# Independent Set Decision Problem

Daniel Jorge Bernardo Ferreira

***Resumo*** **- Este artigo conduz uma investigação focada no Problema de Decisão de Conjuntos Independentes e nas suas soluções algorítmicas, estabelecendo uma clara demarcação entre abordagens exactas e aproximadas. Os algoritmos exactos, incluindo métodos tradicionais e avanços recentes, são analisados quanto à sua otimização em estruturas de grafos. Simultaneamente, as técnicas de aproximação, tais como os algoritmos gulosos e aleatórios, são avaliadas quanto à relação entre eficiência computacional e qualidade da solução.**

***Abstract*** **- This paper conducts a focused investigation into the Independent Set Decision Problem and its algorithmic solutions, drawing a clear demarcation between exact and approximation approaches. Exact algorithms, including traditional methods and recent advancements, are scrutinized for optimality within graph structures. Simultaneously, approximation techniques, such as greedy and randomized algorithms, are evaluated for their trade-offs between computational efficiency and solution quality.**

***Keywords*** **– Independent Set, Decision Problem, Maximum Independent Set, NP-hard, NP-complete, Exhaustive Search, Greedy Heuristics**

## 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 to each other.

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 will focus on the independent set decision problem. Our primary goal is to investigate the problem, examining its theoretical foundations and providing a comprehensive analysis of existing algorithms.

## II. PROBLEM DEFINITION

An independent set in a graph $G = (V, E)$ is a subset $\mathrm{I} \subseteq \mathrm{V}$ such that for all $\mathrm{x}, \mathrm{y} \in \mathrm{I}$, $\{x, y\} \notin E$.

The independent-set problem consists of finding a maximum independent set in a graph. This optimization problem can be transformed into the decision problem corresponding to the following language:

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

Here, $\langle \mathrm{G}, \mathrm{k}\rangle$ is a tuple encoding a graph $G = (V, E)$ and an integer $k$. The language IndSet is true if and only if there exists an independent set in the graph $G$ with size at least $k$.

In simple terms, the independent set decision problem seeks to answer whether it is possible to pick a group of $k$ vertices in a graph where none of them are connected by an edge.

## IV. 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 $O_n(v)$ denote the set of vertices with distance exactly $n$ from $v$.

We may 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 $\mathrm{k} \times 100\%$.

We may also use $\varepsilon_p$ 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.

## V. ALGORITHMICS

In our study, we introduce four distinct algorithms capable of solving the independent set decision problem: Brute Force, Branching, Greedy, and an improved Greedy variant. These algorithms can be easily adapted to tackle the maximum independent set problem, given its inherent connection to the decision problem.

### *A. Exhaustive Search*

In the context of the independent set decision problem, an exhaustive search refers to a brute-force approach where all possible combinations of $k$ vertices are examined to determine whether they form an independent set or not. This approach is conceptually simple, but impractical for large graphs due to the combinatorial explosion of possibilities. It involves generating all possible subsets of $k$ vertices from the graph. The time complexity of this operation is proportional to the binomial coefficient "$n$ choose $k$". This coefficient represents the number of ways to pick $k$ elements from a set of $n$ elements and follows the formula:

$$\binom{n}{k} = \frac{n!}{(n-k)!} \tag{1}$$

The worst-case time complexity of the algorithm is dominated by the generation of these subsets. For each subset, the algorithm checks whether it forms an independent set by examining all possible pairs of vertices within the subset. This additional check introduces a factor of $k^2$ in the overall time complexity. Therefore, the total time complexity of the algorithm can be expressed as $O\left(\binom{n}{k} \cdot k^2\right)$. This renders the algorithm unsuitable for large-scale applications, where the sheer number of potential combinations renders the computational cost prohibitively high.

### *B. Branching Algorithm*

In contrast to exhaustive search, a branching algorithm for the independent set decision problem takes a more targeted approach to explore the solution space efficiently. The algorithm strategically makes decisions at each step, branching into subproblems and avoiding the need to examine all possible combinations while still retaining an exact solution.

The algorithm begins by examining the degrees of nodes in the graph. It strategically selects nodes with degrees satisfying specific conditions, introducing branching based on the degree distribution. Nodes with degrees 0 or 1 are immediately selected, forming the basis of a solution. A node with degree 0 is inherently independent, and a node with degree 1, despite having one adjacent node, can be chosen without loss of generality. As the algorithm encounters nodes with degrees 3 or more, it adopts a more nuanced approach. It branches into two distinct paths. First, it strategically includes the selected node in the independent set solution and excludes it along with its neighbors from the graph. This inclusion accounts for the worst-case scenario, where a node of degree 3 or more necessitates the removal of itself and its multiple neighbors, contributing to the time complexity $T(n-4)$. Alternatively, the algorithm explores the subproblem by excluding only the node itself. This branch acknowledges that, in certain scenarios, excluding the node may lead to a more favorable independent set. The time complexity associated with this exclusion is $T(n-1)$.

In the unique scenario where every node in the graph has a degree of 2, the graph is decomposed into connected components, resembling cycles. An independent set can be efficiently derived in linear time by selecting half of the nodes within each connected component. Mathematically, this is expressed as:

$$k = \sum_{c \in C} \left\lfloor \frac{|V(c)|}{2} \right\rfloor \tag{2}$$

The algorithm concludes its execution when it successfully identifies a valid independent set of size $k$ or exhaustively traverses the solution space, adhering to the specified criteria.

The overall time complexity of the algorithm is expressed by the recurrence relation:

$$T(n) = T(n-1) + T(n-4) + n^c \tag{3}$$

Here, c represents a constant factor associated with the recursive calls. The given relation is an abstraction of the behavior of the algorithm, reflecting the branching into two distinct paths when dealing with nodes of degree 3 or more, and the inclusion/exclusion strategies discussed earlier [5]. It translates to the characteristic equation:

$$x^4 = x^3 + 1 \tag{4}$$

To analyze the time complexity of this recurrence relation, we can use generating functions. Let $X(z)$ be the generating function for the sequence $T(n)$. Multiplying both sides of the recurrence relation $z^n$ and summing over all $n$, we get:

$$X(z) = z + z^4 + \sum_{n=1}^{\infty} (T(n-1)z^n + T(n-4)z^n) + \sum_{n=1}^{\infty} n^c z^n \tag{5}$$

Simplifying and rearranging, we arrive at:

$$X(z) = z + z^4 + z\sum_{n=0}^{\infty} T(n)z^n + z^4\sum_{n=0}^{\infty} T(n)z^n + \sum_{n=1}^{\infty} n^c\, z^n \quad (6)$$

Now, recognizing that $\sum_{n=0}^{\infty} T(n)z^n$ is simply $X(z)$, we can substitute to get:

$$X(z) = z + z^4 + zX(z) + z^4X(z) + \sum_{n=1}^{\infty} n^c\, z^n \quad (7)$$

Rearranging terms and factoring $X(z)$, we get:

$$X(z)(1 - z - z^4) = z + z^4 + \sum_{n=1}^{\infty} n^c\, z^n \quad (8)$$

Now, we know that $1 - z - z^4$ has roots $x1$, $x2$, $x3$, $x4$, and the solution for $X(z)$ involves partial fractions. Solving for $x$, we obtain:

$$X(z) = \frac{z + z^4}{(1 - x_1 z)(1 - x_2 z)(1 - x_3 z)(1 - x_4 z)} \quad (9)$$

The roots of $1 - z - z^4$ are complex, and the dominant root, denoted as $x$ is approximately 1.38. Therefore, the time complexity of the algorithm is $O^*(1.38^n)$, where $n$ is the number of nodes in the graph. This is significantly better than the brute force approach, as shown in Fig. 1.

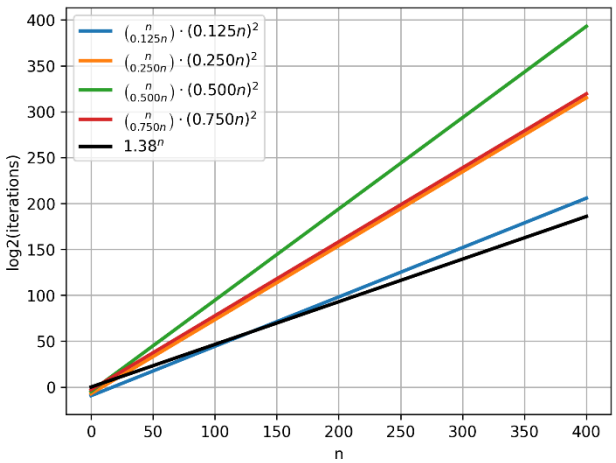


Fig. 1 – Formal analysis of Brute Force and Branching algorithms

### *C. Greedy Heuristics Algorithm*

A greedy algorithm is a heuristic approach to problem-solving that makes locally optimal choices at each step with the hope of finding a global optimum. In the context of the independent set decision problem, a greedy algorithm is particularly appealing due to its simplicity and efficiency. Instead of exhaustively exploring all possible combinations, the algorithm takes a pragmatic approach by iteratively selecting nodes with the minimum degree in the graph. This local optimization aims to minimize potential constraints on independence, as nodes with lower degrees are less likely to be connected to each other. The selected nodes are incrementally added to the solution until either the desired independent set size, $k$, is reached or all nodes in the graph have been considered.

The time complexity of the algorithm is determined by the key operations executed within its iterative process, primarily involving the identification of the node with the minimum degree and the subsequent removal of its adjacency from the graph. Finding the minimum degree node is a linear operation, taking $O(n)$ time in the worst case, where $n$ is the number of nodes in the graph. Removing the adjacency of a node is also a linear operation. This is because, in the worst scenario, the graph is complete, mandating iteration over all nodes. The algorithm continues its iterative process until the termination conditions are met, resulting in a total of $n$ iterations in the worst case. Therefore, the overall time complexity of the algorithm can be expressed as $O(n^2)$.

Despite its significant speed advantage over the previous algorithms, there is a potential downside in terms of false negatives, meaning there is a risk of not finding a solution under certain circumstances. The smallest graph for which greedy can fail, where a sequence of choices leads to a non-optimal solution, has six vertices and is shown in Fig. 2. The first real counter-example, depicted in Fig. 3, has seven vertices with any sequence of choices leading hreedy to a non-optimal solution.

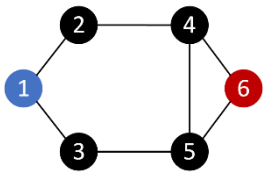


Fig. 2 – Smallest graph where greedy can fail. Picking 1 leads to a solution of size 2, while 6 gives a solution of size 3.

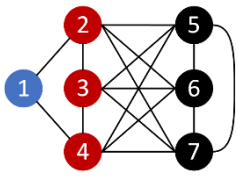


Fig. 3 – Smallest graph where greedy fails. Any solution contains 1 and has size 2, while the optimal solution contains the vertices 2, 3, and 4.

According to Håstad's theorem [6], the greedy algorithm takes a leading role among approximation algorithms when tackling the independent set problem. This result holds significance not only for the decision problem but extends to the broader domain of optimization.

The theorem establishes that the greedy algorithm is an $n$-approximation algorithm, implying that the size of the independent set it produces is, at most, $n$ times larger than the size of the optimal independent set in the graph. In practical terms, if OPT represents the size of the maximum independent set, the theorem guarantees that the solution size, $S$, satisfies $S \leq n \times OPT$. This means that while the greedy algorithm may not always find the optimal solution, the size of its solution is bounded by a factor of $n$ times the optimal size.

Additionally, Håstad's theorem offers valuable insights into the limits of approximation algorithms for the independent set problem. It proves that, within the class of polynomial-time approximation algorithms, Greedy achieves a particularly favorable balance between solution quality and computational efficiency. The theorem also sets a boundary on the achievable approximation ratio, asserting that no polynomial-time algorithm can achieve a substantially better approximation factor than $n^{1-\varepsilon}$, where $\varepsilon > 0$. This result highlights the inherent challenges in improving the approximation factor within polynomial time for the independent set problem.

### *D. Improved Greedy Heuristics Algorithm*

Building upon the foundational principles of greedy heuristics, the improved algorithm refines the node selection process, dynamically adjusting criteria during each iteration. It places specific focus on scenarios where multiple vertices share the same degree, addressing a limitation observed in the previous version of the algorithm where always picking a minimum degree vertex might not guarantee an optimal solution.

The algorithm introduces a novel tie-breaking strategy, when faced with degree ties between two vertices that relies on the concept of $n$th-degree. The $n$th-degree of a vertex $v$, denoted as $d_n(v)$, represents the sum of the degrees of vertices adjacent to those considered for the $(n-1)$th degree, not including vertices already considered in a lesser degree. This is formally expressed as:

$$d_n(v) = \sum_{u \in O_n(v)} \delta(u) \qquad (10)$$

When faced with degree ties between vertices $v1$ and $v2$, the algorithm compares their $n$th-degrees. If $d_n(v1) = d_n(v2)$, the tie-breaking strategy extends to the $(n+1)$th degree. In cases where $n + 1$ is even, the algorithm selects the vertex with the greater $(n+1)$th degree. Conversely, when $n + 1$ is odd, the algorithm opts for the vertex with the lesser $(n+1)$th degree. This tie-breaking mechanism is purposefully designed to enhance induced subgraphs by considering higher-degree properties when resolving ties. For instance, in the scenario where the two vertices share identical primary degrees but $d_2(v1) > d_2(v2)$, selecting $v1$ eliminates vertices with a greater total degree. This results in a less connected induced subgraph, characterized by fewer edges. The reduced edge density implies that the sum of the degrees of the remaining vertices is diminished. Those vertices, with lower degree sums, are prioritized as better candidates for selection in subsequent steps.

The improved algorithm shares a comparable worst-case time complexity of $O(n^2)$ with its initial version. The differences in the node selection criteria and removal processes contribute to variations in their actual performance, depending on the characteristics of the input graph. The benefit of the improved algorithm lies in its increased likelihood of reaching a solution for more complex graphs. Unfortunately, this algorithm, much like the previous one, does not always work [7][8]. For the graph shown in Fig. 4, the algorithm fails on the very first step, because the minimum degree vertex is not in the maximum independent set.

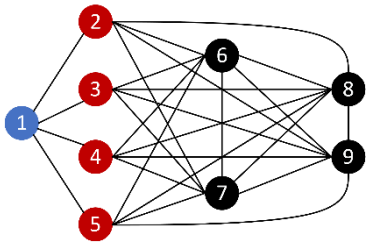


Fig. 4 – Graph where minimum degree vertex is not included in the maximum independent set

## VI. ITERATIONS

In this section, we examine the number of iterations estimated for the brute force and greedy algorithms. The branching and improved greedy algorithms were not assessed in this analysis.

The algorithms were tested in multiple graphs using multiple combinations of parameters, including the number of nodes $(n)$ and edge percentage $(\varepsilon_p)$ of a graph, and the target independent set size $(k)$. The parameters $k$ and $\varepsilon_p$ both assumed values from the set $\{0.125, 0.25, 0.5, 0.75\}$. The range for the number of nodes spanned from 4 to 400.

It is important to note that the figures showcased here exclusively pertain to the scenario where $k = 0.5$. In the context of the independent decision set problem, $k$ represents the proportion $(50\%)$ of nodes targeted in the graph for determining the independent set size. The results obtained for other values of $k$ generally follow the same pattern.

### *A. Exhaustive Search*

In Fig. 5, it is shown a comparative analysis of the results obtained from the implementation of the brute force algorithm against the results derived from the formal proof. The parallel alignment of the line obtained from formal analysis with the empirical results indicates a consistent growth rate in the number of iterations, validating the theoretical expectations. There is a slight discrepancy in intercept, with the formal analysis line positioned slightly lower on average, indicating nuanced variations in computational demands. This disparity may arise from factors such as implementation specifics, or algorithmic optimizations.

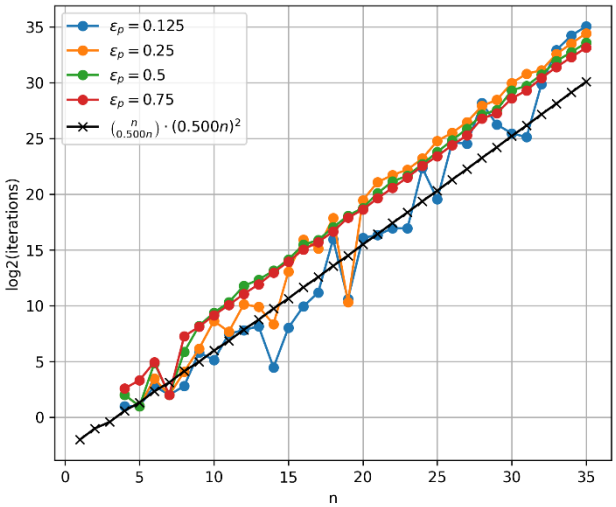


Fig. 5 – Number of iterations for the Brute Force algorithm (k=0.5)

### B. Greedy Heuristics Algorithm

For the greedy heuristics algorithm, the results estimated can be seen in Fig 6. The figure shows a distinctive arc pattern for each line. The nearly coincidental alignment of these arcs implies that, akin to the Brute Force algorithm, the greedy algorithm maintains a reliable growth rate as the proportion of targeted nodes varies.

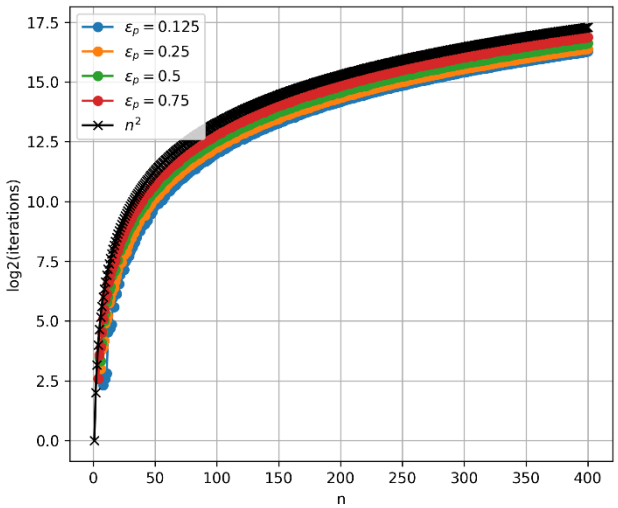


Fig. 6 – Number of iterations for the Greedy Heuristics algorithm (k=0.5)

## VII. ELAPSED TIME

In this section, we provide an overview of the runtime performance associated with each algorithm, delineating the temporal requirements incurred during their execution. The algorithms were tested with the same configuration as the one mentioned in section VI. The processing unit driving the computations was an Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, operating at a clock speed of 2.59 GHz. It had a total RAM capacity of 16.0 GB, with 15.8 GB available for use. The operating system was Windows 11 Home, version 22H2, running on a 64-bit architecture. The Python version was 3.11.

### A. Exhaustive Search

The brute force algorithm managed to solve all the given graphs comprising up to 35 nodes before it started to consume an excessive amount of time and the process was terminated. In Table I, it is shown descriptive statistics of the results obtained for every value of $k$ during the experiment.

TABLE I: Brute Force Elapsed Time Statistics

| $k$ | $min$ | $mean$ | $max$ | $std$ |
|---|---|---|---|---|
| 0.125 | 0.0 | $2.27e-4$ | $4.01e-3$ | $6.19e-4$ |
| 0.25 | 0.0 | $5.25e-1$ | $1.19e1$ | $1.87e0$ |
| 0.5 | 0.0 | $2.12e2$ | $5.82e3$ | $7.57e2$ |
| 0.75 | 0.0 | $4.64e0$ | $9.08e1$ | $1.34e1$ |

The algorithm exhibited increased execution time for $k = 0.5$ in comparison to other values of $k$, differing by two orders of magnitude. This outcome aligns with expectations, given that the peak of a binomial distribution occurs at its central point.

The results indicate an almost consistent growth pattern for every $\varepsilon_p$ across all values of $k$. However, a notable exception to this trend was observed for $k = 0.25$, as illustrated in Fig. 7, where the function displayed a significantly faster growth rate for higher $\varepsilon_p$ values.

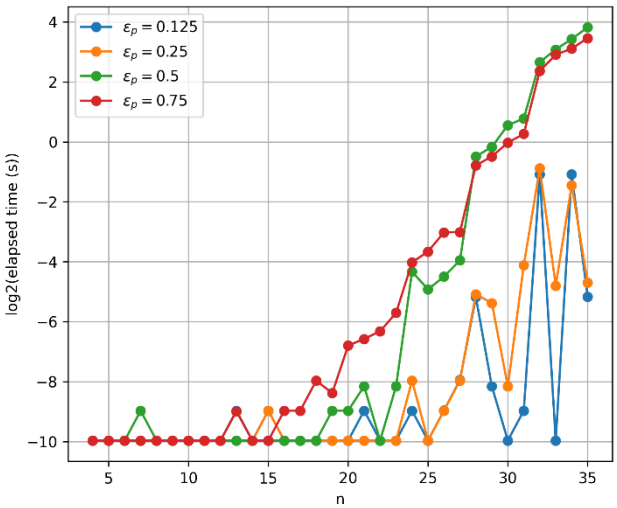


Fig. 7 – Elapsed time for the Brute Force algorithm (k=0.25)

In general, for lower values of $k$, functions with higher $\varepsilon_p$ tend to display faster growth, whereas the opposite is true for higher values of $k$, where functions with lower $\varepsilon_p$ values demonstrate a quicker rate of increase.

### *B. Branching Algorithm*

The branching algorithm, much like the brute force algorithm, gives exact solutions. However, it distinguishes itself by outperforming the latter in terms of speed. It successfully processed all graphs with up to 117 vertices within a reasonable timeframe. For a detailed analysis of the computational efficiency across various $k$ values, refer to Table II.

TABLE II: Branching Elapsed Time Statistics

| $k$ | $min$ | $mean$ | $max$ | $std$ |
|---|---|---|---|---|
| 0.125 | 0.0 | $3.66e-1$ | $7.98e0$ | $1.12e0$ |
| 0.25 | 0.0 | 4.49e1 | $8.67e3$ | $4.60e2$ |
| 0.5 | 0.0 | $2.61e0$ | $7.38e1$ | $9.49e0$ |
| 0.75 | 0.0 | $4.23e-2$ | $4.17e-1$ | $6.14e-2$ |

The algorithm experiences a significant rise in execution time for $k = 0.25$ when compared to other $k$ values, differing by two orders of magnitude. For each value of k, with the exception of $k = 0.125$, the algorithm demonstrated its worst-case performance with lower values of $\varepsilon_p$. There was a substantial increase in the time required, especially evident when $\varepsilon_p$ was set to 0.125. This behavior is illustrated in Fig. 8.

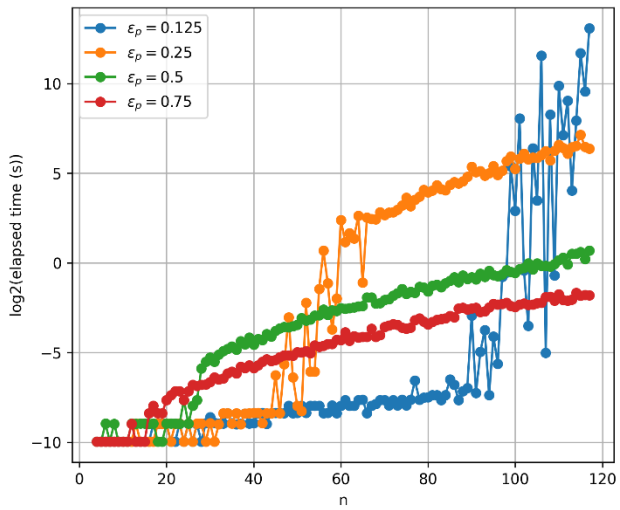


Fig. 8 – Elapsed time for the Branching algorithm (k=0.25)

In the case of $k = 0.125$, the algorithm showed worse performance for higher $\varepsilon_p$ values, as depicted in Fig. 9.

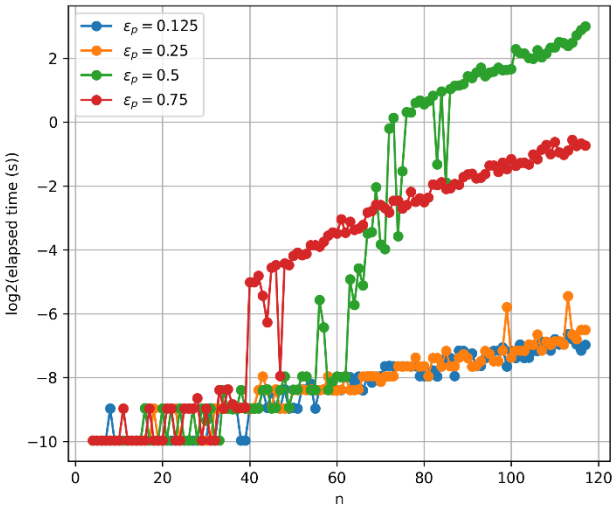


Fig. 9 – Elapsed time for the Branching algorithm (k=0.125)

### *C. Greedy Heuristics Algorithm*

The greedy algorithm successfully solved all the graphs within the dataset, including instances with up to 400 nodes. Table III provides descriptive statistics showcasing the results obtained for each value of $k$ throughout the experiment.

TABLE III: Greedy Elapsed Time Statistics

| $k$ | $min$ | $mean$ | $max$ | $std$ |
|---|---|---|---|---|
| 0.125 | 0.0 | $2.08e-2$ | $1.24e-1$ | $2.49e-2$ |
| 0.25 | 0.0 | $2.07e-2$ | $1.22e-1$ | $2.52e-2$ |
| 0.5 | 0.0 | $2.07e-2$ | $1.22e-1$ | $2.52e-2$ |
| 0.75 | 0.0 | $2.07e-2$ | $1.23e-1$ | $2.52e-2$ |

There was no discernible disparity in runtime for the different $k$ values, but a slight rise in execution time was observed as the $\varepsilon_p$ value increased. In Fig. 10, it is shown a comparative analysis for the different $\varepsilon_p$ values with respect to $k = 0.5$.

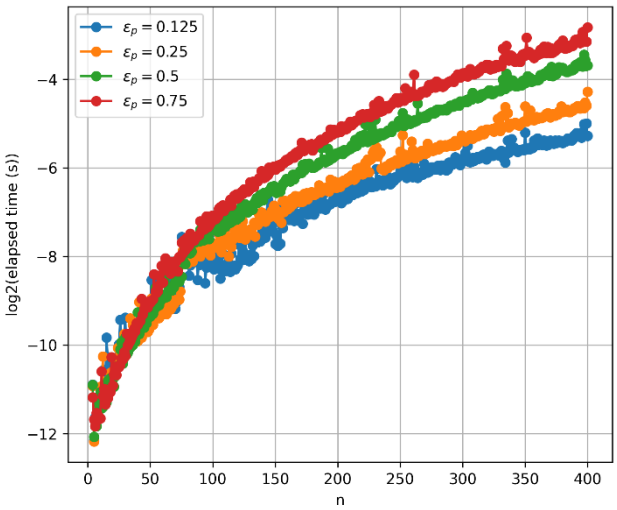

Fig. 10 – Elapsed time for the Greedy algorithm (k=0.5)

The greedy algorithm is classified as an approximation algorithm. In certain scenarios, it may provide solutions that are not necessarily optimal, introducing the possibility of false negatives. This behavior is reflected in the confusion matrix presented in Table IV. The confusion matrix was formed by comparing the results of the branching algorithm with those of the greedy algorithm.

TABLE IV: Greedy Confusion Matrix

| | | Predicted | |
|---|---|---|---|
| | | 1 | 0 |
| Actual | 1 | 574 | 38 |
| | 0 | 0 | 1212 |

The algorithm failed a total of 38 times. It failed 14 times for $k = 0.125$ and 24 times for $k = 0.25$. It failed 12 times for $\varepsilon_p = 0.125$, 8 times for $\varepsilon_p = 0.25$, 14 times for $\varepsilon_p = 0.5$, and 4 times for $\varepsilon_p = 0.75$. It failed twice for $n = 57$. For all other cases, it never failed more than once for the same number of nodes.

The performance metrics for the algorithm can be further assessed by calculating the recall, which is a measure of how well it identifies actual positive instances. It is given by the formula:

$$\mathrm{R} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{11}$$

In this case, with $TP = 574$ and $FN = 38$, we get a recall value of 0.938. This implies that the algorithm successfully identified approximately 93.8% of the actual positive instances.

*D. Improved Greedy Heuristics Algorithm*

The improved version of the greedy algorithm, like its predecessor, also successfully solved all the graphs within the dataset. Table V presents descriptive statistics, providing an overview of the results obtained for each value of $k$ during the experiment.

TABLE V: Improved Greedy Elapsed Time Statistics

| $k$ | $min$ | $mean$ | $max$ | $std$ |
|---|---|---|---|---|
| 0.125 | 0.0 | $2.23e-2$ | $1.28e-1$ | $2.52e-2$ |
| 0.25 | 0.0 | $2.24e-2$ | $1.21e-1$ | $2.55e-2$ |
| 0.5 | 0.0 | $2.24e-2$ | $1.20e-1$ | $2.56e-2$ |
| 0.75 | 0.0 | $2.22e-2$ | $1.21e-1$ | $2.55e-2$ |

The algorithm also demonstrated consistent runtime across various $k$ values with a slight uptick as the $\varepsilon_p$ value increased. This trend is shown in Fig. 11, for different $\varepsilon_p$ values, with $k = 0.5$ as a reference.

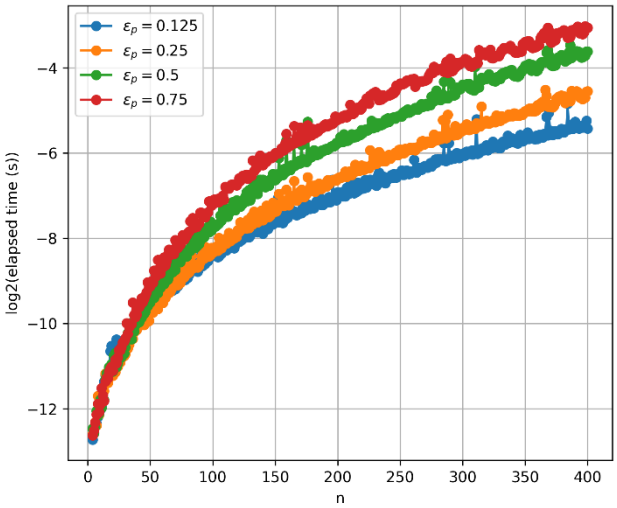


Fig. 11 – Elapsed time for the Improved Greedy algorithm (k=0.5)

The major difference between the improved algorithm and the original version is the node selection criteria which promised to decrease the occurrence of false negatives. Table VI represents the confusion matrix obtained for the improved algorithm.

TABLE VI: Improved Greedy Confusion Matrix

| | | Predicted | |
|---|---|---|---|
| | | 1 | 0 |
| Actual | 1 | 585 | 27 |
| | 0 | 0 | 1212 |

The improved algorithm failed 27 times, which is 11 times fewer than the original algorithm, or about 28.95% less. It failed 10 times for $k = 0.125$ and 17 times for $k = 0.25$. It failed 10 times for $\varepsilon_p = 0.125$, 6 times for $\varepsilon_p = 0.25$, 8 times for $\varepsilon_p = 0.5$, and 3 times for $\varepsilon_p = 0.75$. It also failed twice for $n = 57$, but never more than once for any other number of nodes.

The recall value for this algorithm is 0.956, indicating that it correctly identified around 95.6% of the positive cases.

## IX. SCALING FOR LARGER PROBLEMS

In this section, we employ the Levenberg-Marquardt algorithm to project the computational time needed for solving larger instances of problems, specifically focusing on the brute force and greedy algorithms with $k = 0.5$. The Levenberg-Marquardt algorithm is used to find the set of parameters that minimizes the sum of the squares of the

residuals (the differences between the observed and predicted values) for a given model.

### *A. Exhaustive Search*

In Table VII, we showcase the results obtained from a curve fitting analysis, using data derived from the brute force algorithm. The employed function is defined as:

$$\mathrm{f(n,k)} = \mathrm{a} \cdot \binom{\mathrm{n}}{\mathrm{k} \cdot \mathrm{n}} \cdot \mathrm{k}^2 + \mathrm{b} \qquad (12)$$

The curve fit was executed with the goal of optimizing the parameter $a$ 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.

TABLE VII: Brute Force Predicted Coefficients

| $k$ | $\varepsilon_p$ | $a$ | $b$ | $r^2$ |
|---|---|---|---|---|
| 0.5 | 0.125 | $5.12e-6$ | $-3.96e1$ | $9.85e-1$ |
| | 0.25 | $3.43e-6$ | $4.40e-1$ | $9.99e-1$ |
| | 0.5 | $2.01e-6$ | 4.03 | $9.97e-1$ |
| | 0.75 | $1.47e-6$ | 3.85 | $9.98e-1$ |

A visual representation of the curve fitting results is shown in Fig. 12.

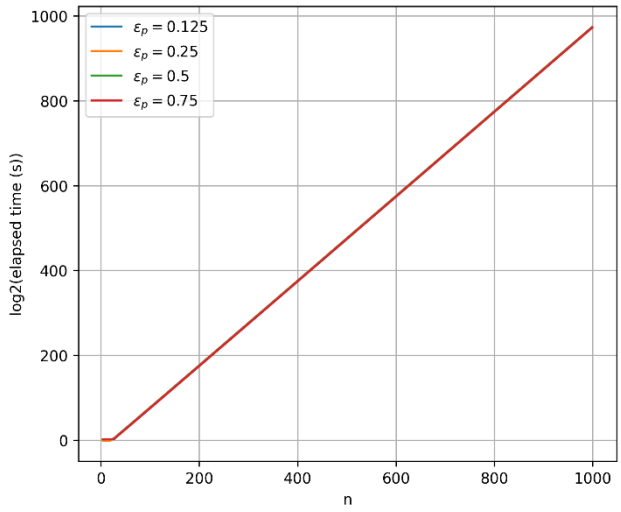


Fig. 12 – Predicted elapsed time for the Brute Force algorithm (k=0.5)

### *B. Greedy Heuristics Algorithm*

In Table VIII, we present the outcomes of a curve fitting analysis conducted using data obtained from the greedy algorithm. The model employed for this analysis is defined as:

$$f(n,k) = a \cdot n^2 + b \qquad (13)$$

TABLE VIII: Greedy Predicted Coefficients

| $k$ | $\varepsilon_p$ | $a$ | $b$ | $r^2$ |
|---|---|---|---|---|
| 0.5 | 0.125 | $1.63-7$ | $2.22e-3$ | $9.68e-1$ |
| | 0.25 | $2.57e-7$ | $1.60e-3$ | $9.84e-1$ |
| | 0.5 | $4.95e-7$ | $2.33e-4$ | $9.93e-1$ |
| | 0.75 | $7.41e-7$ | $-5.71e-4$ | $9.91e-1$ |

Fig. 13 visually illustrates the results of the curve fitting process for the greedy algorithm.

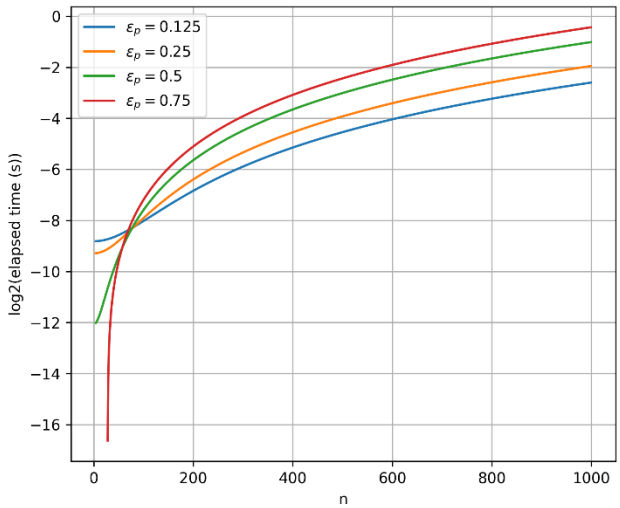


Fig. 13 – Predicted elapsed time for the Greedy algorithm (k=0.5)

## X. CONCLUSION

In addressing the independent set decision problem within graph theory, exhaustive methods have traditionally been the go-to approach. However, their computational complexity escalates exponentially, rendering them impractical for larger graphs. Despite potential gains through parallelization, even optimized strategies fall short of providing an efficient solution. With no apparent superior alternative, the reduction of computational complexity remains a primary concern. Hence, the adoption of techniques offering rapid, albeit approximate, results become imperative. This approach has demonstrated efficacy in providing reliable solutions, especially when considering the inherent challenges of the independent set decision problem. While diverse methods and varied approaches increase the likelihood of a correct solution, critical scenarios demand a delicate equilibrium between speed and the assurance of accuracy. The choice of method is contingent on specific application requirements.

## REFERENCES


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

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

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

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

[5] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to algorithms. MIT press.

[6] Hastad, J. (1996, October). Clique is hard to approximate within n/sup 1-/spl epsiv. In Proceedings of 37th Conference on Foundations of Computer Science (pp. 627-636). IEEE.

[7] Mathieu, M. A. R. I. (2017). Study of greedy algorithm for solving Maximum Independent Set problem.

[8] West, D. B. (2001). Introduction to graph theory (Vol. 2). Upper Saddle River: Prentice hall.