Paper Title\* **(**use style: ***paper title***)

Subtitle as needed ***(paper subtitle)***

Adiel Rum - NIM

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung, Jalan Ganesha 10 Bandung

E-mail: [author@gmail.com](mailto:author@gmail.com) , [author@std.stei.itb.ac.id](mailto:author@std.stei.itb.ac.id)

*Abstract*—This electronic document is a “live” template and already defines the components of your paper [title, text, heads, etc.] in its style sheet. *\*CRITICAL: Do Not Use Symbols, Special Characters, or Math in Paper Title or Abstract*. (*Abstract*)

Keywords—component; formatting; style; styling; insert (key words)

# Introduction

In the fields of computer science and discrete mathematics, graph theory stands as a cornerstone for modeling and analyzing complex networks and relationships. A graph, in its simplest form, is a collection of vertices (or nodes) connected by edges, representing a vast array of real-world scenarios, from social networks and transportation systems to computer networks and molecular biology. Within this domain lies a fundamental optimization problem known as the Vertex Cover problem. A vertex cover is defined as a subset of a graph's vertices such that every edge in the graph is incident to at least one vertex within this subset. This concept is not merely a theoretical curiosity; it has profound practical applications, such as placing the minimum number of security cameras to cover all hallways in a building or identifying a minimal set of key individuals in a network to monitor for information dissemination.

The primary challenge, however, is not just finding *any* vertex cover, but finding a *minimum vertex cover*, that is, a vertex cover with the smallest possible number of vertices. Achieving this optimization is critical for efficiency and resource conservation. The difficulty of this task is formally captured by its classification as an NP-hard problem. This means that as the size of the graph grows, the time required to find the guaranteed optimal solution using any known method increases exponentially. For large, real-world graphs, finding the minimum vertex cover through exhaustive search becomes computationally infeasible, pushing the limits of even the most powerful computers.

This inherent complexity necessitates a trade-off between optimality and efficiency, leading to the development of various algorithmic strategies. This project, "Explorations of algorithms to find the vertex cover of a graph," delves into this challenge by implementing and visualizing a spectrum of these algorithms. We will explore three distinct approaches: a Brute-Force algorithm that guarantees optimality by exhaustively checking every possible subset of vertices; a Greedy algorithm that uses a simple heuristic of repeatedly selecting the vertex with the highest degree; and a 2-Approximation algorithm that provides a provable guarantee that its solution will be no more than twice the size of the true minimum. To bridge the gap between abstract theory and practical understanding, these algorithms are integrated into an interactive visualization tool, allowing for a step-by-step observation of how each method traverses the graph and constructs its solution.

Through this comparative exploration and visualization, this project aims to provide clear insights into the behavior, performance, and trade-offs associated with different approaches to solving the vertex cover problem. By observing these algorithms in action, we can better appreciate the intricate balance between computational cost and the quality of a solution, a central theme in the study of algorithm design and combinatorial optimization.

# Literature Review

## Vertex Cover

The Vertex Cover problem is a central and well-studied problem in the fields of graph theory and computational complexity. Formally, let be an undirected graph, where is the set of vertices and is the set of edges. A vertex cover of is a subset of vertices such that for every edge , at least one of its endpoints is included in the subset, i.e., . While any graph has several possible vertex covers (with the set of all vertices, V, always being a trivial one), the optimization challenge lies in finding a *minimum vertex cover*. This is a vertex cover that has the smallest possible cardinality, denoted by . The associated decision problem asks, for a given graph and an integer , whether there exists a vertex cover of size at most .

The computational difficulty of the vertex cover problem is one of its most defining characteristics. It was famously included in Richard Karp's list of 21 NP-complete problems in his seminal 1972 paper, "Reducibility Among Combinatorial Problems." The classification of a problem as NP-complete signifies that there is no known algorithm that can solve it in polynomial time for all inputs. Furthermore, if such an algorithm were ever discovered for vertex cover, it would imply that P=NP, which would resolve one of the most profound open questions in computer science. This inherent hardness makes finding the exact minimum vertex cover for large graphs computationally intractable, as any known exact algorithm, such as brute-force, has a runtime that grows exponentially with the number of vertices. Consequently, the study of vertex cover has largely focused on developing approximation algorithms and heuristics that can find near-optimal solutions in a reasonable amount of time

## Brute Force Algorithm

The most direct approach to solving the vertex cover problem is through a brute-force algorithm. This method is a straightforward, exhaustive search paradigm that systematically enumerates every possible candidate for a solution and checks whether each candidate satisfies the problem's statement. For the vertex cover problem, the candidates are all possible subsets of the graph's vertex set, . The total number of such subsets is , where is the number of vertices. A brute-force implementation would generate each of these subsets and, for each one, verify if it constitutes a valid vertex cover by checking if every edge in the graph is covered. The algorithm would keep track of the smallest valid cover found during this exhaustive process.

To find the minimum vertex cover specifically, the brute-force strategy can be refined. Instead of generating all subsets at once, the algorithm can iterate through possible cover sizes, , from 0 to . For each , it generates all vertex subsets of size (i.e., all combinations of vertices) and checks if any of them form a valid vertex cover. The first value of for which a valid cover is found will correspond to the size of the minimum vertex cover, and the corresponding subset will be an optimal solution. While this method guarantees optimality, its runtime complexity of makes it practical only for very small graphs, serving primarily as a baseline for understanding the problem's difficulty and for verifying the correctness of more sophisticated algorithms on small test cases..

## Greedy Algorithm

Given the inefficiency of brute-force methods, heuristic approaches are often employed to find good, albeit not necessarily optimal, solutions quickly. The Greedy algorithm for vertex cover is a prime example of such a heuristic. Its strategy is intuitively simple: at each step, select the vertex that covers the most uncovered edges. This means the algorithm calculates the degree (the number of incident edges) of every vertex in the current graph state and adds the vertex with the highest degree to the vertex cover. After a vertex is chosen, it and all its incident edges are removed from the graph. This process is repeated until no edges remain.

While this greedy approach is fast and often produces reasonably small vertex covers, it provides no guarantee of optimality. It is possible to construct graphs where this strategy yields a solution that is significantly larger than the minimum vertex cover. The ratio between the size of the cover found by the greedy algorithm and the size of the optimal cover can be as large as Unlike the 2-approximation algorithm, it does not have a constant approximation ratio, meaning its performance relative to the optimal solution can degrade as the graph size increases. Nevertheless, its simplicity and efficiency make it a valuable tool for obtaining a quick estimate or an initial solution.

## 2-Approximation Algorithm

In contrast to heuristics with no performance guarantees, approximation algorithms offer a provable bound on the quality of their solution relative to the optimal one. The 2-Approximation algorithm for vertex cover is a classic and elegant example. The algorithm operates on a simple iterative process: as long as there are edges remaining in the graph, it picks an arbitrary edge, say , adds both of its endpoints, and , to the vertex cover, and then removes both vertices and all their incident edges from the graph. This loop continues until all edges have been covered.

The power of this algorithm lies in its approximation ratio of 2. This means the size of the vertex cover it produces is guaranteed to be no more than twice the size of the true minimum vertex cover. This guarantee arises from a simple observation: for every edge chosen by the algorithm, at least one of its endpoints must be in any valid vertex cover, including the minimum one. Since the algorithm adds both endpoints, it adds at most twice as many vertices as would be required for an optimal cover of those same chosen edges. With a time complexity of , it provides a robust and efficient method for finding a good-quality solution with a predictable upper bound on its error.

# Algorithm Implementation

Before you begin to format your paper, first write and save the content as a separate text file. Keep your text and graphic files separate until after the text has been formatted and styled. Do not use hard tabs, and limit use of hard returns to only one return at the end of a paragraph. Do not add any kind of pagination anywhere in the paper. Do not number text heads-the template will do that for you.

Finally, complete content and organizational editing before formatting. Please take note of the following items when proofreading spelling and grammar:

## Problem Statement

The Minimum Vertex Cover problem states: Given a graph , find the smallest subset of vertices such that every edge in has at least one endpoint in . The program will take in an undirected graph . Then the program will output a list of edges, that is the vertex cover of the graph. (Jelasin vertex cover)

Generally across the algorithms, to verify if a given is a solution to the Vertex Cover problem. That is meets the criteria of a vertex cover, we use the following procedure.

|  |
| --- |
| **FUNCTION** IsVertexCover(Graph G, Subset S) |
| INPUT: A graph G with edges E, and a subset of vertices S OUTPUT: true if S is a vertex cover, false otherwise |
| FOR EACH edge (u, v) IN E:    IF (u is NOT IN S) AND (v is NOT IN S) THEN      RETURN false    END IF  END FOR  RETURN true |

The complexity of this function is where denotes the number of edges in the graph.

## Brute Force Implementation

1. Problem Mapping
   1. Solution space

For a graph with vertices, the total number of possible subsets is . All possible subset is a candidate for vertex cover.

* 1. Generating Function

The generating function generates all possible subsets of the vertices. The complexity for generating all subsets is where is the number of vertices

* 1. Validation Function

For every subset of vertices that is generated, The *IsVertexCover* function is used to verify if the given subset is a vertex cover.

1. Complexity Analysis

Since the algorithm generates possible solutions, and every solution is verified in time. The total complexity for the brute force algorithm is .

1. Implementation

The following is the pseudocode for the implementation of the brute force algorithm.

|  |
| --- |
| **FUNCTION** BruteForceVertexCover(Graph G): |
| INPUT: A graph G with vertices V and edges E OUTPUT: A minimum vertex cover V\_cover |
| FOR k FROM 0 TO size(V):    all\_subsets\_of\_size\_k = generate\_combinations(V, k)    FOR EACH subset S IN all\_subsets\_of\_size\_k:      IF IsVertexCover(G, S) THEN        RETURN S      END IF    END FOR  END FOR  RETURN an empty set |

Identify applicable sponsor/s here. If no sponsors, delete this text box (*sponsors).*

## Greedy Implementation

1. Problem mapping
   1. Greedy Heuristic

The greedy algorithm selectis the vertex with the highest degree to cover the maximum number of edges.

* 1. Solution Construction

It iteratively builds a solution by adding the highest-degree vertex to the cover, then removing that vertex and all its incident edges from consideration.

* 1. Termination and Validity

The process repeats until no edges remain, guaranteeing that the final set of chosen vertices forms a valid vertex cover for the entire graph/

1. Complexity Analysis

The algorithm's main loop continues as long as there are edges in the graph. In the worst case, this loop can run times. Within each iteration, the most computationally expensive task is to find the vertex with the highest degree. This requires calculating the degrees of all vertices by iterating through the remaining edges, which takes time. Therefore, the total time complexity for this implementation of the greedy algorithm is .

1. Implementation

The following is the pseudocode for the greedy algorithm.

|  |
| --- |
| **FUNCTION** GreedyVertexCover(Graph G): |
| INPUT: A graph G with vertices V and edges E OUTPUT: A vertex cover V\_cover |
| V\_coverV\_cover = an empty set  E\_remaining = a copy of E  WHILE E\_remaining is not empty:    let v\_max\_degree = the vertex in V with the highest degree in the subgraph formed by E\_remaining    add v\_max\_degree to V\_cover    edges\_to\_remove = an empty set    FOR EACH edge (u, v) IN E\_remaining:      IF u == v\_max\_degree OR v == v\_max\_degree THEN        add edge (u, v) to edges\_to\_remove      END IF    END FOR    E\_remaining = E\_remaining - edges\_to\_remove  END WHILE  RETURN V\_cover |

## 2-Approximation Implementation

1. Problem Mapping
   1. Core Idea

The algorithm is based on the principle that for any edge , a valid vertex cover must contain either or (or both). This algorithm conservatively includes both endpoints of a selected edge to guarantee coverage.

* 1. Solution Construction

It iteratively picks an arbitrary uncovered edge, adds both of its vertices to the cover, and then removes all edges incident to either of these two vertices.

* 1. Termination and Validity

The process repeats until no edges remain, which ensures a valid cover. This method is a 2-Approximation algorithm, meaning the size of the cover it produces is provably no more than twice the size of the optimal minimum vertex cover.

1. Complexity Analysis

The algorithm processes each edge in the graph at most once. The main loop continues as long as there are uncovered edges. In each step, at least one edge is selected and removed, along with other incident edges. With an efficient implementation (e.g., using adjacency lists), the total time complexity is linear in the size of the graph, which is .

1. Implementation

The following is the pseudocode for the implementation of the 2-Approximation algorithm.

|  |
| --- |
| **FUNCTION** TwoApproxVertexCover(Graph G): |
| INPUT: A graph G with vertices V and edges E OUTPUT: A vertex cover V\_cover |
| V\_cover = an empty set  E\_remaining = a copy of E  WHILE E\_remaining is not empty:    let (u, v) be an edge in E\_remaining    add u to V\_cover    add v to V\_cover    edges\_to\_remove = an empty set    FOR EACH edge (x, y) IN E\_remaining:      IF x == u OR x == v OR y == u OR y == v THEN        add edge (x, y) to edges\_to\_remove      END IF    END FOR    E\_remaining = E\_remaining - edges\_to\_remove  END WHILE  RETURN V\_cover |

Number equations consecutively. Equation numbers, within parentheses, are to position flush right, as in (1), using a right tab stop. To make your equations more compact, you may use the solidus ( / ), the exp function, or appropriate exponents. Italicize Roman symbols for quantities and variables, but not Greek symbols. Use a long dash rather than a hyphen for a minus sign. Punctuate equations with commas or periods when they are part of a sentence, as in

*a**b*    

Note that the equation is centered using a center tab stop. Be sure that the symbols in your equation have been defined before or immediately following the equation. Use “(1),” not “Eq. (1)” or “equation (1),” except at the beginning of a sentence: “Equation (1) is ...”

## Some Common Mistakes

* The word “data” is plural, not singular.
* The subscript for the permeability of vacuum **0, and other common scientific constants, is zero with subscript formatting, not a lowercase letter “o.”
* In American English, commas, semi-/colons, periods, question and exclamation marks are located within quotation marks only when a complete thought or name is cited, such as a title or full quotation. When quotation marks are used, instead of a bold or italic typeface, to highlight a word or phrase, punctuation should appear outside of the quotation marks. A parenthetical phrase or statement at the end of a sentence is punctuated outside of the closing parenthesis (like this). (A parenthetical sentence is punctuated within the parentheses.)
* A graph within a graph is an “inset,” not an “insert.” The word alternatively is preferred to the word “alternately” (unless you really mean something that alternates).
* Do not use the word “essentially” to mean “approximately” or “effectively.”
* In your paper title, if the words “that uses” can accurately replace the word using, capitalize the “u”; if not, keep using lower-cased.
* Be aware of the different meanings of the homophones “affect” and “effect,” “complement” and “compliment,” “discreet” and “discrete,” “principal” and “principle.”
* Do not confuse “imply” and “infer.”
* The prefix “non” is not a word; it should be joined to the word it modifies, usually without a hyphen.
* There is no period after the “et” in the Latin abbreviation “et al.”
* The abbreviation “i.e.” means “that is,” and the abbreviation “e.g.” means “for example.”

An excellent style manual for science writers is [7].

# Using the Template

After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save As command, and use the naming convention prescribed by your conference for the name of your paper. In this newly created file, highlight all of the contents and import your prepared text file. You are now ready to style your paper; use the scroll down window on the left of the MS Word Formatting toolbar.

## Authors and Affiliations

The template is designed so that author affiliations are not repeated each time for multiple authors of the same affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization). This template was designed for two affiliations.

### For author/s of only one affiliation (Heading 3): To change the default, adjust the template as follows.

#### Selection (Heading 4): Highlight all author and affiliation lines.

#### Change number of columns: Select the Columns icon from the MS Word Standard toolbar and then select “1 Column” from the selection palette.

#### Deletion: Delete the author and affiliation lines for the second affiliation.

### For author/s of more than two affiliations: To change the default, adjust the template as follows.

#### Selection: Highlight all author and affiliation lines.

#### Change number of columns: Select the “Columns” icon from the MS Word Standard toolbar and then select “1 Column” from the selection palette.

#### Highlight author and affiliation lines of affiliation 1 and copy this selection.

#### Formatting: Insert one hard return immediately after the last character of the last affiliation line. Then paste down the copy of affiliation 1. Repeat as necessary for each additional affiliation.

#### Reassign number of columns: Place your cursor to the right of the last character of the last affiliation line of an even numbered affiliation (e.g., if there are five affiliations, place your cursor at end of fourth affiliation). Drag the cursor up to highlight all of the above author and affiliation lines. Go to Column icon and select “2 Columns”. If you have an odd number of affiliations, the final affiliation will be centered on the page; all previous will be in two columns.

## Identify the Headings

Headings, or heads, are organizational devices that guide the reader through your paper. There are two types: component heads and text heads.

Component heads identify the different components of your paper and are not topically subordinate to each other. Examples include ACKNOWLEDGMENTS and REFERENCES, and for these, the correct style to use is “Heading 5.” Use “figure caption” for your Figure captions, and “table head” for your table title. Run-in heads, such as “Abstract,” will require you to apply a style (in this case, italic) in addition to the style provided by the drop down menu to differentiate the head from the text.

Text heads organize the topics on a relational, hierarchical basis. For example, the paper title is the primary text head because all subsequent material relates and elaborates on this one topic. If there are two or more sub-topics, the next level head (uppercase Roman numerals) should be used and, conversely, if there are not at least two sub-topics, then no subheads should be introduced. Styles named “Heading 1,” “Heading 2,” “Heading 3,” and “Heading 4” are prescribed.

## Figures and Tables

### Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation “Fig. 1,” even at the beginning of a sentence.

1. Table Styles

| Table Head | Table Column Head | | |
| --- | --- | --- | --- |
| Table column subhead | Subhead | Subhead |
| copy | More table copya |  |  |

1. Sample of a Table footnote. *(Table footnote)*
2. Example of a figure caption. *(figure caption)*

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity “Magnetization,” or “Magnetization, M,” not just “M.” If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write “Magnetization (A/m)” or “Magnetization (A ( m(1),” not just “A/m.” Do not label axes with a ratio of quantities and units. For example, write “Temperature (K),” not “Temperature/K.”

##### Video Link at Youtube *(Heading 5)*

Include link of your video on YouTube in this section.

##### Acknowledgment *(Heading 5)*

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g.” Avoid the stilted expression “one of us (R. B. G.) thanks ...”. Instead, try “R. B. G. thanks...”. Put sponsor acknowledgments in the unnumbered footnote on the first page.

##### References

The template will number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use “Ref. [3]” or “reference [3]” except at the beginning of a sentence: “Reference [3] was the first ...”

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors’ names; do not use “et al.”. Papers that have not been published, even if they have been submitted for publication, should be cited as “unpublished” [4]. Papers that have been accepted for publication should be cited as “in press” [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

1. G. Eason, B. Noble, and I.N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529-551, April 1955. (*references*)

We suggest that you use a text box to insert a graphic (which is ideally a 300 dpi resolution TIFF or EPS file with all fonts embedded) because this method is somewhat more stable than directly inserting a picture.

To have non-visible rules on your frame, use the MSWord “Format” pull-down menu, select Text Box > Colors and Lines to choose No Fill and No Line.

1. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68-73.
2. I.S. Jacobs and C.P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G.T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271-350.
3. K. Elissa, “Title of paper if known,” unpublished.
4. R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.
5. Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740-741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 1 Juni 2025

Ttd

Nama dan NIM