Randomized Algorithms for Independent Set Decision Problem

Daniel Jorge Bernardo Ferreira

Abstract **– This study explores Randomized algorithms for the Independent Set Decision Problem, comparing their performance with Brute Force and Greedy algorithms. Focusing on efficiency, scalability, and overall effectiveness, we evaluate these algorithms across diverse graph instances. Our findings provide insights into the applicability of Randomized techniques, contributing to algorithmic design and real-world implementations.**

Resumo **– Este estudo explora algoritmos aleatórios para o problema de decisão de conjuntos independentes, comparando o seu desempenho com algoritmos exaustivos e gulosos. Com foco na eficiência, escalabilidade e eficácia geral, avaliamos esses algoritmos em diversas instâncias de grafos. As nossas descobertas fornecem informações sobre a aplicabilidade de técnicas aleatórias, contribuindo para a conceção de algoritmos e implementações no mundo real.**

Keywords **– Independent Set, Decision Problem, NP-hard, NP-complete, Randomized Algorithms**

Palavras chave **– Conjunto Independente, Problema de Decisão, NP-difícil, NP-completo, Algoritmos Aleatórios**

I. Introduction

In mathematics, graph theory is the study of graphs, which are mathematical structures used to model pairwise relations between objects. Graphs can be used to model many types of relations and processes in the real world, ranging from social networks and transportation systems to biological interactions [1], [2], [3].

One important concept in graph theory is that of an independent set. The independent set of a graph also known as a stable set, coclique, or anticlique is a subset of vertices, such that no two vertices are adjacent. There are several computational problems related to independent sets that have been studied. The most famous of these might be the Maximum Independent Set Problem, an optimization problem that seeks to find the largest possible independent set in a given graph. The problem is known to be NP-hard, meaning that there is no known efficient algorithm that can solve it in polynomial time [4].

The Independent Set Decision Problem is essentially a differently formulated version of the Maximum Independent Set Problem. It arises from the observation that a graph possesses an independent set of at least k vertices if and only if it harbors an independent set of exactly k vertices. This observation underscores the equivalence of these two problems, as the formulation of the Independent Set Decision Problem retains the computational complexity inherent in the Maximum Independent Set Problem.

In this paper, we explore the effectiveness of Randomized algorithms to solve the Independent Set Decision Problem, emphasizing their theoretical foundations and conducting a comprehensive analysis.

II. Problem Definition

An independent set in a graph $G = (V, E)$ is a subset $I \subseteq V$ such that for all $x, y \in I, \{x, y\} \notin E$. The independent set problem consists of finding the largest independent set within a given graph. This optimization problem can be reformulated as a decision problem represented by the following language:

$$S = \{\langle G, k\rangle \mid \text{there exists an independent set } I \subseteq V(G) \text{ such that } |I| \geq k\} \tag{1}$$

where $\langle G, k\rangle$ is a tuple encoding a graph $G = (V, E)$ and an integer k. The language S is true if and only if there exists an independent set in the graph G with size of at least k. In simpler terms, the Independent Set Decision Problem seeks to answer whether it is possible to select a group of k vertices in a graph such that none of them are adjacent, meaning there are no edges connecting them.

III. Preliminaries

Let $G = (V, E)$ stand for a simple undirected graph with a set V of vertices and a set E of edges. Let $|G|$ denote $|V|$. We will use n to denote $|V| = |G|$. The vertex set and edge set of a graph G are denoted by $V(G)$ and $E(G)$, respectively. For simplicity, we may denote a singleton set $\{v\}$ by v.

For a vertex v in a graph G, we define the following notations. Let $\delta(v) = |N(v)|$ denote the degree of v, and $V_d(v)$ denote the set of vertices with distance exactly d from v.

We will use k to represent the target size of an independent set. In cases where k is an integer, it indicates the number of nodes required to form the independent set. If k is a float, it represents the proportion of nodes contributing to the independent set, given by $k \cdot 100\%$. We may also use ε_p to denote the percentage of edges in a graph relative to the maximum number of edges possible given n vertices.

The terms "vertex" and "node" (as well as their plural forms) will be used interchangeably. Furthermore,

we might use "ISDP" to refer to the Independent Set Decision Problem.

IV. DATASETS

This section presents the datasets used in our analysis, offering a diverse collection of graph instances. These datasets, varying in size, density, and origin, contribute to a comprehensive evaluation of the Independent Set Decision Problem.

A. RGPI

The dataset by Ernesto Parra Inza, titled "Random Graph (1)", provides a collection of instances generated using various pseudo-random methods. Two specific folders, BD0 and BD1, were used from this dataset, each comprising 100 graphs ranging from 50 to 545 nodes. For further details, refer to the original dataset [5].

B. AAT1

The AAT1 dataset focuses on the Independent Set Decision Problem [6]. It includes 1588 graphs ranging from 4 to 400 nodes. For each number of nodes, there are four graphs, each with different levels of density (0.125, 0.25, 0.5, and 0.75 of the maximum possible number of edges). This design provides a diverse set of instances for analysis, capturing variations in both size and density

C. SW

Two graphs from the computational tests conducted by Sedgewick and Wayne were also included in our analysis. Unfortunately, the specific reference is not available. These graphs, denoted as SW graphs, include a tiny graph with 13 nodes and a medium-sized graph with 250 nodes.

V. ALGORITHMICS

A. Randomized Algorithm

A Randomized algorithm is an algorithm that makes certain decisions based on the outcome of a random process, such as tossing a coin, drawing a card, or generating a random number. A Randomized algorithm can be classified into two types: Las Vegas and Monte Carlo. A Las Vegas algorithm always produces the correct answer, but its execution time can vary depending on the random choices. A Monte Carlo algorithm can produce an incorrect answer with some probability, but its running time is fixed or limited. Both algorithms can be useful for solving difficult problems, depending on the trade-off between accuracy and efficiency.

In our case, we implemented a Monte Carlo algorithm to address the Independent Set Decision Problem. The algorithm was tested over multiple graph instances with varying numbers of nodes n and densities ε_p, iterations I, and target independent set sizes k. The parameters k and ε_p both assumed values of 0.125, 0.25, 0.5, and 0.75. The number of iterations I was set to 50, 100, 500, 1,000, 2,500, 5,000, 7,500, 10,000, 12,500, 15,000, 17,500, and 20,000 for the fixed iteration setting. For the dynamic iteration setting, I was determined by the percentage of total vertex combinations with size k in the input graph. The specified percentages include 0.10%, 0.25%, 0.50%, 0.75%, 1.00%, 2.50%, 5.00%, 7.50%, and 10.00%. The maximum number of attempts to generate a unique subset M was set to 10. Furthermore, we performed 10 repetitions R for each instance to guarantee the reliability and convergence of our results. Consequently, the reported metrics such as execution time and success rate represent the average values obtained from these 10 runs.

The algorithm, described in Algorithm 1, operates by repeatedly generating random subsets of the graph's vertices until an independent set of size k is found. The algorithm maintains a set of used subsets to ensure that each subset considered during the iterations is unique, preventing redundancy. For each generated subset, it checks whether it forms an independent set by verifying the absence of edges between all pairs of vertices in the subset. The algorithm terminates once it discovers an independent set or completes the predefined number of iterations. This probabilistic approach provides a practical solution with controllable computational costs.

Algorithm 1 Randomized Algorithm for ISDP

```
procedure HasIndependentSet(G, k, N)
    usedSubsets ← ∅
    for i ← 1 to N do
        repeat
            S ← RandomSubset(V(G), k)
        until S ∉ usedSubsets
        usedSubsets ← usedSubsets ∪ {S}
        isIndependentSet ← True
        for u ∈ S do
            for v ∈ S do
                if u ≠ v and ¬HasEdge(G, u, v) then
                    isIndependentSet ← False
        if isIndependentSet then
            return True
    return False
```

VI. TIME COMPLEXITY

A. Basic Operations

In this section, we examine the number of basic operations performed by the Randomized algorithm. We consider the following operations as basic: generating a random subset, checking whether a subset is already in use, and checking whether a subset forms an independent set. We compare the empirical results with the theoretical complexity of the algorithm, which is given by $O(n \cdot I \cdot M \cdot (n \cdot k)^2)$, or simply $O(n^3)$ when I, M, and k are constants.

In Fig. 1, we observe that the number of basic operations increases rapidly, following a cubic trend up

to $n \approx 15$, after which it stabilizes. This behavior is expected, given that the number of iterations I is constrained to 20,000. We also note a correlation between the increase in the number of basic operations and the simultaneous rise in k along with the decrease in ε_p. This is attributed to the reduced probability of short-circuiting during the assessment of subset independence.

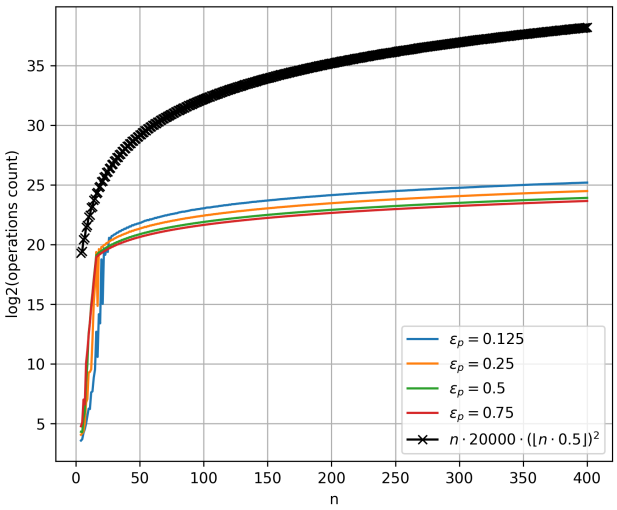

Fig. 1: Comparing the Number of Basic Operations of a Randomized Algorithm Against Formal Proof Results with a Fixed Number of Iterations (20,000) and a Parameter Setting of $k = 0.5$.

For the dynamic iteration setting, the number of basic operations increases factorially, conforming to a binomial distribution with parameters n and k, as shown in Fig. 2. Therefore, the number of basic operations peaks for $k = 0.5$.

The number of basic operations exhibits greater variance for smaller values of ε_p. This phenomenon arises from the increased probability of identifying an independent set in less dense graphs, prompting the algorithm to terminate prematurely upon finding one. Nevertheless, smaller values of ε_p should present a worst-case scenario for the number of basic operations, as the algorithm is less likely to short-circuit during the evaluation of subset independence.

B. Execution Time

In this section, we present an overview of the runtime performance of the Randomized algorithm. The computations were performed on an Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, with a clock speed of 2.59 GHz. The system had a total RAM capacity of 16.0 GB, with 15.8 GB available for use. The operating system used was Windows 11 Home, version 22H2, running on a 64-bit architecture. The Python version in play was 3.11.

Table I shows the performance metrics for the fixed iteration setting with 20,000 iterations for each k value. The algorithm achieves faster execution times for lower

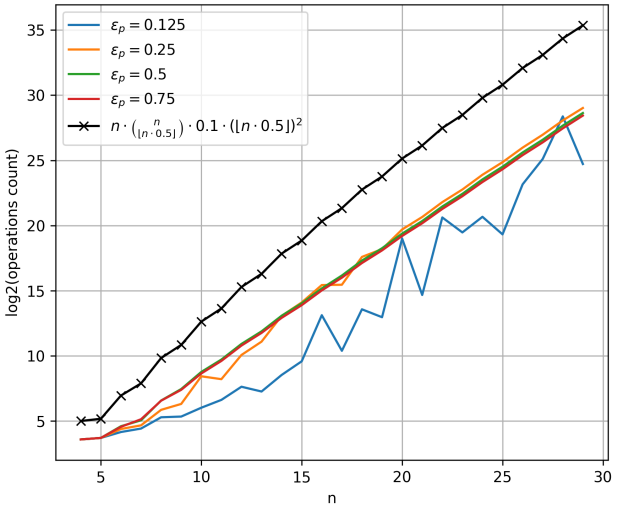

Fig. 2: Comparing the Number of Basic Operations of a Randomized Algorithm Against Formal Proof Results with 10% of Total Vertex Combinations as Iterations and a Parameter Setting of $k = 0.5$.

k values, specifically 0.125. The standard deviation is positively correlated with k. This suggests a higher degree of unpredictability or dispersion in the execution times for larger k values.

TABLE I: Randomized Algorithm Performance Metrics (k values with 20,000 iterations)

k	*min*	*max*	*mean*	*std*
0.125	0.00	$8.57 \cdot 10^{-1}$	$4.31 \cdot 10^{-1}$	$2.18 \cdot 10^{-1}$
0.25	0.00	1.39	$7.13 \cdot 10^{-1}$	$3.35 \cdot 10^{-1}$
0.5	0.00	2.03	1.08	$4.95 \cdot 10^{-1}$
0.75	0.00	2.75	1.42	$6.67 \cdot 10^{-1}$

In Table II, we observe that the execution time remains consistent across different ε_p values. Additionally, there is a negative correlation between the standard deviation and ε_p suggesting a greater predictability or concentration in execution times for larger ε_p values.

TABLE II: Randomized Algorithm Performance Metrics (ε_p values with 20,000 iterations)

ε_p	*min*	*max*	*mean*	*std*
0.125	0.00	2.75	$9.14 \cdot 10^{-1}$	$6.14 \cdot 10^{-1}$
0.25	0.00	2.68	$9.10 \cdot 10^{-1}$	$5.95 \cdot 10^{-1}$
0.5	0.00	2.73	$9.12 \cdot 10^{-1}$	$5.85 \cdot 10^{-1}$
0.75	0.00	2.70	$9.13 \cdot 10^{-1}$	$5.81 \cdot 10^{-1}$

In Fig. 3, we observe the execution time for $k = 0.5$ as a function of n and ε_p. The execution time increases rapidly until $n \approx 15$, and then stabilizes, confirming the cubic trend drafted in the previous section.

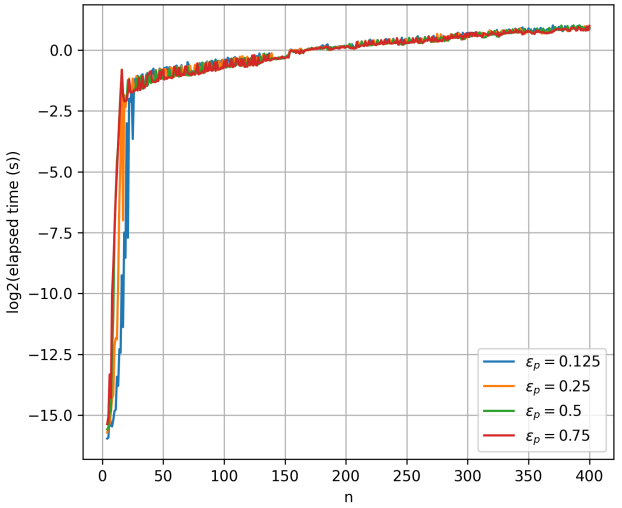

Fig. 3: Execution Time of the Randomized Algorithm with a Fixed Number of Iterations (20,000) and a Parameter Setting of $k = 0.5$.

In the dynamic iteration setting, the runtime performance metrics for various values of k are presented in Table III, where 10% of total vertex combinations serve as iterations. As described in the previous section, the algorithm demonstrates a higher execution time as k converges towards 0.5. The standard deviation also experiences an upward trend as k approaches 0.5, suggesting a higher degree of unpredictability for values closer to 0.5.

TABLE III: Randomized Algorithm Performance Metrics (k values with 10% of Total Vertex Combinations as Iterations)

k	*min*	*max*	*mean*	*std*
0.125	0.00	$4.01 \cdot 10^{-4}$	$2.26 \cdot 10^{-5}$	$6.20 \cdot 10^{-5}$
0.25	0.00	2.53	$1.11 \cdot 10^{-1}$	$4.50 \cdot 10^{-1}$
0.5	0.00	$1.03 \cdot 10^{2}$	7.28	$2.05 \cdot 10^{1}$
0.75	0.00	$1.02 \cdot 10^{1}$	$6.84 \cdot 10^{-1}$	2.00

The analysis of Table IV does not seem to reveal a significant correlation between execution time and ε_p. The reality is that the performance varies notably across different ε_p values, contingent upon the chosen k value. For instance, with $k = 0.125$ and $k = 0.25$, higher ε_p values result in increased execution times. In contrast, for $k = 0.75$, execution time remains relatively constant across ε_p values, and for $k = 0.5$, the execution time exhibits greater variability with smaller ε_p values. However, it is important to acknowledge that the available data may not be conclusive enough to establish a definitive relationship between ε_p and execution time.

In Fig. 4, the execution time for $k = 0.5$ is depicted as a function of both n and ε_p with 10% of the total vertex combinations serving as iterations. The execution time remains relatively constant for each ε_p value, except for $\varepsilon_p = 0.125$ which displays a more erratic and unpredictable behavior.

TABLE IV: Randomized Algorithm Performance Metrics (ε_p values with 10% of Total Vertex Combinations as Iterations)

ε_p	*min*	*max*	*mean*	*std*
0.125	0.00	$6.10 \cdot 10^{1}$	$9.73 \cdot 10^{-1}$	6.15
0.25	0.00	$1.03 \cdot 10^{2}$	2.38	$1.21 \cdot 10^{1}$
0.5	0.00	$1.01 \cdot 10^{2}$	2.36	$1.17 \cdot 10^{1}$
0.75	0.00	$1.03 \cdot 10^{2}$	2.37	$1.18 \cdot 10^{1}$

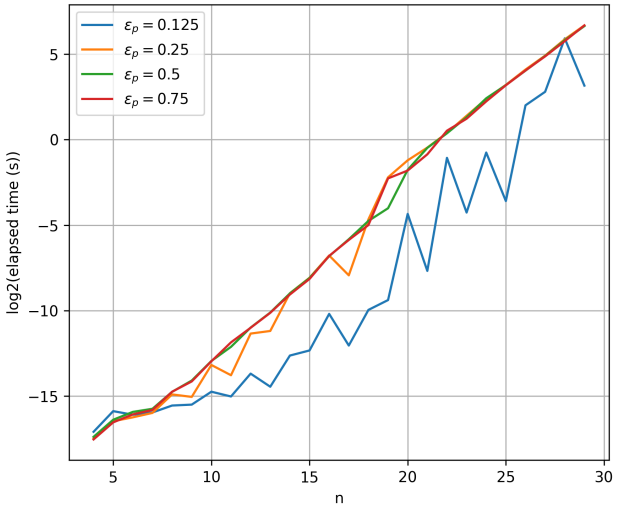

Fig. 4: Execution Time of the Randomized Algorithm with 10% of Total Vertex Combinations as Iterations and a Parameter Setting of $k = 0.5$.

VII. CORRECTNESS

In this section, our objective is to assess the correctness of the Randomized algorithm, where correctness is defined as the probability of the algorithm generating the accurate output for a given input. Specifically, the desired output is a boolean value indicating whether an independent set of size k exists in the input graph. The correctness of the Randomized algorithm is closely tied to the number of iterations I. A higher number of iterations generally leads to an increased probability of successfully identifying an independent set of size k within the input graph.

The confusion matrix presented in Table V shows the number of true positives, false positives, true negatives, and false negatives for the fixed iteration setting with 20,000 iterations. It was generated by comparing the output of the Randomized algorithm with the output of a Branching algorithm, an exact algorithm of the Independent Set Decision Problem [6].

The algorithm failed a total of 146 times. It started to fail consistently from $n \approx 50$. Beyond this thresh-

TABLE V: Randomized Algorithm Confusion Matrix (20,000 iterations)

		Predicted	
		1	0
Actual	1	466	146
	0	0	1212

old, the number of iterations I proved insufficient to reliably ensure the identification of an independent set. The distribution of these errors across different k and ε_p values is shown in Tables VI and VII, respectively.

The algorithm failed predominantly for $k = 0.125$ and $k = 0.25$, with these two cases contributing to 96.6% of the total number of errors. It is important to acknowledge that the absence of failures for $k = 0.5$ and $k = 0.75$ may be attributed to the fact that these cases are inherently more inclined to produce a False outcome. Consequently, the occurrence of false negatives is less likely in these scenarios.

TABLE VI: Distribution of Errors across Different k Values for the Randomized Algorithm with a Fixed Number of Iterations (20,000)

k	TP	FP	TN	FN
0.125	277	0	117	62
0.25	128	0	249	79
0.5	48	0	403	5
0.75	13	0	443	0

The edge density ε_p also appears to influence the correctness of the algorithm, with a notable prevalence of failures for lower ε_p values. This phenomenon aligns with the hypothesis provided in the preceding paragraph. As the edge density increases, the probability of identifying an independent set of size k diminishes, thereby increasing the likelihood of the algorithm producing a False outcome and subsequently reducing the probability of a False Negative.

TABLE VII: Distribution of Errors across Different ε_p Values for the Randomized Algorithm with a Fixed Number of Iterations (20,000)

ε_p	TP	FP	TN	FN
0.125	196	0	190	70
0.25	134	0	269	53
0.5	83	0	353	20
0.75	53	0	400	3

The cumulative number of errors for $k = 0.125$ across different n and ε_p for the fixed iteration setting is shown in Fig. 5.

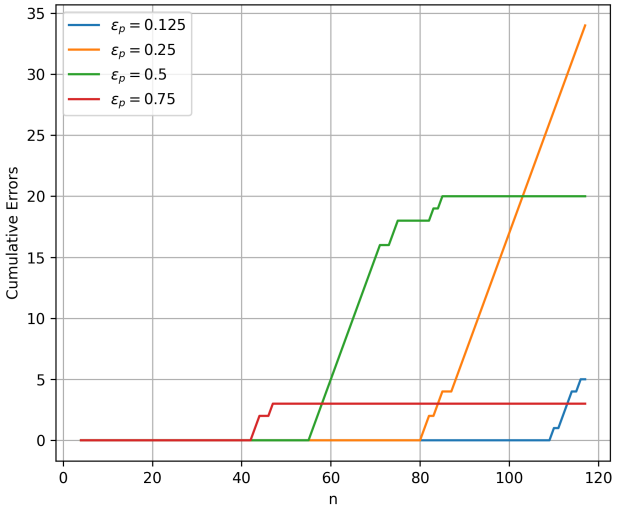

Fig. 5: Cumulative Errors of the Randomized Algorithm with a Fixed Number of Iterations (20,000) and a Parameter Setting of $k = 0.125$.

The evaluation of the Randomized algorithm under different I values within the fixed iteration setting is depicted in Table VIII. The data is unbalanced, with the number of True Negatives far exceeding the number of True Positives, rendering accuracy an unreliable metric for assessing the correctness of the algorithm. The algorithm never produces False Positives, meaning that the precision is always 1.0. In this context, every error corresponds to a False Negative. Therefore, recall emerges as a suitable metric for gauging the correctness of the algorithm. In line with our hypothesis, the recall demonstrates an upward trend with an increase in the number of iterations I, reaching 76.1% for $I = 20,000$.

TABLE VIII: Randomized Algorithm Performance Metrics (Fixed Iterations)

I	$Acc.$	P	R	$F1$
50	$8.60 \cdot 10^{-1}$	1.00	$5.82 \cdot 10^{-1}$	$7.36 \cdot 10^{-1}$
100	$8.68 \cdot 10^{-1}$	1.00	$6.08 \cdot 10^{-1}$	$7.56 \cdot 10^{-1}$
500	$8.86 \cdot 10^{-1}$	1.00	$6.60 \cdot 10^{-1}$	$7.95 \cdot 10^{-1}$
1000	$8.94 \cdot 10^{-1}$	1.00	$6.85 \cdot 10^{-1}$	$8.13 \cdot 10^{-1}$
2500	$9.04 \cdot 10^{-1}$	1.00	$7.12 \cdot 10^{-1}$	$8.32 \cdot 10^{-1}$
5000	$9.12 \cdot 10^{-1}$	1.00	$7.39 \cdot 10^{-1}$	$8.50 \cdot 10^{-1}$
7500	$9.16 \cdot 10^{-1}$	1.00	$7.48 \cdot 10^{-1}$	$8.56 \cdot 10^{-1}$
10000	$9.16 \cdot 10^{-1}$	1.00	$7.50 \cdot 10^{-1}$	$8.57 \cdot 10^{-1}$
12500	$9.18 \cdot 10^{-1}$	1.00	$7.55 \cdot 10^{-1}$	$8.60 \cdot 10^{-1}$
15000	$9.18 \cdot 10^{-1}$	1.00	$7.55 \cdot 10^{-1}$	$8.60 \cdot 10^{-1}$
17500	$9.19 \cdot 10^{-1}$	1.00	$7.58 \cdot 10^{-1}$	$8.62 \cdot 10^{-1}$
20000	$9.20 \cdot 10^{-1}$	1.00	$7.61 \cdot 10^{-1}$	$8.65 \cdot 10^{-1}$

Table IX presents the confusion matrix corresponding to the dynamic iteration setting, where 10% of total vertex combinations are used as iterations. The algorithm only failed 4 times. It failed twice for $n = 6$ and

one time for both $n = 9$ and $n = 12$. The breakdown of these errors across various k values is illustrated in Table X, while Table XI delves into the distribution across different ε_p values. It is worth noting that the execution of the algorithm, for this particular setting, was limited to graphs with up to 29 nodes, resulting in a relatively small sample size. Consequently, the data may not be conclusive enough to establish any definitive conclusions.

TABLE IX: Randomized Algorithm Confusion Matrix (10% of Total Vertex Combinations as Iterations)

		Predicted	
		1	0
Actual	1	251	4
	0	0	161

The maxinum number of times the algorithm failed for any given k value was 2, which occurred for $k = 0.75$. However, the limited occurrence of errors makes it challenging to derive substantial insights into the relationship between k and the correctness of the algorithm.

TABLE X: Distribution of Errors across Different k Values for the Randomized Algorithm with 10% of Total Vertex Combinations as Iterations

k	TP	FP	TN	FN
0.125	104	0	0	0
0.25	88	0	15	1
0.5	48	0	55	1
0.75	11	0	91	2

The same observation applies to the distribution of errors across different ε_p values. The algorithm encountered a maximum of 2 errors for any specific ε_p value. The low incidence of errors limits the statistical significance needed for a robust analysis of the relationship. The most we can infer is that the algorithm appears to have an exceedingly low likelihood of failure.

TABLE XI: Distribution of Errors across Different ε_p Values for the Randomized Algorithm with 10% of Total Vertex Combinations as Iterations

ε_p	TP	FP	TN	FN
0.125	86	0	18	0
0.25	68	0	35	1
0.5	55	0	47	2
0.75	42	0	61	1

Figure 6 illustrates the cumulative occurrence of errors for $k = 0.25$ across various n and ε_p values in the dynamic iteration setting. It specifically highlights a singular occurrence of error, observed at $n = 12$.

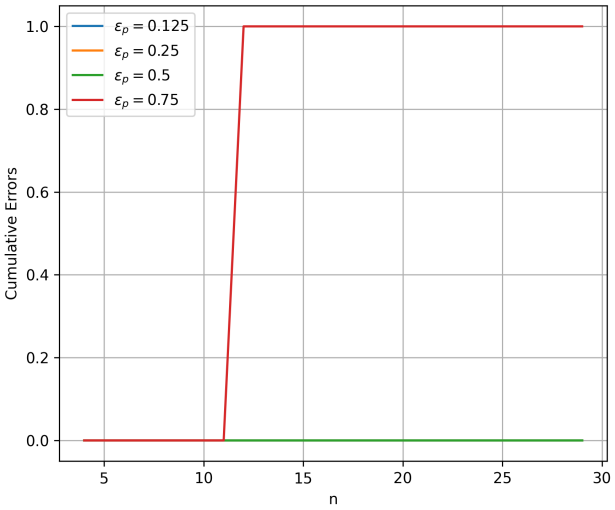

Fig. 6: Cumulative Errors of the Randomized Algorithm with 10% of Total Vertex Combinations as Iterations and a Parameter Setting of $k = 0.25$.

The performance metrics for all the I values considered in the dynamic iteration setting are presented in Table XII. Similar to our observations in the fixed iteration setting, the recall of the algorithm demonstrates improvement with higher I values, reaching 98.4% for $I = 10\%$. This surpasses the results achieved in the fixed iteration setting, where the recall peaked at 76.1%. However, it is crucial to bear in mind that the number of samples was considerably smaller for the dynamic iteration setting, which may have skewed the results.

TABLE XII: Randomized Algorithm Performance Metrics (% of Total Vertex Combinations as Iterations)

I(%)	$Acc.$	P	R	$F1$
0.10	$8.99 \cdot 10^{-1}$	1.00	$8.35 \cdot 10^{-1}$	$9.10 \cdot 10^{-1}$
0.25	$9.30 \cdot 10^{-1}$	1.00	$8.86 \cdot 10^{-1}$	$9.40 \cdot 10^{-1}$
0.50	$9.45 \cdot 10^{-1}$	1.00	$9.10 \cdot 10^{-1}$	$9.53 \cdot 10^{-1}$
0.75	$9.50 \cdot 10^{-1}$	1.00	$9.18 \cdot 10^{-1}$	$9.57 \cdot 10^{-1}$
1.00	$9.52 \cdot 10^{-1}$	1.00	$9.22 \cdot 10^{-1}$	$9.59 \cdot 10^{-1}$
2.50	$9.66 \cdot 10^{-1}$	1.00	$9.45 \cdot 10^{-1}$	$9.72 \cdot 10^{-1}$
5.00	$9.81 \cdot 10^{-1}$	1.00	$9.69 \cdot 10^{-1}$	$9.84 \cdot 10^{-1}$
7.50	$9.86 \cdot 10^{-1}$	1.00	$9.76 \cdot 10^{-1}$	$9.88 \cdot 10^{-1}$
10.00	$9.90 \cdot 10^{-1}$	1.00	$9.84 \cdot 10^{-1}$	$9.92 \cdot 10^{-1}$

VIII. Comparison with Other Algorithms

In this section, we compare the performance of the Randomized algorithm against other algorithms, namely the Brute Force, Branching, Greedy, and Improved Greedy algorithms [6]. We take into account both the time complexity and the correctness of the

algorithms to determine the most suitable algorithm for solving the Independent Set Decision Problem.

In terms of time complexity, the Brute Force algorithm is the most inefficient, with a time complexity of $O(2^n/\sqrt{n} \cdot k^2)$ after applying the Stirling approximation [7], followed by the Randomized algorithm under a dynamic iteration setting. The most efficient algorithms are the Greedy and Improved Greedy algorithms, with a time complexity of $O(n^2)$. This is illustrated in Fig. 7, where the measured execution time of the Randomized algorithm is compared against the other algorithms for a parameter setting of $k = 0.5$ and $\varepsilon_p = 0.5$.

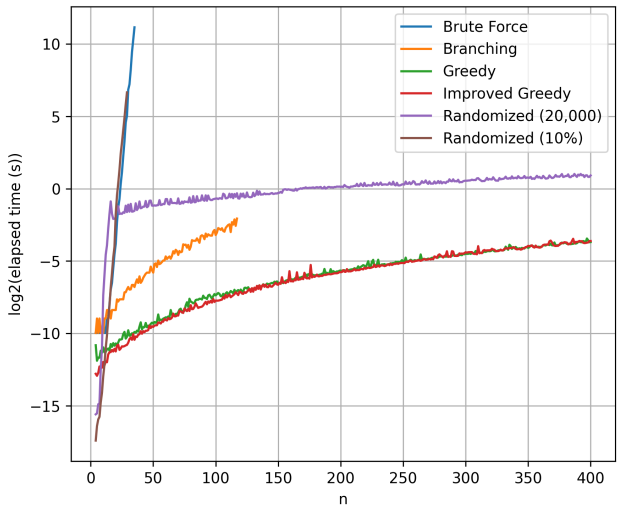

Fig. 7: Execution Time Comparison of the Randomized Algorithm Against Other Algorithms with a Parameter Setting of $k = 0.5$ and $\varepsilon_p = 0.5$

Table XIII outlines the performance metrics for the Randomized algorithm in comparison to other algorithms. The table includes the largest graph size successfully processed, the maximum execution time recorded, and the recall for each algorithm.

In terms of correctness, the Brute Force and Branching algorithms are the most reliable, with a perfect recall of 100%. The Randomized algorithm follows closely, achieving a recall of 98.4% under dynamic iteration settings. However, under a fixed iteration setting, the recall of the Randomized algorithm drops to 76.1%, which is significantly lower than the other algorithms.

Determining the best algorithm, considering both time complexity and correctness, involves a trade-off analysis. It depends on the specific requirements and priorities of our application. For instance, if we are dealing with a small graph, the Brute Force algorithm may be the most suitable option, as it guarantees perfect correctness. However, if we are dealing with a large graph, the Brute Force algorithm may not be feasible due to its high time complexity.

If a balance between correctness and time complexity is crucial, the Improved Greedy algorithm may be a good choice as it achieves high recall with a relatively low time complexity. However, if perfect correctness is paramount and time is not a significant concern, the Brute Force or Branching algorithm might be preferable.

TABLE XIII: Performance Metrics of the Randomized Algorithm Against Other Algorithms

Algorithm	***Nodes***	***Time***	***Recall***
Brute Force	35	$5.82 \cdot 10^3$	1.00
Branching	117	$8.67 \cdot 10^3$	1.00
Greedy	400	$1.41 \cdot 10^{-1}$	$9.38 \cdot 10^{-1}$
Improved Greedy	400	$1.33 \cdot 10^{-1}$	$9.56 \cdot 10^{-1}$
Random (20,000)	400	2.75	$7.61 \cdot 10^{-1}$
Random (10%)	29	$1.03 \cdot 10^2$	$9.84 \cdot 10^{-1}$

The Randomized algorithm is more difficult to assess, as its performance is contingent upon the number of iterations I. Additional research would be required to figure out the ideal number of iterations for a specific input. Only then can one determine the suitability of the Randomized algorithm for a given application in comparison to other algorithms.

IX. Scaling For Larger Problems

In this section, we leverage the Levenberg-Marquardt algorithm to project the computational time needed for solving larger instances of problems [8]. The Levenberg-Marquardt algorithm is used to find the set of parameters that minimizes the sum of squared residuals, representing the disparities between observed and predicted values in a given model.

Table XIV displays the predicted coefficients for the Randomized algorithm with a fixed number of iterations (20,000) under different k and ε_p values. The mapping function used to generate the predicted coefficients is defined as follows:

$$f(n, k, I, M, a, b) = a \cdot n \cdot I \cdot M \cdot \lfloor n \cdot k \rfloor^2 + b \qquad (2)$$

The number of iterations I is fixed at 20,000, the maximum number of attempts M is constant at 10, and k is always smaller than n. Consequently, the mapping function for predicting coefficients simplifies to:

$$f(n, a, b) = a \cdot n^3 + b \qquad (3)$$

The Levenberg-Marquardt optimizes the parameters a and b to best approximate the given data. The r^2 values provide an assessment of the goodness of fit, indicating the proportion of the variance in the data that is captured by the model. In this case, the r^2 values are relatively modest, ranging from 54.6% to 80.2%, suggesting that the conceived model is not a perfect fit for the data. The r^2 values seem to increase as k increases. This trend may be attributed to the higher volatility of the algorithm for smaller k values, posing a greater challenge in predicting the execution time accurately.

TABLE XIV: Randomized Algorithm Predicted Coefficients with a Fixed Number of Iterations (20,000)

k	ε_p	a	b	r^2
0.125	0.125	$3.41 \cdot 10^{-11}$	$2.42 \cdot 10^{-1}$	$5.68 \cdot 10^{-1}$
	0.25	$2.95 \cdot 10^{-11}$	$2.79 \cdot 10^{-1}$	$5.46 \cdot 10^{-1}$
	0.5	$2.68 \cdot 10^{-11}$	$3.09 \cdot 10^{-1}$	$5.56 \cdot 10^{-1}$
	0.75	$2.52 \cdot 10^{-11}$	$3.20 \cdot 10^{-1}$	$5.78 \cdot 10^{-1}$
0.25	0.125	$1.32 \cdot 10^{-11}$	$4.43 \cdot 10^{-1}$	$7.01 \cdot 10^{-1}$
	0.25	$1.28 \cdot 10^{-11}$	$4.53 \cdot 10^{-1}$	$7.20 \cdot 10^{-1}$
	0.5	$1.23 \cdot 10^{-11}$	$4.68 \cdot 10^{-1}$	$7.46 \cdot 10^{-1}$
	0.75	$1.23 \cdot 10^{-11}$	$4.74 \cdot 10^{-1}$	$7.71 \cdot 10^{-1}$
0.5	0.125	$4.91 \cdot 10^{-12}$	$6.98 \cdot 10^{-1}$	$7.74 \cdot 10^{-1}$
	0.25	$4.76 \cdot 10^{-12}$	$6.97 \cdot 10^{-1}$	$7.84 \cdot 10^{-1}$
	0.5	$4.84 \cdot 10^{-12}$	$6.85 \cdot 10^{-1}$	$8.00 \cdot 10^{-1}$
	0.75	$4.74 \cdot 10^{-12}$	$6.88 \cdot 10^{-1}$	$7.84 \cdot 10^{-1}$
0.75	0.125	$2.91 \cdot 10^{-12}$	$9.18 \cdot 10^{-1}$	$8.00 \cdot 10^{-1}$
	0.25	$2.89 \cdot 10^{-12}$	$9.01 \cdot 10^{-1}$	$8.03 \cdot 10^{-1}$
	0.5	$2.88 \cdot 10^{-12}$	$8.94 \cdot 10^{-1}$	$8.03 \cdot 10^{-1}$
	0.75	$2.89 \cdot 10^{-12}$	$8.94 \cdot 10^{-1}$	$8.02 \cdot 10^{-1}$

The predicted execution time for the Randomized algorithm with a fixed number of iterations (20,000) and a parameter setting of $k = 0.5$ is shown in Fig. 8.

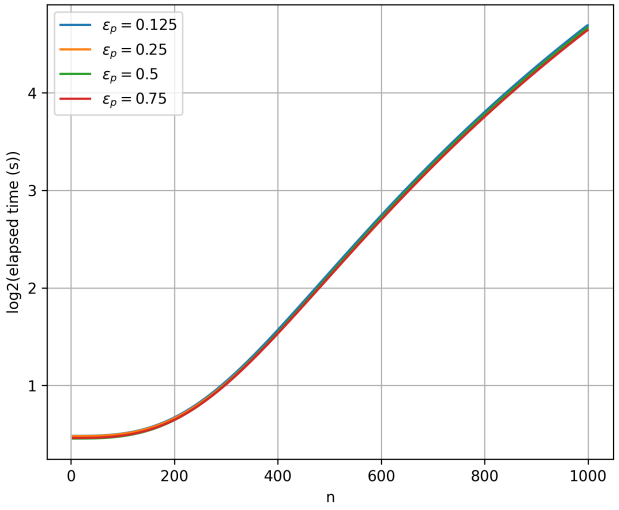

Fig. 8: Predicted Time Complexity for the Randomized Algorithm with a Fixed Number of Iterations (20,000) and a Parameter Setting of $k = 0.5$.

The predicted execution time for the Randomized algorithm, considering 10% of total vertex combinations as iterations, is summarized in Table XV. In contrast to the fixed iteration setting, the number of iterations I is now dependent on both the number of vertices n and the parameter k rather than being a constant value. The initial mapping function is defined as:

$$f(n, k, I, M, a, b) = a \cdot n \cdot \binom{n}{\lfloor n \cdot k \rfloor} \cdot I \cdot M \cdot \lfloor n \cdot k \rfloor^2 + b \tag{4}$$

This can be further simplified to:

$$f(n, k, a, b) = a \cdot n^3 \cdot \binom{n}{\lfloor n \cdot k \rfloor} + b \tag{5}$$

The r^2 values for the predicted coefficients consistently surpass 90%, indicating a strong fit of the model to the data. This suggests that the model adeptly captures the relationship between the input variables and expected execution time. There is a notable exception to this trend, with the r^2 value for $k = 0.125$ and $\varepsilon_p = 0.125$ dropping to a mere 50.8%.

TABLE XV: Randomized Algorithm Predicted Coefficients with 10% of Total Vertex Combinations as Iterations

k	ε_p	a	b	r^2
0.125	0.125	$1.00 \cdot 10^{-10}$	$4.39 \cdot 10^{-6}$	$5.08 \cdot 10^{-1}$
	0.25	$1.57 \cdot 10^{-10}$	$4.43 \cdot 10^{-6}$	$6.99 \cdot 10^{-1}$
	0.5	$4.31 \cdot 10^{-10}$	$5.79 \cdot 10^{-6}$	$8.33 \cdot 10^{-1}$
	0.75	$3.59 \cdot 10^{-9}$	$7.15 \cdot 10^{-6}$	$8.87 \cdot 10^{-1}$
0.25	0.125	$8.01 \cdot 10^{-13}$	$1.51 \cdot 10^{-5}$	$8.32 \cdot 10^{-1}$
	0.25	$1.56 \cdot 10^{-11}$	$3.80 \cdot 10^{-5}$	$9.82 \cdot 10^{-1}$
	0.5	$1.20 \cdot 10^{-8}$	$-4.91 \cdot 10^{-3}$	$9.90 \cdot 10^{-1}$
	0.75	$1.17 \cdot 10^{-8}$	$2.24 \cdot 10^{-2}$	$9.89 \cdot 10^{-1}$
0.5	0.125	$6.86 \cdot 10^{-10}$	$9.64 \cdot 10^{-1}$	$2.96 \cdot 10^{-1}$
	0.25	$2.42 \cdot 10^{-9}$	1.07	$9.89 \cdot 10^{-1}$
	0.5	$2.35 \cdot 10^{-9}$	1.04	$9.89 \cdot 10^{-1}$
	0.75	$2.38 \cdot 10^{-9}$	$9.24 \cdot 10^{-1}$	$9.92 \cdot 10^{-1}$
0.75	0.125	$1.88 \cdot 10^{-9}$	$6.39 \cdot 10^{-2}$	$9.94 \cdot 10^{-1}$
	0.25	$1.87 \cdot 10^{-9}$	$4.25 \cdot 10^{-2}$	$9.97 \cdot 10^{-1}$
	0.5	$1.78 \cdot 10^{-9}$	$6.60 \cdot 10^{-3}$	$9.94 \cdot 10^{-1}$
	0.75	$1.79 \cdot 10^{-9}$	$3.92 \cdot 10^{-2}$	$9.97 \cdot 10^{-1}$

The predicted execution time for the Randomized algorithm with 10% of total vertex combinations as iterations and a parameter setting of $k = 0.5$ is shown in Fig. 9.

X. CONCLUSION

While the Randomized algorithm shows potential, it currently lags behind alternatives in efficiency and correctness. The Improved Greedy algorithm emerges as a more suitable choice for real-world applications, offering a better balance between computational time and solution accuracy. Further research may refine the performance of the Randomized algorithm, but its current applicability seems limited compared to other existing algorithms.

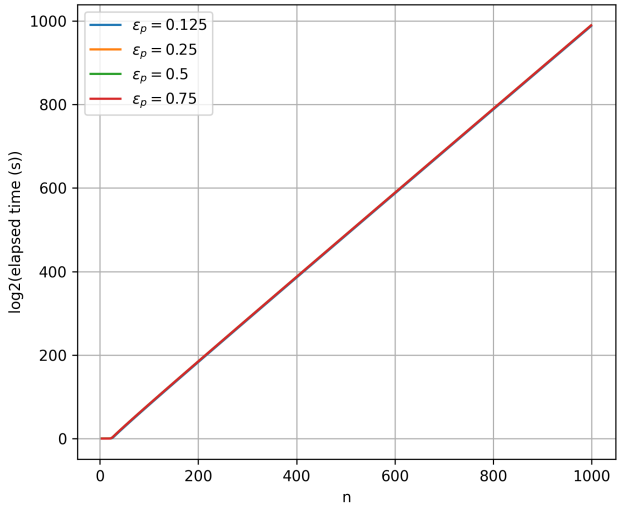

Fig. 9: Predicted Time Complexity for the Randomized Algorithm with 10% of Total Vertex Combinations as Iterations and a Parameter Setting of $k = 0.5$.

References

[1] E. Otte and R. Rousseau, *Social network analysis: a powerful strategy, also for the information sciences*, *Journal of Information Science*, **28**(6), 441-453, 2002.

[2] M. Barthélemy, *Spatial networks*, *Physics Reports*, **499**(1-3), 1-101, 2011.

[3] A. D. McNaught, *Compendium of chemical terminology*, **1669**, Oxford: Blackwell Science, 1997.

[4] M. R. Garey and D. S. Johnson, *"Strong" NP-completeness results: Motivation, examples, and implications*, *Journal of the ACM (JACM)*, **25**(3), 499-508, 1978.

[5] Ernesto Parra Inza, *Random Graph (1)*, `https://doi.org/10.17632/rr5bkj6dw5.3`, Mendeley Data, V3, 2022.

[6] D. Ferreira, *Independent Set Decision Problem*, *First project of Advanced Algorithms*, Nov. 2023.

[7] Wikipedia, *Stirling's Approximation*, `https://en.wikipedia.org/wiki/Stirling%27s_approximation`, Accessed on 12 Dec. 2023.

[8] Wikipedia, *Levenberg–Marquardt Algorithm*, `https://en.wikipedia.org/wiki/Levenberg%E2%80%93Marquardt_algorithm`, Accessed on 4 Dec. 2023.