Independent Set Decision Problem

Daniel Jorge Bernardo Ferreira

*Resumo* - This text follows de DETUA Journal format. Its context is informative so you must remove it to use strictly the template styles. This part of the text must be in Portuguese while de rest of the article can be in Portuguese or English.

*Abstract* - Must be in English.

# 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 graphalso 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 vertices if and only if it harbors an independent set of exactly 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. Related work

# III. Preliminaries

Let stand for a simple undirected graph with a set of vertices and a set of edges. Let denote We will use to denote = . The vertex set and edge set of a graph are denoted by and respectively. For simplicity, we may denote a singleton set {} by .

For a vertex in a graph , we define the following notations. Let = denote the degree of and denote the set of vertices with distance exactly from .

We may use to represent the … The words vertex and node will be used interchangeably

# III. Algorithmics

Developed 4 algorithms

All can be modified slightly to solve the maximum independent problem. This is the correspondent decision problem after all.

I often reference the maximum independent set (I am not referring to the 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 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 vertices from the graph. The time complexity of this operation is proportional to the binomial coefficient “ choose ”. This coefficient represents the number of ways to pick elements from a set of elements and follows the formula:

|  |  |  |
| --- | --- | --- |
|  |  | (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 in the overall time complexity. Therefore, the total time complexity of the algorithm can be expressed as . 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 . 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 .

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:

|  |  |  |
| --- | --- | --- |
|  |  | (2) |

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

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

|  |  |  |
| --- | --- | --- |
|  |  | (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:

|  |  |  |
| --- | --- | --- |
|  |  | (4) |

To analyze the time complexity of this recurrence relation, we can use generating functions. Let be the generating function for the sequence . Multiplying both sides of the recurrence relation and summing over all , we get:

|  |  |  |
| --- | --- | --- |
|  |  | (5) |

Simplifying and rearranging, we arrive at:

|  |  |  |
| --- | --- | --- |
|  |  | (6) |

Now, recognizing that is simply , we can substitute to get:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | (7) |  | (6) |

Rearranging terms and factoring , we get:

|  |  |  |
| --- | --- | --- |
|  |  | (8) |

Now, we know that has roots , , , , and the solution for involves partial fractions. Solving for , we obtain:

|  |  |  |
| --- | --- | --- |
|  |  | (9) |

The roots of are complex, and the dominant root, denoted as is approximately 1.38. Therefore, the time complexity of the algorithm is , where is the number of nodes in the graph. This is significantly better than the brute force approach, as shown in Fig. 1.

A graph of different colored lines

Description automatically generated

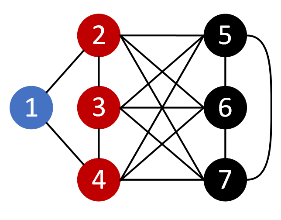
Fig. 1 − Formal Analysis of Brute Force and Branching Algorithms

## 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, , 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 time in the worst case, where 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 iterations in the worst case. Therefore, the overall time complexity of the algorithm can be expressed as .

A screenshot of a black background with numbers and circles

Description automatically generated 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 -approximation algorithm, implying that the size of the independent set it produces is, at most, times larger than the size of the optimal independent set in the graph. In practical terms, if represents the size of the maximum independent set, the theorem guarantees that the solution size, , satisfies . 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 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 , where . 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 th-degree. The th-degree of a vertex , denoted as , represents the sum of the degrees of vertices adjacent to those considered for the th degree, not including vertices already considered in a lesser degree. This is formally expressed as:

|  |  |  |
| --- | --- | --- |
|  |  | (10) |

When faced with degree ties between vertices and , the algorithm compares their th-degrees. If , the tie-breaking strategy extends to the th degree. In cases where is even, the algorithm selects the vertex with the greater th degree. Conversely, when is odd, the algorithm opts for the vertex with the lesser 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 , selecting 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. 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.

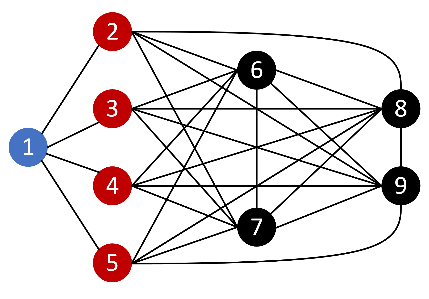


Fig. 4 − Graph where minimum degree vertex is not included in the maximum independent set

# 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.