exhaustive Algorithm

Algorithm 1 Exhaustive V1

Require: Graph $G = (V, E)$ with node weights $w : V \to \mathbb{R}$
Ensure: Maximum weight independent set S^*

```
S* ← ∅
max_weight ← 0
for all S ⊆ V do
    if S is an independent set in G then
        weight ← Σ_{v∈S} w(v)
        if weight > max_weight then
            max_weight ← weight
            S* ← S
        end if
    end if
end for
    return S*
```

This algorithm iterates through all possible subsets S of the nodes V in G. For each subset:

- It checks if S is an independent set;
- If S is independent, it calculates the total weight of S;
- If the total weight is bigger than the weight of the currently stored S^*, then S is stored and becomes the new S^*.

The algorithm then returns the maximum weight independent set found.

We can analyze the time complexity in 3 parts:

- There are $2^{|V|}$ possible subsets for V, therefore the complexity of iterating all subsets is $O(2^{|V|})$.
- For each subset, we need to determine whether it forms an independent set. In the worst case, every pair of nodes is checked to determine if an edge exists between them. Letting $s = |S|$, the complexity of this operation will be $O(s^2)$, and since $s \leq |V|^2$ this step is bounded by $O(|V|^2)$.
- When a suitable subset is found, the total weight of the subset is calculated, which takes $O(|S|)$, or $O(|V|)$ at most.

Combining all 3 parts, we get an overall time complexity of

$$O(2^{|V|} \cdot (|V|^2 + |V|)) \approx O(2^{|V|})$$

This is exponential due the $2^{|V|}$ factor, meaning the algorithm is feasible only for very small graphs. Because of that, I developed a better algorithm that uses backtracking and pruning to reduce the amount of subsets checked.

Algorithm 2 Exhaustive V2

Require: Graph $G = (V, E)$ with node weights $w : V \to \mathbb{R}$
Ensure: Maximum weight independent set S^*

```
S* ← ∅
max_weight ← 0
function BACKTRACK(current_set, remaining_nodes, current_weight, max_set_info)
    if current_weight > max_weight then
        max_set_info[0] ← current_set
        max_set_info[1] ← current_weight
    end if
    for each node n ∈ remaining_nodes do
        if no neighbor of n is in current_set then
            Add n to current_set
            next_weight ← current_weight + w(n)
            BACKTRACK(current_set, remaining_nodes[i+1:], next_weight, max_set_info)
            Remove n from current_set
        end if
    end for
end function
BACKTRACK(set(), list(G.nodes), 0, [set(), 0])
    return max_set_info[0]
```

This improved algorithm avoids checking all subsets by traversing the search space recursively and pruning paths where nodes are not independent.

The algorithm begins by calling the *Backtrack* function with an empty subset and a list of all nodes in the graph as the remaining nodes to consider. In each recursive call, it first checks if the weight of the current subset is greater than the maximum weight found so far. If it is, the current subset and weight information are updated to record this as the best subset. Next, a *for* loop iterates over the remaining nodes:

- For each node, the algorithm checks if it can be added to the current subset while keeping it independent (i.e., no neighbors of the node are already in the subset).
- If the node is compatible with the subset, it gets added, and a recursive call is made to explore this new subset further with updated remaining nodes.
- If the node cannot be added because it violates the independence condition, the algorithm skips this branch, effectively pruning it from further exploration.

After exploring all options for a node in the subset, the node is removed (backtracked), allowing the algorithm to explore alternative branches. The recursion terminates when all possible nodes and subsets have been considered, ensuring that all independent sets are explored. The maximum weight independent set found is returned.

This algorithm differs from the previous one by pruning branches of subsets that cannot yield any solution. Because of that, depending on the density of the graph, the time it takes to reach a solution can be a lot faster.

We can formulate this by pondering about the worst and best cases. The worst case would be a graph where the density of edges is null, meaning there are no edges, The algorithm would have to run each possible subset, which would result in a complexity of:

$$O(2^{|V|)}$$

The best case would be if the density was maxed, meaning all possible edges exist and the solution's size is a singular node. The algorithm would only have to backtrack once, meaning the complexity would be:

$$\Omega(|V|^2)$$

Depending on the edge probability (density) k of the graph, the time complexity will be better for higher values of k and worse for lower values.

Overall, we can conclude that the *Exhaustive V2* version of the algorithm is an improvement on the previous algorithm.

B. Greedy Algorithm

Algorithm 3 Greedy Example

Require: Graph $G = (V, E)$ with node weights $w : V \to \mathbb{R}$
Ensure: Maximum weight independent set S^*
1: $S^* \leftarrow \emptyset$
2: Sort nodes in V by heuristic in descending order, resulting in sorted_nodes
3: excluded_nodes $\leftarrow \emptyset$
4: **for** each node v in sorted_nodes **do**
5: **if** $v \in$ excluded_nodes **then**
6: **continue** ▷ Skip nodes already adjacent to selected nodes
7: **end if**
8: Add v to S^*
9: Add v and all neighbors of v to excluded_nodes
10: **end for**
return S^*

This algorithm can be applied to any of the heuristics chosen for this paper. The way it works is by first sorting all nodes of a given graph $G = (V, E)$ by an heuristic, which can be either the biggest weight, smallest node degree or a ratio of both. It then iterates the resulting list, choosing the best node first, adding it to the solution set, removing its neighbors from the search space and then moving on to the next possible node.

Compared to the exhaustive algorithm, this results in a much lower time complexity due to the number of operations no longer being exponential. By analyzing the algorithm step by step, we can determine that:

- Sorting the nodes has a time complexity of $O(n \log n)$, where $n = |V|$ is the number of nodes in the graph.
- In each loop, the exclusion check for each node is $O(1)$ given that the set implementation is efficient, and adding each neighbor to the excluded set is at worst $O(n)$. Therefore, the complexity of this step is bounded by $O(n + m)$, where $m = |E|$ is the number of edges.

Combining both steps, the overall time complexity of this algorithm is:

$$O(|V| \log |V| + |V| + |E|) \approx O(|V| \log |V|)$$

III. Experiments

The algorithms and the respective tests were developed in the Python language, using multiple libraries to help generate graphs and plot comparisons between results. The main packages that were used are NetworkX (graph generation), Matplotlib (plot to generate figures with information) and Time (determine how long algorithms take to solve graphs). Other packages were used for smaller tasks, such as Pickle to store graphs in files and Itertools to facilitate subset generation.

Each graph was generated using NetworkX, with each node assigned a random weight between 1 and 100. To ensure the graphs were equal for the comparison of the algorithms, my student number was used as the seed for both the graphs and the weights. For larger graphs, where the creation time interferes with the algorithm's solving time, graphs were generated in advance and stored in files.

A. Initial Remarks

Using the Exhaustive Algorithm, which provides the correct solution every time, we can make some observations about the MWIS problem.

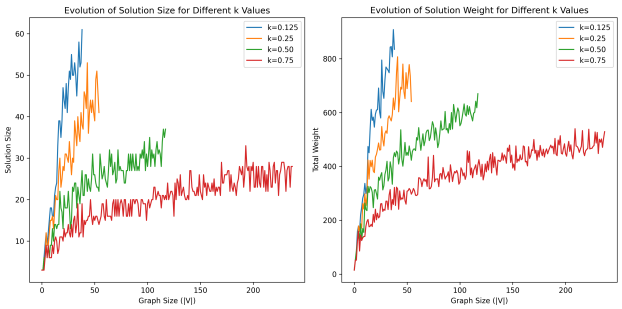

Fig. 1: Solution Size And Total Weight Analysis For Different Values Of k

This graph shows the solution size and total weight for different graph sizes and k values. Looking at the evolution of the data, we can determine that both the size and the total weight of the solution heavily depend on the density of the graph, where a lower k leads to larger and heavier solutions, and a higher k does the opposite.

B. Exhaustive Algorithm

Two versions of the exhaustive algorithm were developed and tested (V1 and V2). The results show that V2 is a great improvement on the V1 version, solving graphs in faster times and allowing larger graphs to be processed. However, its complexity still increases exponentially.

The following graphs use a logarithmic scale to ensure all data points are clearly visible.

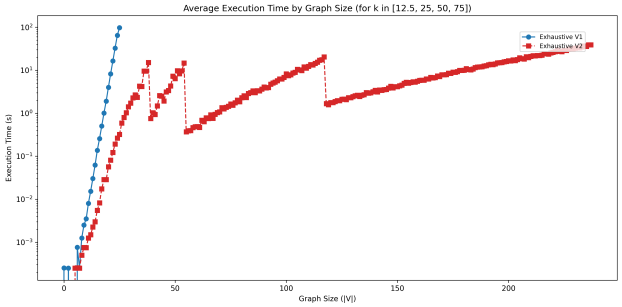

Fig. 2: Execution Time Comparison For Exhaustive V1 And V2

This graph shows the average execution time by graph size for k values in $[0.125, 0.25, 0.50, 0.75]$. We can observe two patterns in this data:

- The V2 algorithm can handle larger graphs and operates faster than V1, but it still exhibits a somewhat exponential growth in runtime as graph size increases.
- While V1's time complexity follows a clear pattern of $(2^{|V|})$, V2 exhibits three distinct points where the solving time drops. This variation occurs because V2 was able to solve larger graphs as k grew, meaning more data was available for the comparison. Those points represent the thresholds for k=0.125, k=0.25 and k=0.50.

Next, we look at the number of operations performed:

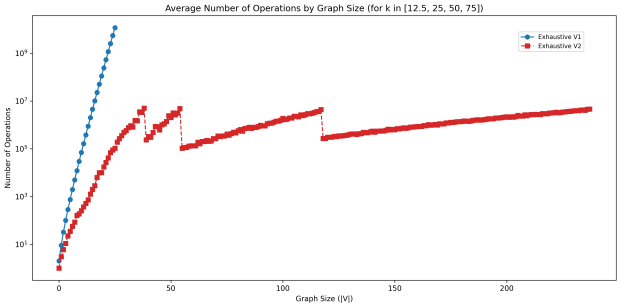

Fig. 3: Number Of Operations Comparison For Exhaustive V1 And V2

This graph displays the average number of operations per graph size for the same set of k values. The observed patterns align with previous results, confirming that V2 outperforms V1 primarily due to its reduced number of operations.

Overall, this is consistent with the formal analysis of both versions of the algorithm.

C. Greedy Algorithm

Three heuristics were used to test the Greedy Algorithm: WMax, which prioritizes node weight; DMin, which prioritizes the least node degree; and WDMix, a combination of both metrics using their ratio.

These tests where harder to design and perform, due to the fact that beyond a certain number of nodes the graph generation becomes slower and detrimental to the test. Because of that, I decided to generate the graphs separately and store them into files, which overall improved the efficiency but still maintained the graph size limitation. In the end, the range of graphs that I used for testing where 1 to 800 for comparing with the exhaustive method, and 100 to 3500 in increments of 100 to test how long it takes for larger problems.

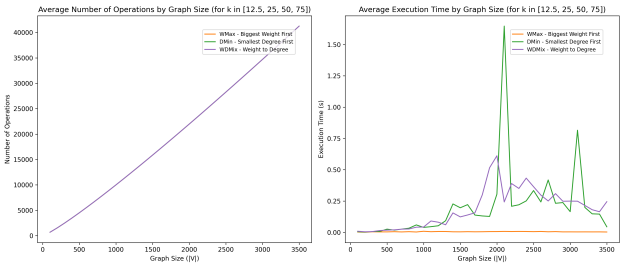

Fig. 4: Average Number Of Operations and Execution Time Comparison Between Heuristics

The first graph shows that the number of operations is nearly identical across heuristics, as their underlying algorithms are similar. Minor fluctuations in this metric are present but too small to be visible at the graph's scale.

The second graph reveals an anomaly: while WMax achieves near-zero execution times, DMin and WDMix exhibit occasional spikes. This discrepancy likely arises from how node degrees are accessed. Both DMin and WDMix rely on the NetworkX function $G.degree(n)$ for each node n. This function may introduce additional steps, explaining the observed spikes. Although this function is suggested to be a cached property, which should minimize overhead, its exact complexity remains undocumented, leaving its impact on performance unclear.

Returning to the first graph, the observed growth in operations aligns with the expected $O(n \log n)$ complexity. For graphs larger than 3500 nodes, a similar trend would likely continue.

D. Exhaustive VS Greedy

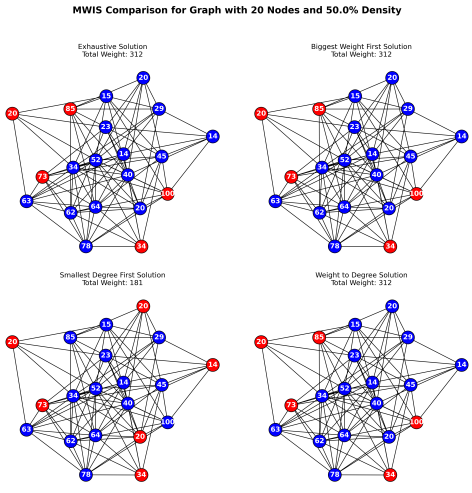

Fig. 5: Solution Comparison For All Algorithms

The image above shows the best solution found by each algorithm for a graph with 20 nodes and a density of 50%. As we can see, the definitive solution is an independent set of 5 nodes, with a total weight of 312. Both WMax and WDMix succeeded, however the DMin heuristic reached a sub-optimal solution of

only 181 total weight. We can analyze these tendencies more clearly in the following graphs.

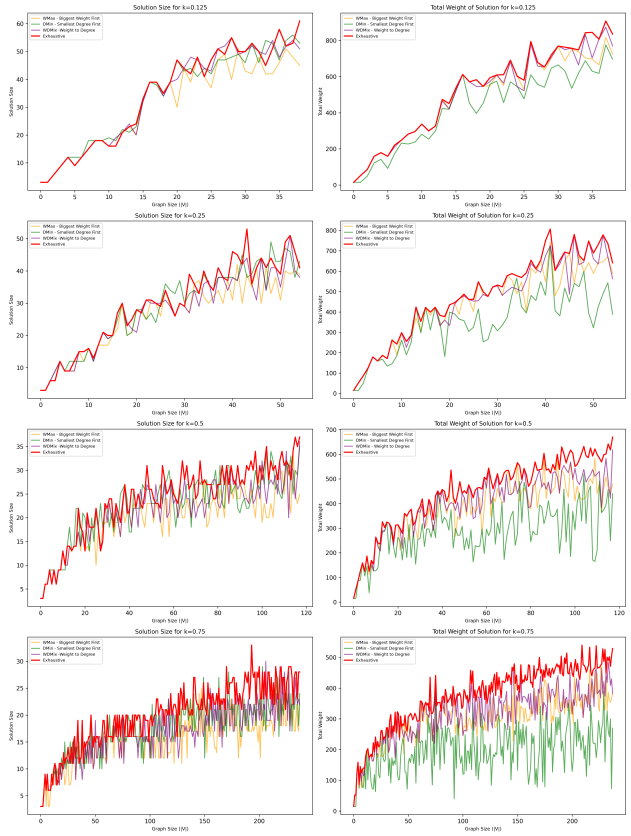

Fig. 6: Solution Size And Total Weight Comparison For All Algorithms

Disclaimer: *The graphs appear denser for higher values of k because the exhaustive algorithm was able to solve larger graphs, meaning more data was able to be compared.*

	k=0.125				k=0.25			
Algorithm	Solution Size	Total Weight	Speed	Average	Solution Size	Total Weight	Speed	Average
Exhaustive	1.51	1.0	3.46	1.99	1.42	1.0	3.62	2.01
WDMix	1.64	1.79	1.0	1.48	2.05	1.95	1.0	1.67
WMax	2.33	1.74	1.0	1.69	2.53	1.96	1.11	1.87
DMin	2.0	3.77	1.0	2.26	1.78	3.51	1.0	2.1

TABLE I: Rankings for k=0.125 and k=0.25

	k=0.50				k=0.75			
Algorithm	Solution Size	Total Weight	Speed	Average	Solution Size	Total Weight	Speed	Average
Exhaustive	1.4	1.0	3.75	2.05	1.27	1.0	3.82	2.03
WDMix	2.13	2.04	1.21	1.79	2.33	2.05	1.26	1.88
WMax	2.71	2.4	1.06	2.06	2.84	2.44	1.09	2.12
DMin	2.13	3.67	1.04	2.28	2.19	3.79	1.14	2.38

TABLE II: Rankings for k=0.50 and k=0.75

Disclaimer: *The rankings range from 1 to 4, with 1 representing the best and 4 the worst. The highest-ranked solutions are those with the maximum weight, largest size, and fastest speed.*

The comparison of execution times between the exhaustive and greedy algorithms is omitted in this section because, for the graph sizes where the exhaustive algorithm could complete within a reasonable time, the heuristic algorithm's execution time was negligible—typically 0.0 seconds. This disparity is so large that including both on the same scale would provide little meaningful insight beyond illustrating that the heuristic is vastly faster.

Let's start by analyzing *Fig.*6, which compares the solution size and total solution weight of the four algorithms:

- In the right column of graphs, the red line consistently shows the highest total solution weight, confirming that the exhaustive algorithm reliably finds the optimal solution. This outcome is expected, given its comprehensive search approach.
- As the graph size increases, the greedy algorithm shows a noticeable decline in solution weight. A similar trend is observed as the value of k increases. This indicates that while the greedy approach is computationally efficient, it struggles to maintain solution quality for larger or denser graphs.
- Both WDMix and WMax rank as the top performers in terms of solution weight, with WDMix slightly outperforming WMax as k increases. On the other hand, DMin consistently ranks the lowest in solution weight by a noticeable margin.
- Notably, WMax shows frequent fluctuations in solution size, indicating some instability in terms of maximizing the number of vertices in the solution.
- The evolution of the graphs suggests that the total solution weight increases at a diminishing rate as the number of nodes grows, indicating that it may eventually plateau for larger graphs.

The graphs reveal distinct patterns, allowing us to rank each algorithm based on three key metrics: maximum solution weight, largest solution size, and fastest execution speed. This visual analysis is corroborated by the data in *TABLE I* and *TABLE II*, which validate several of these observations:

- WDMix achieves the highest average ranking across all four values of k, making it the top choice in terms of balancing solution quality and stability.
- The greedy algorithms, as expected, are significantly faster than the exhaustive approach. However, the exhaustive algorithm consistently ranks highest in solution size across all values of k, suggesting that the optimal solution also tends to maximize size. This could provide interesting insights when comparing with the Maximum Independent Set problem for the same graphs.
- While WDMix and WMax have similar rankings, WDMix tends to yield heavier solutions as both graph size and k increase. This consistency positions WDMix as the more reliable choice for larger

or denser graphs.
- Surprisingly, the DMin heuristic performed worse than anticipated in terms of solution size. This is unexpected, as one might assume that selecting nodes with the smallest degree would consistently yield larger independent sets.

Lastly, let's turn our attention to precision. Our analysis so far suggests that greedy algorithms are an effective choice for solving the Maximum Weighted Independent Set (MWIS) problem, with WDMix emerging as the best-performing heuristic among the three we tested. Now, we need to assess the accuracy of the heuristics by evaluating the margin of error between the greedy and exhaustive algorithms to determine how closely the heuristics approximate the optimal solution.

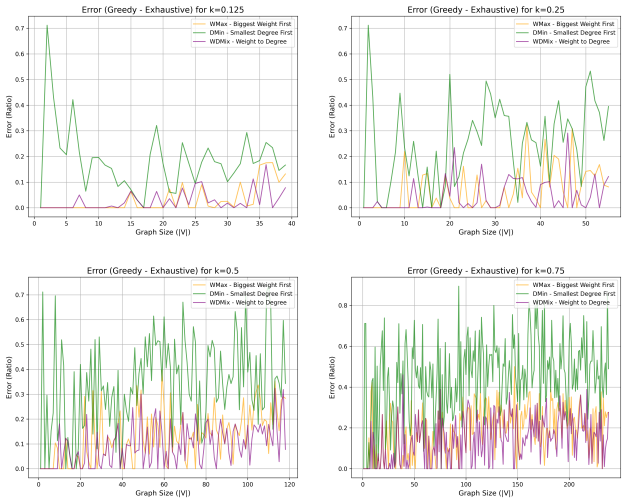

Fig. 7: Error Ratio Of Greedy Algorithms For Each Value Of k

	k=0.125			k=0.25		
Algorithm	Average Deviation	Maximum Deviation	Accuracy	Average Deviation	Maximum Deviation	Accuracy
WDMix	17.77	142	0.49	26.58	200	0.44
WMax	24.28	148	0.59	40.24	241	0.47
DMin	88.00	222	0.05	130.13	367	0.11

TABLE III: Deviation and Accuracy Metrics for k=0.125 and k=0.25

	k=0.50			k=0.75		
Algorithm	Average Deviation	Maximum Deviation	Accuracy	Average Deviation	Maximum Deviation	Accuracy
WDMix	46.72	197	0.24	58.88	207	0.18
WMax	66.09	213	0.21	75.51	228	0.11
DMin	162.52	457	0.06	176.31	431	0.03

TABLE IV: Deviation and Accuracy Metrics for k=0.50 and k=0.75

Fig. 7 illustrates the error ratio when comparing each heuristic's solution weight to the optimal solution obtained by the exhaustive algorithm. A higher ratio indicates a greater deviation from the optimal solution, suggesting a less accurate heuristic.

TABLE III and *TABLE IV* provide further insights into the deviations and accuracy of the heuristics. The first column displays the average deviation, identifying which heuristic generally comes closest to the optimal solution. The second column presents the maximum deviation, highlighting the heuristic with the largest outliers. Finally, the third column shows accuracy, defined as the percentage of instances in which each algorithm finds the optimal solution, relative to the total number of graphs solved.

From this data, we can draw the final conclusions regarding which heuristic performs better:

- Across all values of k, the Weight To Degree (WDMix) algorithm has the lowest average deviation and maximum deviation, making it the most reliable heuristic for approximate solutions to the Maximum Weighted Independent Set problem.
- As k increases, we see the accuracy decline for all algorithms. For WMax, it decreases from 0.59 at k=0.125 to 0.11 at k=0.75, nearly an 81% drop. Similarly, we also see this decline in WDMix, although it retains more accuracy than the other heuristics, dropping only around 64%.
- Focusing on the DMin, we confirm once more that this heuristics performs poorly and is not well-suited for these type of problems.
- Higher values of k generate the biggest outliers, proven by the steady increase in the maximum deviation.

IV. Conclusion

In this study, I explored two distinct approaches for solving the Maximum Weighted Independent Set (MWIS) problem: the exhaustive and greedy algorithms. The experiments demonstrated that the naive exhaustive approach can be significantly optimized through backtracking and pruning techniques. These enhancements not only reduced execution times but also enabled the solution of larger graphs, particularly those with higher density. For the greedy algorithm, I evaluated three heuristics: WMax (prioritizing maximum node weight), DMin (choosing minimum node degree), and WDMix (using a weight-to-degree ratio). Although similar in implementation, each heuristic exhibited unique performance characteristics. Through experimental analysis across multiple metrics—such as speed, solution quality, and error ratio—WDMix consistently emerged as the most effective heuristic, offering the best balance between solution weight and computational efficiency.

In conclusion, the optimized exhaustive algorithm is ideal for smaller or denser graphs where exact solutions are feasible. However, for larger or sparser graphs where exhaustive search becomes impractical, the greedy algorithm with the WDMix heuristic provides a reliable alternative, delivering near-optimal solutions efficiently, especially when perfect accuracy is not critical.

V. Future Work

The next project will be about designing and testing randomized algorithms for the same combinatorial problem (MWIS). As such, I intend on using the

knowledge acquired in this work to continue exploring the Maximum Weighted Independent Set problem, and see if I can find better ways to reach solutions faster and more accurately. I also want to do more research on this topic and implement algorithms published by other researchers, which was something I meant to do in this work but didn't have the time. [4]

References

[1] Deepak Puthal, Surya Nepal, Cecile Paris, Rajiv Ranjan, and Jinjun Chen, "Efficient algorithms for social network coverage and reach", *Proceedings - 2015 IEEE International Congress on Big Data, BigData Congress 2015*, pp. 467–474, 8 2015.

[2] Pedro V. Sander, Diego Nehab, Eden Chlamtac, and Hugues Hoppe, "Efficient traversal of mesh edges using adjacency primitives", *ACM Transactions on Graphics*, vol. 27, 12 2008.
URL: `https://dl.acm.org/doi/10.1145/1409060.1409097`

[3] Peter Gartland, Daniel Lokshtanov, Tomáš Masařík, Marcin Pilipczuk, Michał Pilipczuk, and Paweł Rzazewski, "Maximum weight independent set in graphs with no long claws in quasi-polynomial time", *Proceedings of the Annual ACM Symposium on Theory of Computing*, pp. 683–691, 6 2024.
URL: `https://dl.acm.org/doi/10.1145/3618260.3649791`

[4] Yuanyuan Dong, Andrew V. Goldberg, Alexander Noe, Nikos Parotsidis, Mauricio G.C. Resende, and Quico Spaen, "A local search algorithm for large maximum weight independent set problems", *Leibniz International Proceedings in Informatics, LIPIcs*, vol. 244, pp. 45:1–45:16, 9 2022.