On the Computational Complexity of the Vertex Cover Problem on Cubic Bridgeless Graphs

Kunal Relia*
Independent Scientist
kunal.relia91@gmail.com / krelia@nyu.edu

Abstract

In this two-part study on the vertex cover problem on cubic bridgeless graphs ($\texttt{VC} - \texttt{CBG}$), we discover that: (I) $\texttt{VC} - \texttt{CBG}$ is **NP**-complete and (II) $\texttt{VC} - \texttt{CBG} \in \mathbf{P}$.

Part I - We prove that $\texttt{VC} - \texttt{CBG}$ is **NP**-complete by showing that (i) $\texttt{VC} - \texttt{CBG}$ is in **NP** and (ii) $\texttt{VC} - \texttt{CBG}$ is **NP**-hard via a polynomial-time reduction from a known **NP**-hard problem, namely, the vertex cover problem on cubic graphs.

Part II - In this core contribution of the paper, we discover an unconditional deterministic polynomial-time exact algorithm for $\texttt{VC} - \texttt{CBG}$. The algorithm is divided into three phases:

- Phase I finds a maximum matching of the given graph using the Blossom Algorithm [Edmonds, Canadian Journal of Mathematics 1965]. The maximum matching is always a perfect matching because we use a cubic bridgeless graph [Petersen, Acta Mathematica 1891].
- Phase II initially constructs a tree from the given graph by doing a breadth-first search of lexicographically sorted vertices. It then augments the 2-approximation algorithm of the vertex cover problem by (i) selecting edges that are in the perfect matching and (ii) ordering the selection of the edges based on the tree-level of the vertices. This augmented 2-approximation algorithm populates a novel data structure called the "represents table", which stores the information of each endpoint picked by the augmented algorithm and the neighbors of each endpoint. The endpoints form a vertex cover.
- Phase III first uses a heuristic pruning mechanism to remove some endpoints from the represents table such that the endpoints that are frozen (not removed) still form a vertex cover but with no known guarantee on its size. Then, a key concept called the "diminishing hops" is introduced, which is a local property used to guarantee a global minimality. More specifically, just like Berge's Theorem [Berge, PNAS 1957] established the relation between the absence of an augmenting path (local property) and a maximum matching (global maximum), we establish the relation between the absence of a diminishing hop (local property) and a minimum vertex cover (global minimum). Consequently, this relation forms the basis of Phase III of the algorithm, in line with Berge's Theorem forming the basis of the Blossom Algorithm.

*This work was supported in part by Julia Stoyanovich (while the author was her student at NYU and initially conceptualized the work) and by personal savings (derived from Julia's grants). The work is also a result of learning opportunities provided by Subhash Khot, Phokion G. Kolaitis, and other scientists.

Contents

1 Introduction

The $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ question [Coo71, Lev73, Kar72], directly and indirectly, is arguably one of the biggest curiosities in computer science and mathematics. Informally, the question asks if every computational problem whose solution can be *verified* in polynomial time can also be *solved* in polynomial time. Formally, the complexity class **P** consists of computational problems whose solution can be computed in time polynomial in the size of input, which is considered "efficient"[1]. Alternatively, the problems in **P** are a set of all languages that can be decided by a *deterministic* Turing Machine [Tur36, Tur38] in polynomial time. On the other hand, the class **NP** consists of computational problems that, when given a candidate solution, we can verify in time polynomial in the size of the input whether the candidate solution is correct or not. Hence, the problems in **NP** are a set of all languages that can be decided by a *non-deterministic* Turing Machine in polynomial time (or that can only be verified by a *deterministic* Turing Machine in polynomial time). The question of whether the problems in **NP** can be computed efficiently or not forms the basis of $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$.

The $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ question was formalized due to the Cook-Levin Theorem [Coo71, Lev73]. Since then, it has led to some remarkable work. An early example is by Karp who showed that twenty one computational (combinatorial) problems were, indeed, **NP**-complete [Kar72], thus formally further cementing what earlier scientists like Nash (in his 1955 letter to the National Security Agency) and Gödel (in his 1956 letter to von Neumann) already believed [Aar16]. Interestingly, eleven of the twenty-one Karp's **NP**-complete problems are *directly* graph-based problems (and at least one other problem is a graph-based problem indirectly; for example, the Job Sequencing problem may be trivially formulated on a disjunctive graph). Furthermore, among these eleven graph-based problems, one of the extensively studied is the vertex cover problem, the topic of our discussion.

1.1 Vertex Cover and its Computational Aspects

Given an unweighted undirected graph (specifically, a 2-uniform hypergraph)[2], the vertex cover of the graph is a set of vertices that includes at least one endpoint of every edge of the graph. Formally, given a graph $G = (V, E)$ consisting of a set of vertices V and a collection E of 2-element subsets of V called edges, the vertex cover of the graph G is a subset of vertices $S \subseteq V$ that includes at least one endpoint of every edge of the graph, i.e., for all $e \in E$, $e \cap S \neq \emptyset$. The corresponding computational problem of finding the minimum-size vertex cover (`MVC`) is **NP**-complete[3] (Node Cover, Problem 5 in [Kar72]). However, the hardness meant that there is no *known* unconditional deterministic polynomial-time algorithm to solve `MVC` unless $\mathbf{P} = \mathbf{NP}$. Hence, a rich line of research ensued that improved our general understanding related to the complexity of the `MVC` problem, especially around its (in)approximability and parameterized complexity.

1.1.1 Approximation Algorithm and Inapproximability

A natural relaxation to counter the hardness of any problem is to find an approximate solution that can be computed efficiently. For the `MVC`, a trivial 2-approximation algorithm computes a vertex

[1] The first association between efficiency, polynomial-time computability, and the complexity class **P** may be attributed to the Cobham-Edmonds thesis [Cob65, Edm65].

[2] For the remainder of the paper, a graph refers to an unweighted undirected finite graph. Furthermore, without loss of generality (w.l.o.g.), we assume the graph is simple (no loops and no multiple edges) and connected (Appendix A).

[3] Strictly speaking, the decision version (`VC`) of the minimum-size vertex cover problem is **NP**-complete whereas the `MVC` itself (search version) is **NP**-hard. See Section 2.1 of [Kho19] for a lucid explanation delineating (a) search and decision problems and (b) NP-hardness and NP-completeness. Until formalized, we use `MVC` and `VC` interchangeably.

cover of size at most twice the minimum size vertex cover in polynomial time. The algorithm picks an arbitrary edge $e = (u, v) \in E$, adds both the vertices u and v to the vertex cover S, removes all edges connected to either of the two vertices (u and v), and repeats until no edge remains. Additionally, complex techniques like linear programming-based algorithms also obtain an approximation ratio of 2 [ABLT06]. However, it is not known whether an approximation algorithm with a strictly better approximation ratio exists or not. This is among the major open problems within the Theoretical Computer Science (TCS) community. A path towards understanding this was explored by Håstad who, following the PCP theorem [FGL+96, AS98, ALM+98, Din07], used a 3-bit PCP to show that it is **NP**-hard to approximate `MVC` within a factor of $1.1667 - \varepsilon$ [Hås01][4]. Dinur and Safra went beyond this factor to $1.3606 - \varepsilon$ [DS05]. Khot, Minzer, and Safra further improved the bound to $1.4142 - \varepsilon$ (as an implication of proving the 2-to-2 Games Conjecture) [KMS23]. Independently, assuming Khot's Unique Games Conjecture (UGC) [Kho02], we know that the `MVC` problem may be hard to approximate within a factor of $2 - \varepsilon$ [KR08], thus matching the bound of the known trivial 2-approximation algorithm. Khot and Regev actually gave a generalization of this result: assuming the UGC, the `MVC` on k-uniform hypergraphs may be hard to approximate within $k - \varepsilon$ for all integers $k \geq 2$. Without assuming the UGC, the `MVC` on k-uniform hypergraphs is hard to approximate within $k - 1 - \varepsilon$ for all integers $k \geq 3$ [DGKR03].

1.1.2 Parameterized Complexity

Another relaxation to overcome the worst-case intractability of the `MVC` is to assume the size of certain parameters. For instance, if we assume that the size of the vertex cover is small, then there are known algorithms that run in time polynomial in the size of the input (i.e., number of edges and vertices). However, all such results are conditioned on the assumption of the size of one or more parameters, including a new parameter called bridge-depth [BJS22]. Hence, given our aim to provide *unconditional* results, we refer the readers to a comprehensive discussion on the parameterized complexity [DF12] and, in particular, on the fixed-parameter tractability of the `MVC` [DF95] and on the parameterized complexity of its variants [GNW07].

A common denominator across the above-discussed computational aspects of the `MVC` is the missing (tangible and visible) effort to discover an unconditional deterministic polynomial-time (exact) algorithm. To the best of our knowledge, no recorded work aims to either (i) significantly improve the trivial 2-approximation algorithm unconditionally and deterministically or (ii) tame the exponential component of a parameterized algorithm. Additionally, there are no advances in efficiently solving the `MVC` via parallel computing or under quantum complexity (else the relationship between complexity classes **BQP** (or **EQP**) and **NP** would be known).

1.1.3 Tractability under Restricted Graphs

The `MVC` becomes tractable under various restricted scenarios. For example, the `MVC` is in **P** when the graph is restricted to (i) a tree, (ii) bipartite (Kőnig's theorem [Kon31]), or (iii) claw-free (because the maximum independent set problem on claw-free graphs is in **P** [Sbi80, Min80]). However, no such study aims to discover an unconditional deterministic polynomial-time (exact) algorithm for an **NP**-complete variant of the `MVC`.

In summary, there is neither a study to *significantly* improve the 2-approximation algorithm for the `MVC` nor a study on an algorithm for a restricted setting of the `MVC` that is **NP**-complete. Consequently, we shift our discussion to understanding how the research on resolving the $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$

[4] ε denotes an arbitrarily small constant such that $\varepsilon > 0$ and the results are meant to hold for every such ε.

question relates to the MVC. This is important especially because all **NP**-complete problems are "equivalent" in a certain technical sense. In particular, we assess if existing research either prohibits or limits the discovery of an algorithm for an **NP**-complete variant of the MVC.

1.2 $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$, Lower Bounds, Barriers, and the MVC

The $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ question has arguably attracted unparalleled research in the number of approaches to solving it. On the surface, there are three possible outcomes: (i) $\mathbf{P} = \mathbf{NP}$, (ii) $\mathbf{P} \neq \mathbf{NP}$, or (iii) $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ is unsolvable or undecidable. On zooming in, each outcome, especially the first two, is associated with multiple approaches. The third outcome has not received particular attention.

To prove $\mathbf{P} = \mathbf{NP}$ using a constructive proof, we can either (a) discover a polynomial-time algorithm for an **NP**-complete problem or (b) prove that a problem in **P** is **NP**-complete. While the technique used for both approaches may be the same, the problem space being targeted is different. A constructive proof for $\mathbf{P} = \mathbf{NP}$, especially an algorithm with a lower order polynomial time complexity, would fundamentally reshape how we study complexity theory, as not only will all "hard" problems be in **P**, but there will be an algorithm to solve them all! A non-constructive proof for $\mathbf{P} = \mathbf{NP}$ may not have similar major practical consequences. On the other hand, the aim to prove $\mathbf{P} \neq \mathbf{NP}$, which is widely believed to be the case and would imply that many of the problems in **NP** cannot be computed efficiently, has led to multiple important breakthroughs in how and more importantly, how not to approach a proof for $\mathbf{P} \neq \mathbf{NP}$.

1.2.1 Arguments Against a Proof of $\mathbf{P} \neq \mathbf{NP}$

There is an amazing breadth and depth of research focused towards proving $\mathbf{P} \neq \mathbf{NP}$. Yet, each major approach, ranging from the earlier logic-based techniques to the most recent Geometric Complexity Theory, has either hit at least one of the barriers in complexity theory or been stagnated. We discuss some of these approaches as arguments against a proof of $\mathbf{P} \neq \mathbf{NP}$ and then provide an overview of its relevance to the MVC.

<u>Approaches that Hit a Barrier</u> We enlist approaches that were negated by one of the three barriers in complexity theory, namely, relativization [BGS75], natural proofs [RR94], or algebrization [AW09].

- **Logical Techniques:** Initial research in TCS that aimed at separating the classes **P** and **NP** used techniques borrowed from logic and computability theory, especially previously successful techniques that were used for separation results. One such strong candidate was the diagnolization technique. However, Baker, Gill, and Solovay [BGS75] showed that these techniques that relativize cannot be used to solve the $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ question. This is because a relativizing proof for $\mathbf{P} \neq \mathbf{NP}$ would mean that there exists another "relativized world" where **P** and **NP** Turing machines can compute a problem in polynomial time, even in a single time stamp. Hence, there are relativized worlds where $\mathbf{P} = \mathbf{NP}$, and other relativized worlds where $\mathbf{P} \neq \mathbf{NP}$. Therefore, any solution to the $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ problem will require non-relativizing techniques.
- **Proof Complexity and Circuit Lower Bounds:** The need for a non-relativizing approach led the researchers to turn to proof complexity and circuit complexity. The proof complexity approach would lead to counterintuitive results such that, with some additional work, one could prove $\mathbf{P} \neq \mathbf{NP}$, and even $\mathbf{NP} \neq \mathbf{coNP}$. This is because the resolution technique

(and its enhancements) in proof complexity discuss exponential lower bounds on the sizes of unsatisfiability proofs but not for arbitrary proof systems. Consequently, if one could prove super-polynomial lower bounds for arbitrary proof systems, the above-mentioned counterintuitive result would hold. Hence, researchers shifted the focus to circuit lower bounds.

One of the exciting circuit lower bound approaches was the monotone circuit lower bounds program due to an exponential lower bound for the clique problem[5] by a then-graduate student Razborov [Raz85a]. However, this hit a wall when an exponential lower bound for the matching problem[6] was discovered by Razborov [Raz85b]. Thus, a discussion on monotone circuit lower bounds was actually a discussion on the weakness of monotone circuits and not on the "hardness" of **NP**-complete problems.

Despite such limitations, the circuit lower bound program continued to be promising. It used a novel but intuitive approach where, in addition to restricting the number of gates (as done with the monotone circuits), the "depth" of the circuits was restricted, i.e., the number of layers of gates between input and output was restricted. Hence, such small-depth circuits, coupled with combinatorial techniques like the polynomial method and random restriction, were examined. However, this entire approach hit a new barrier, namely, the natural proofs barrier [RR94]. Specifically, a natural proof would show that the very problems that were proven hard had an efficient algorithm.

- **Arithmetization (+ Logic):** Given the existence of the relativization and natural proofs barriers, researchers turned their attention to an approach called arithmetization.

  Specifically, we know that diagonalization relativizes but circumvents natural proofs. On the other hand, techniques using circuit complexity hit the natural proof barrier. Hence, there was a need to circumvent both of these barriers. This was the reason for the use of arithmetization, a technique that promoted the basic logical gates to polynomials and arithmetic operations. Thus, the technique (i) enabled the use of properties like error-correcting that were not usable for the Boolean case and (ii) also did not relativize. Hence, the mixture of the non-relativizing arithmetization with non-naturalizing diagonalization seemed to be a good approach. However, Aaronson and Wigderson [AW09] showed the existence of a new barrier: algebraic relativization (algebrization). This barrier depicted that *all* known arithmetization-based results that do not relativize, algebrize! Simultaneously, they showed that it is imperative for a technique to *not* algebrize for it to solve a host of basic complexity-related problems (see Section 1.2, second set of bullet points in [AW09] for a list), which meant that a solution to the $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ question also needs to be non-algebrizing.

The three barriers in complexity theory – relativization, natural proofs, and algebrization – have shown that approaches based on diagonalization (and other logic methods), circuit lower bounds, and arithmetization cannot be used to prove that complexity classes **P** and **NP** are distinct. Hence, the very existence of these barriers against a proof for a separation result is a strong argument against $\mathbf{P} \neq \mathbf{NP}$. Moreover, an unconditional deterministic polynomial-time algorithm for an **NP**-complete problem is not affected by these barriers, and a correct proof will hold without violating or contradicting any existing theory! This acts as an argument in favor of $\mathbf{P} = \mathbf{NP}$.

[5]Given a graph G, a clique is a subset of vertices $W \subseteq V$ such that all vertices in the subset are adjacent to each other (i.e., the subset forms a complete graph). The corresponding computational problem of finding the maximum size cliques in a graph is the clique problem. The clique problem is **NP**-complete.

[6]Given a graph G, matching M is a subset of edges such that no vertex is incident to more than one edge (Definition 7). The corresponding computational problem of finding the maximum size matching is the matching problem. The matching problem is in **P**.

On the flip side, these barriers also suggest that proving separation between the classes will require significantly different approaches. A few approaches that circumvent each of these barriers have been explored but are, to the best of our knowledge, currently stagnated:

"Stagnated" Approaches We now enlist approaches that have made progress but are stagnated.

- **Ironic Complexity Theory (ICT)**[7]**:** The term "ironic" in the name is apt - the ICT program aims to assess whether an efficient algorithm for one problem can be used to show that an efficient algorithm cannot exist for another problem. Conversely, is it possible that proving the non-existence of an efficient algorithm for one problem implies that an efficient algorithm solves another problem? At a high level, the ICT program aims to discover algorithms to prove lower bounds. However, theoretically, such surprising results depend on collapse(s) in the Time Hierarchy Theorem. Previous examples of positive results involve understanding, say time-space complexity tradeoffs [LV99], to discover surprising algorithms and collapses that do occur to establish new lower bounds! While examples of such amazing results are there, especially by Williams (e.g., [Wil14]), the common denominator across all approaches is that it will still require years of work.

- **Arithmetic Complexity Theory (ACT):** The ACT program, a generalization of the traditional Turing Machine and Boolean circuits using Boolean values, uses arithmetic circuits, which consider computer programs that use some larger field of values, such as real or complex numbers instead of Boolean. Then, the task here is to find the minimum number of operations needed to compute some polynomial over the chosen field of values. To that end, the arithmetic complexity world analog of the $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ question is the permanent versus determinant question for an $n \times n$ matrix[8]. It is known that the determinant is computable in polynomial time but the permanent is #**P**-complete [Val79b]. This led to a remarkable line of research on the study of the lower bounds for arithmetic circuits concerning this question. However, all approaches fell short of resolving the question, mainly the Valiant Conjecture. The reasons include the absence of a technique that works for permanent but fails for determinant and a technique that circumvents the arithmetic variant of the natural proof barrier, if there is one.

- **Geometric Complexity Theory (GCT):** The GCT program was a once-promising approach started by Mulmuley [Mul99] and forwarded along with Sohoni [MS01, MS08] and others[9]. At a high level, it aims to resolve the $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ question via a resolution of Valiant's algebraic analog, the **VP** vs **VNP** conjecture [Val79a]. Importantly, GCT had the potential, in part, because it overcame the three barriers. However, Panova recently discussed that the study of Kronecker and plethysm coefficients has effectively stagnated the progress of the GCT program [IP17, BIP19, DIP20]. In particular, for the GCT to progress, asymptotic representation theoretic multiplicities need to be studied, which can then be used to understand the computational complexity lower bounds [Pan23]. In summary, the GCT program, as of 2025 and except for the *general* approach of resolving the permanent versus determinant question (borrowed from previous approaches), has almost stagnated and is not a strong contender to separate the complexity classes **P** and **NP** in the near future.

[7] We borrow the use of the term ironic complexity theory from Aaronson's overview of the topic in Section 6.4 of [Aar16] where he primarily discusses the work of Williams.

[8] The definitions of determinant and the permanent are the standard definitions used for any square $n \times n$ matrix.

[9] We refer the reader to [Mul11] for a formal overview of GCT and to [Mul12] for an informal one.

The progress on the three promising approaches mentioned above has either stagnated or is a long shot from a solution. Here, we include a discussion because they overcome the three barriers in complexity theory and directly relate to our paper's overall topic.

Finally, we acknowledge that none of the six arguments presented above rule out the possibility of a new approach to proving $\mathbf{P} \neq \mathbf{NP}$[10]. However, we stress that a constructive proof for $\mathbf{P} = \mathbf{NP}$ would support the present theory – specifically, explain the presence of barriers, the absence of exponential lower bounds, and the lack of significant progress despite efforts in proving $\mathbf{P} \neq \mathbf{NP}$.

1.2.2 Relevance to the `MVC`

The research on proving $\mathbf{P} \neq \mathbf{NP}$ is connected to the `MVC` in two ways: (i) barriers in complexity theory do not prevent an algorithm for the `MVC` and (ii) given that all **NP**-complete problems are "equivalent" in a certain technical sense, an exponential (or more generally, a super-polynomial) lower bound for one of the them (or for any analogous lower bound programs) would imply that the `MVC` cannot be solved efficiently. However, no such lower bounds are known. In the other direction, our proof of $\mathbf{P} = \mathbf{NP}$ wouldn't be a total disaster for lower bounds research, too. This is because our result would adhere to the known theories as it will provide a polynomial upper bound to the lower bounds instead of the current exponential upper bound. Also, while there are such indirect implications, none of the studies, to the best of our knowledge, provide insight to *directly* improve our understanding of the vertex cover problem[11].

In summary, given the state of research, we do not have a technical reason ("barrier") that would prohibit ("lower bound") us from having a polynomial-time algorithm to solve an **NP**-complete problem, namely the `MVC`, let alone the $\texttt{VC}-\texttt{CBG}$. On the contrary, an algorithm for the $\texttt{VC}-\texttt{CBG}$ would validate our arguments and explain many current theories in computational complexity literature! Hence, we now shift our discussion to some non-computational aspects of graph theory with a focus on the research relevant to vertex covers and, specifically, on research that facilitates the discovery of an algorithm for an **NP**-complete variant of the `MVC` in Part II of the paper.

[10] We kindly refer the reader to [Coo03, Wig06, Aar16] for a detailed discussion on the importance and progress on solving $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ question and to [Wig09, For09, Var10, For21] for a relatively non-technical discussion.

[11] We stress that we are aware of some of the results that shed light on some of the **NP**-complete problems. For example, Williams showed that any algorithm for the `MVC` and other related problems like SAT and independent set need at least $n^{2\cos(\pi/7)} \approx n^{1.8019}$ time to be solved if they use $n^{o(1)}$ space [BW15, Wil19]. Recently, it was shown that any algorithm using $n^{o(1)}$ space cannot be solved in $n^2/\log_c(n)$ time for some constant $c > 0$ [Wil25]. However, these results do not improve our understanding of the vertex cover problem per se, especially its graphical properties. Interestingly, the above-stated result can be a meta argument within the ICT argument because the stagnancy of the $2\cos(\pi/7)$ bound shows that no known technique can improve this number and hence, we have a long way to go before proving $\mathbf{L} \neq \mathbf{NP}$, let alone $\mathbf{PSPACE} \neq \mathbf{P}$ or $\mathbf{P} \neq \mathbf{NP}$.

Nonetheless, we note that the algorithm in Part II adheres to these time-space bounds even when the algorithm is for the restricted case where the graphs are cubic bridgeless graphs. Additionally, we suspect the time complexity for the `MVC` on graphs may be a higher-order polynomial or have a huge constant. For instance, a preliminary analysis shows that if the time complexity of an algorithm to solve the $\texttt{VC}-\texttt{CBG}$ is $\mathcal{O}(poly(m,n))$ where $poly(m,n)$ is some (high order) polynomial in the number of vertices (m) and the number of edges (n), say $m^5 \cdot n^4$, then the time complexity of the algorithm to solve `MVC` on (i) 4-regular graphs would be $\mathcal{O}(68,719,476,736 \cdot poly(m,n))$, (ii) 5-regular graphs would be $\mathcal{O}(322,687,697,779 \cdot poly(m,n))$, (iii) 6-regular graphs would be $\mathcal{O}(3.108710029642957 \cdot 10^{16} \cdot poly(m,n))$, and so on. However, we leave this analysis for future work.

1.3 Some Non-Computational Aspects of Graph Theory

Since Euler (formally) introduced graphs to solve the Königsberg Bridge problem[12] in 1736 [Eul36], the field of Graph Theory has evolved and found applications in various areas, including computer science, medicine, and social science. Here, we discuss aspects of graph theory relevant to this paper.

1.3.1 Matching Theory

A matching M of a graph G is a set of edges such that no two edges in M share a common vertex. Three variants of matching have been studied extensively. (i) *Maximal Matching:* A matching M is maximal if every edge in graph G has a non-empty intersection with at least one edge in matching M. (ii) *Maximum Matching:* A matching M is maximum if M is maximal and the size of M is the largest possible for the given graph. (iii) *Perfect Matching:* A matching M is perfect if every vertex v of the graph G is incident to an edge of the matching.

Existence of a Matching Each graph has at least one maximal matching and one maximum matching (by definition). However, no such trivial guarantee is known for the existence of a perfect matching in a given graph, except for the trivial observation that no graph with an odd number of vertices has a perfect matching. Hence, the existence of a perfect matching in a given graph has been studied extensively. Here, we discuss a few relevant seminal papers on the existence of a perfect matching.

One of the first papers on perfect matching was by Petersen, who, in 1891, showed that every cubic (also called 3-regular or trivalent) bridgeless (also called isthmus-free or having no cutedge or 2-edge-connected) graph has at least one perfect matching (also called a 1-factor) [Pet91]. A relaxation to the bridgeless condition is known where every cubic graph with at most 2 bridges has at least one perfect matching. Next, Hall gave a characterization of the existence of a perfect matching in bipartite graphs (Hall's Marriage Theorem) [Hal35]. A generalization of these results is due to Tutte who characterized arbitrary graphs that do not have a perfect matching [Tut47]. This is further generalized by the Tutte-Berge formula to include infinite graphs.

Finding a Matching The existence of maximum matching in every graph and the research on the existence of perfect matching in a given graph resulted in two research foci: (i) counting the number of matchings in a graph and (ii) attempts to find a matching, especially efficiently. For instance, for cubic bridgeless graphs, it was conjectured [LP09] (Conjecture 8.1.8) that there are exponentially many perfect matchings in every cubic bridgeless graph. This was proven [EKK+11] through a series of incremental results that (a) gradually improved the lower bound for the general case [EPL82, KSS09, EKŠŠ10, EKK12] and (b) proved the exponential bound for special graphs [Voo79, Oum11, CS12]. Next, we discuss the research on *finding* a matching. In particular, in the context of this paper, we discuss Berge's Theorem [Ber57], which is at the heart of the Blossom Algorithm [Edm65] that finds a maximum matching in a graph.

Berge's Theorem, stated in 1957, relies on the concepts of alternating paths and augmenting paths in a graph G with respect to (w.r.t.) a given matching M. An alternating path in a graph is a path (i) that starts from a vertex v that is not incident to any edge in M and (ii) whose edges alternate between not being in M and being in M (or being in M and not being in M). An augmenting path is an alternating path starting and ending on two distinct vertices that are not

[12] The Königsberg Bridge problem was to determine whether it was possible to walk through the city of Königsberg (now Kaliningrad, Russia), crossing each of its seven bridges exactly once. Euler proved it is impossible to do so.

incident to any edge in M. An augmenting path consists of an odd number of edges because the number of edges in an augmenting path that is not in M is one more than the number of edges in M. Using these concepts, Berge proved that given a matching M, M is a maximum matching if and only if there is no augmenting path in the graph G w.r.t. matching M. Consequently, given any matching M, one can find a maximum matching by using augmenting paths [Edm65].

Overall, we discussed some non-computational aspects of matching theory. We focused on understanding the existence of perfect matchings (particularly, Petersen's Theorem) and on theories to count and find matchings (particularly, Berge's Theorem towards finding the maximum matching).

1.3.2 Graph Theory and Vertex Cover

We now assess the relation between the aforementioned topics in graph theory and vertex cover. Specifically, we understand the relation between the maximum matching and the minimum vertex cover and explore our known understanding of the vertex cover on cubic bridgeless graphs.

<u>Matching and Vertex Cover</u> Matching of a graph and the vertex cover of a graph are closely related. Just like maximum matching, a minimum vertex cover always exists (by definition). Additionally, given an arbitrary graph, the size of the minimum vertex cover is at least the size of the maximum matching. In case of the existence of a perfect matching, the size of the minimum vertex cover is at least $\frac{m}{2}$. If the graph is bipartite, then the size of the maximum matching is equal to the size of the minimum vertex cover (Konig's Theorem [Kon31]). However, despite such numerical relations between maximum matching and minimum vertex cover, there is no known structural relation between the two[13]. Moreover, like Berge's Theorem is to maximum matching, there is no such analog to the minimum vertex cover.

<u>Cubic Bridgeless Graphs and Vertex Cover</u> Regular graphs have been extensively studied from computational (e.g., [AKS11, Fei03]) and non-computational (e.g., see page 585 of [Wes01] for a list of mentions of the words regular, 3-regular, and k-regular) perspectives. More specifically, for the non-computational aspects, regular graphs and particularly 3-regular graphs have been well-studied in the matching theory. Importantly, starting with Petersen's paper [Pet91], cubic *bridgeless* graphs have been a focus. However, no such study of vertex cover on cubic bridgeless graphs exists. Consequently, no computational papers focus on the `MVC` on cubic bridgeless graphs.

In summary, we focused our discussion on the existence of matchings in graphs (especially perfect matching in a cubic bridgeless graph) and the theories that help us count the number of matchings and the (non-computational, graph-theoretic) results that are used to find a matching (especially Berge's Theorem for finding a maximum matching)[14]. Next, in the context of vertex cover, we observed that the otherwise well studied cubic bridgeless graphs are not studied for the vertex cover. Additionally, there is no analog of Berge's Theorem for the minimum vertex cover problem. Hence, in this paper, we focus on understanding the non-computational aspects of the minimum vertex cover in cubic bridgeless graphs. In turn, we study the complexity of the corresponding computational problem of finding the minimum vertex cover in cubic bridgeless graphs.

[13] By structural relation, we mean the possibility of some relation between the pairs of endpoints of the edges in perfect matching (or maximum matching) and the vertices in minimum vertex cover.

[14] For a curious reader, we refer them to some textbooks that discuss the amazing work done in graph theory ([BLW86, Gib85, BM08, Wes01, LP09]).

1.4 Section Summary

We summarize the key observations:

- **Vertex Cover:** There is an extremely exciting line of work to understand the computational complexity of the minimum vertex cover problem (`MVC`), especially around its (in)approximability. However, no work aims to either find an algorithm for an **NP**-complete variant of the `MVC` or aims to *significantly* reduce the factor 2 approximation successfully. (Also, given that we dive deep into the differences between the variants of the `MVC`, we omitted the discussion on the research progress of every other **NP**-complete problem as it is beyond the scope of this paper, even when all **NP**-complete problems are "equivalent" in a certain technical sense[15].)

- **$\mathbf{P} \stackrel{?}{=} \mathbf{NP}$:** The rich breadth and depth of research surrounding an answer for the $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ question has resulted in (i) three barriers in complexity theory that do not allow easy separation of complexity classes **P** and **NP**. Rather, a (constructive) proof for $\mathbf{P} = \mathbf{NP}$ may explain the existence of these barriers! (ii) The programs that explore the lower bounds and that also overcome the barriers do not prohibit a polynomial-time for the `MVC`, let alone for the restricted case of the `MVC` on cubic bridgeless graphs.

- **Matching and Cubic Bridgeless Graphs**: We studied amazing non-computational aspects of graph theory. Specifically, we learned that: (i) Every cubic bridgeless graph has a perfect matching. (ii) Berge's Theorem proves that a matching is maximum if and only if there is no augmenting path w.r.t. the given matching. There is no analog of Berge's Theorem for the `MVC`. (iii) The vertex cover problem on cubic bridgeless graphs has not been studied.

These observations are essential for our paper, especially for the algorithm in Part II. We particularly stress the importance of the following three results: Petersen's Theorem in [Pet91], Berge's Theorem in [Ber57], and the Blossom Algorithm [Edm65]. We also assume a basic familiarity with standard algorithmic techniques. Finally, we make the following contribution:

Contribution: In this study on the vertex cover problem on cubic bridgeless graphs ($\texttt{VC}-\texttt{CBG}$), we prove that (i) $\texttt{VC}-\texttt{CBG}$ is **NP**-complete (Theorem 1) and (ii) $\texttt{VC}-\texttt{CBG} \in \mathbf{P}$ (Theorem 2).

More specifically, we work on an understudied problem, the vertex cover problem on cubic bridgeless graphs ($\texttt{VC}-\texttt{CBG}$). In Part I, we show that $\texttt{VC}-\texttt{CBG}$ is **NP**-complete by reducing from a known **NP**-complete problem, namely, the vertex cover problem on cubic graphs [GJS74, GJ02]. Next, in Part II, we present the core contribution of the paper: an unconditional deterministic polynomial-time algorithm for the $\texttt{VC}-\texttt{CBG}$, which is spread over three phases (Figure 1). Phase I leverages the knowledge of the existence of a perfect matching in every cubic bridgeless graph [Pet91] to find a perfect matching for the given graph using the Blossom algorithm [Edm65]. Phase II uses the perfect matching and a breadth-first search tree to create an augmented version of the (vanilla) 2-approximation algorithm for the vertex cover problem. This augmented algorithm populates a novel data structure called the "represents table", which stores the information of each endpoint picked by the augmented algorithm and the neighbors of each endpoint. Phase III introduces a novel technique called the "diminishing hops". The use of a diminishing hop to find a minimum vertex cover is analogous to the use of an augmenting path to find a maximum matching [Ber57]. The

[15] Given the massive problem space of **NP**-complete problems, we acknowledge the possibility that our work may bear similarities to, say, some random, seemingly unrelated **NP**-complete problem's approximation algorithm or its hardness of approximation or an algorithm for its restricted case. However, to the best of our knowledge, no such known similarity exists.

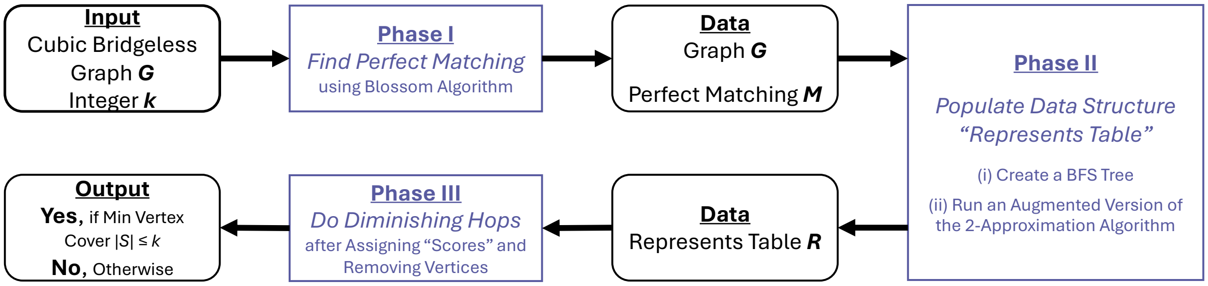

Figure 1: A schematic representation of the three phases of the algorithm, with the corresponding input and output data for each phase clearly indicated.

amalgamation of these three phases results in an algorithm for the $\texttt{VC}-\texttt{CBG}$. As mentioned earlier, our work conforms to the existing theories in computational complexity literature and provides a rationale for the existence of the known barriers in complexity theory. Additionally, our work provides a constructive algorithm for $\texttt{VC}-\texttt{CBG}$ by focusing on improving the understanding of the graph theory-related aspects of $\texttt{VC}-\texttt{CBG}$.

Organization: In Section 2, we formalize the statements of the two main theorems, and provide a high-level overview of the corresponding proofs. In Section 3, we fix the notation used and define the computational problems being worked on. In Part I, we prove $\texttt{VC}-\texttt{CBG}$ is **NP**-complete by showing that $\texttt{VC}-\texttt{CBG} \in \mathbf{NP}$ and subsequently showing that $\texttt{VC}-\texttt{CBG}$ is **NP**-hard by providing a polynomial-time reduction from a known **NP**-hard problem. In Part II, we present an unconditional deterministic polynomial-time algorithm for the $\texttt{VC}-\texttt{CBG}$, which implies $\texttt{VC}-\texttt{CBG} \in \mathbf{P}$. Each part begins with a brief introduction and an overview of its sections.

2 High-level Overview of Results

This paper is divided into two main results:

2.1 Part I: NP-completeness

In this part, we prove the following theorem:

Theorem 1. *The vertex cover problem on cubic bridgeless graphs* ($\texttt{VC}-\texttt{CBG}$) *is* **NP***-complete.*

We prove the **NP**-completeness of $\texttt{VC}-\texttt{CBG}$ in two steps: (i) we show that $\texttt{VC}-\texttt{CBG} \in \mathbf{NP}$ and (ii) we show that $\texttt{VC}-\texttt{CBG}$ is **NP**-hard via a polynomial-time reduction from the vertex cover problem on cubic graphs, a known **NP**-hard problem. For the latter, the key idea in the reduction is the replacement of each edge, whether a bridge or not, with two blocks of dummy vertices. Specifically, we split the edge to insert one block of dummy vertices and we split each endpoint of the edge to insert another block of dummy vertices. We call this an "atomic operation". The constructed gadget with the inclusion of dummy vertices (and corresponding edges that are part of at least one cycle by design) ensures that the edge from the given graph is also part of at least one cycle, which in turn implies, it is not a bridge. Each atomic operation results in one edge of the given graph guaranteed to be transformed into a bridgeless edge. Hence, we repeat an atomic operation for each edge in the graph. This ensures that the graph becomes cubic bridgeless. The reduction is polynomial in the size of input. The proof of correctness ensues because of the mapping between the edge and an atomic operation. We defer a more detailed discussion of the reduction and the proof to Section 4, which uses visualizations for easier understanding.

2.2 Part II: Algorithm

In this part, we prove the following main theorem:

Theorem 2. *The vertex cover problem on cubic bridgeless graphs ($\texttt{VC} - \texttt{CBG}$) is in* $\mathbf{P}$.

We prove this theorem through discovery of an unconditional deterministic polynomial-time exact algorithm for $\texttt{VC} - \texttt{CBG}$, which is a sequential implementation of three phases (Figure 1):

2.2.1 Phase I: Find a Perfect Matching

Input: a cubic bridgeless graph G

Execution: Phase I of the algorithm uses the Blossom Algorithm [Edm65] to find a maximum matching of the given graph. However, Petersen's Theorem [Pet91] states that every cubic bridgeless graph has a perfect matching. Hence, given we use a cubic bridgeless graph as the input, the maximum matching found by the Blossom Algorithm is indeed a perfect matching. An advantage of using the Blossom Algorithm is that it inherently takes care of the messy odd cycles.

Output: a cubic bridgeless graph G, a perfect matching M

2.2.2 Phase II: Populate Represents Table

Input: a cubic bridgeless graph G, a perfect matching M

Execution: Phase II first constructs a tree from graph G by doing a breadth-first search of its lexicographically sorted vertices. Then, it uses the tree and the perfect matching to augment the 2-approximation algorithm for the vertex cover problem. More specifically, instead of randomly choosing the edges during each iteration of the the 2-approximation algorithm, Phase II selects an edge (i) that is in perfect matching M and (ii) based on the order determined by the level on which an endpoint of the edge is in the BFS tree. The output of this augmented 2-approximation algorithm is a novel ordered unique data structure called the "Represents Table".

More specifically, for each edge picked by the augmented 2-approximation algorithm, the represents table R stores the information of the endpoints of the edge picked and of the vertices adjacent to each endpoint during a given iteration. The idea behind the name "represents table" is that each endpoint of the picked edge "represents" the vertices adjacent to it (think: in an election, a "picked" candidate represents its voters). This simple data structure leads to some unique properties. For instance, for all non-negative integers i, j where $i < j$, an endpoint in row j of the represents table R cannot represent an endpoint in row i (Property 1). Additionally, the endpoints in the table form a vertex cover, and all the table operations such as that of removal of an endpoint are designed such that the endpoints that are not removed always form a vertex cover (Property 3, Property 4). Such properties are inherently important during the last phase of the algorithm. We defer the formalization and discussion of properties and operations to Section 5.2.

Output: a represents table R such that its endpoints form a vertex cover

2.2.3 Phase III: Diminishing Hops

Input: a represents table R such that its endpoints form a vertex cover

Execution: Phase III initially uses "representation score" (heuristic pruning) to remove some endpoints from the represents table R such that the frozen (unremoved) endpoints continue to form a vertex cover. At a high level, the idea is to associate a score with each endpoint in the table R such that the score is used to decide whether to freeze (or remove) an endpoint or not. The assignment of the score begins from the top row of the table R. The score assigned to an endpoint u is the sum of the scores of the endpoint that is on the same row as each endpoint that represents u (but is not in the same row as u). For instance, consider that the endpoint u is in the i^{th} row for some integer $i > 1$. Next, if some endpoint x in some row $[1, i-1]$ represents the endpoint u, then the score of endpoint y that is on the same row as endpoint x is added to the score of endpoint u plus one. The higher the score of endpoint y, the higher the chance of endpoint u being frozen so that endpoint x can in turn be removed. Note that both the endpoints in the first row have a score of zero each and the computation moves downward. At the end of the score assignment, and freezing and removal of endpoints, the frozen endpoints always form a vertex cover (Theorem 4).

Phase III then introduces the concept of diminishing hops, which is inspired by Berge's Theorem [Ber57]. More specifically, the Blossom algorithm for maximum matching relies on Berge's Theorem: given a graph G and a matching M, M is a maximum matching if and only if there is no M-augmenting path in graph G. We "reimagine" the concept of augmenting paths for maximum matching as diminishing hops for minimum vertex cover. Specifically, the aim is to prove the following result (stated informally here; we formalize it later as Theorem 7):

Theorem. *Given a cubic bridgeless graph G, a represents table R and a vertex cover S derived using R, S is the minimum-size vertex cover derivable from the represents table R if and only if there is no S-diminishing hop in the represents table R.*

We prove the contrapositive of the theorem. The reverse direction follows from the definition of diminishing hops. For the forward direction, we first create a subgraph G' by taking the symmetric difference between some vertex cover S and S' such that $|S| > |S'|$. Then we study properties of G' and show its relation to the vertex covers S and S'. Subsequently, we show how to identify an S-diminishing hop in table R based on graph G'. Having identified an S-diminishing hop in table R and having mapped it to G', we prove that if we cannot identify an S-diminishing hop in table R, it implies S is the minimum vertex cover because a smaller vertex cover S' does not exist. We elaborate upon the proof and the corresponding concepts in Section 5.3.

Finally, we formally state the algorithm (Section 6) that amalgamates the three phases. It shows that finding S-diminishing hops takes time polynomial in the size of input, in line with Blossom algorithm showing that finding M-augmenting paths takes time polynomial in the size of input.

Output: a minimum vertex cover S

3 Notation and Preliminaries

We kindly refer the reader to standard texts in theoretical computer science (TCS) for the definitions of complexity classes **P** and **NP** and other definitions in complexity theory such as polynomial-time reductions and **NP**-completeness. For example, for a lucid overview, please refer to Sections 2.1 and 2.2 in [Kho19], which is an interesting paper discussing foundational work leading to the proof of the 2-to-2 Games Theorem. For comprehensive definitions and discussion, please refer to a handbook of TCS [VL91] or to some of the standard TCS textbooks [KT06, GJ02, CLRS22]. We treat these standard definitions as done in the literature. Additionally, our algorithm treats a graph as a set of vertices and a set of edges. Hence, throughout the paper, we perform standard set operations on a given graph. Therefore, graph operations implicitly follow all standard set theory laws (e.g., associative, commutative, etc.). See Appendix B of [CLRS22] for details on set operations and laws. We now define the computational problems related to finding the vertex cover of a given graph. First, we define the search/optimization problem:

Definition 1 (Minimum Vertex Cover Problem (`MVC`))**.** *Given a graph G, what is the smallest non-negative integer k such that the graph G has a vertex cover S of size k?*

Next, we restate the above as a decision problem and formalize the difference between the search version and the decision version of the computational problem:

Definition 2 (Vertex Cover Problem (`VC`))**.** *Given a graph G and a non-negative integer k, does the graph G have a vertex cover S of size at most k?*

Unless stated otherwise, we henceforth discuss solving `VC` (i.e., the *decision* version of the vertex cover problem as stated in Definition 2), which is **NP**-complete. Next, to define the computational problems corresponding to the variants of `VC` we use in the paper, we first define the graphs that will be used.

Definition 3 (Cubic Graphs)**.** *A cubic graph, also called a 3-regular graph or a trivalent graph, refers to a graph in which each vertex has a degree of three.*

Definition 4 (Bridgeless Graphs)**.** *A bridgeless graph, also known as a 2-edge-connected graph or an isthmus-free graph or a graph with no cutedge, is a graph that does not contain an edge, called a bridge*[16]*, whose deletion increases the number of connected components in the graph.*

Consequently, a cubic bridgeless graph is a graph in which each vertex has a degree of three and there are no bridges. Henceforth, graphs refer to arbitrary graphs. We specifically use the terms cubic and bridgeless when necessary. Finally, we define the computational problems that use cubic and bridgeless graphs, which is the focus of this paper.

Definition 5 (Vertex Cover Problem on Cubic Graphs ($\texttt{VC}-\texttt{CG}$))**.** *Given a cubic graph G and a non-negative integer k, does the cubic graph G have a vertex cover S of size at most k?*

Definition 6 (Vertex Cover Problem on Cubic Bridgeless Graphs ($\texttt{VC}-\texttt{CBG}$))**.** *Given a cubic bridgeless graph G and a non-negative integer k, does the cubic bridgeless graph G have a vertex cover S of size at most k?*

While $\texttt{VC}-\texttt{CG}$ is known to be **NP**-complete [GJS74], the complexity of the $\texttt{VC}-\texttt{CBG}$ is not known. Finally, please note that we define other terminology used in this paper in situ.

An unconditional deterministic polynomial-time algorithm: Throughout the paper, an algorithm refers to an *exact* algorithm unless noted otherwise. An unconditional algorithm does not depend on any assumptions. A deterministic algorithm always produces the same output for a given input. Finally, the number of operations of a polynomial-time algorithm is upper bounded by a polynomial in the size of the input (denoted by $\mathcal{O}()$). An example of an unconditional deterministic polynomial-time algorithm is the AKS primality test algorithm that takes polynomial time in the size of the input (number of bits to represent a number $(\log n)$) to deterministically compute whether a given number is prime or not without relying on any hypothesis/conjecture [AKS04].

[16] In other words, an edge is a bridge if and only if it is not contained in any cycle.

I $\texttt{VC} - \texttt{CBG}$ IS NP-COMPLETE

In this Part I of the paper, we establish the **NP**-completeness of the vertex cover problem on cubic bridgeless graphs ($\texttt{VC} - \texttt{CBG}$) by reducing from a known **NP**-complete problem, namely, the vertex cover problem on cubic graphs ($\texttt{VC} - \texttt{CG}$).

The vertex cover problem on cubic graphs is trivially known to be **NP**-complete because the vertex cover problem on graphs with vertex degree at most three is **NP**-complete [GJS74] (GJS74 calls vertex cover as node cover). However, the hardness of the $\texttt{VC}$ on cubic bridgeless graphs does not follow the hardness of the $\texttt{VC}$ on cubic graphs. This is because the stipulation that the graph is bridgeless introduces a restriction to the family of cubic graphs being considered. Such structural restrictions are known to make the $\texttt{VC}$ problem tractable. For example, the $\texttt{VC}$ on claw-free graphs[17] is in **P** as a consequence of the maximum independent set problem on claw-free graphs being in **P** [Sbi80, Min80] (we refer the reader to a survey on claw-free graphs for more details [FFR97]). Hence, restricting the cubic graphs to being bridgeless requires us to establish its computational complexity. Furthermore, the reduction used to prove the hardness of the $\texttt{VC}$ on graphs with vertex degree at most three in Theorem 2.4[18] in [GJS74] does not necessarily consist of a bridgeless graph. Thus, its reduction cannot be used to establish the hardness of the $\texttt{VC}$ on cubic bridgeless graphs.

In summary, we need to establish the hardness of the $\texttt{VC}$ on cubic bridgeless graphs because (i) the stipulation of graphs being bridgeless introduces a restriction to the family of cubic graphs being considered and such restrictions may make the problem tractable and (ii) the reduction used to prove the hardness of the $\texttt{VC}$ on graphs with vertex degree at most three consists of bridges. Therefore, we prove Theorem 1 in Part I, which proves that the vertex cover problem on cubic bridgeless graphs ($\texttt{VC} - \texttt{CBG}$) is **NP**-complete.

Part I Contribution and Organization: We show that $\texttt{VC} - \texttt{CBG}$ is **NP**-complete (Section 4).

4 Proof of NP-completeness of $\texttt{VC} - \texttt{CBG}$

For consistency, we restate the theorem we prove:

Theorem (1 restated). *The vertex cover problem on cubic bridgeless graphs (*$\texttt{VC} - \texttt{CBG}$*) is* **NP**-*complete.*

Proof. The proof consists of two parts: (i) we show $\texttt{VC} - \texttt{CBG} \in \mathbf{NP}$ and (ii) we show $\texttt{VC} - \texttt{CBG}$ is **NP**-hard by giving a polynomial time reduction from an instance of a known **NP**-hard problem, namely, the vertex cover problem on cubic graphs ($\texttt{VC} - \texttt{CG}$), to an instance of the vertex cover problem on cubic bridgeless graphs ($\texttt{VC} - \texttt{CBG}$). The latter is denoted by $\texttt{VC} - \texttt{CG} \leq_P \texttt{VC} - \texttt{CBG}$.

$\underline{\texttt{VC} - \texttt{CBG} \in \mathbf{NP}}$ Given a cubic bridgeless graph $G = (V, E)$, a candidate solution consisting of a set of vertices $S \subseteq V$, and a non-negative integer k, we can verify in polynomial time whether vertices in candidate solution S form a vertex cover of size at most k or not.

$\underline{\texttt{VC} - \texttt{CG} \leq_P \texttt{VC} - \texttt{CBG}}$ We reduce an instance of the vertex cover problem on cubic graphs ($\texttt{VC} - \texttt{CG}$) to an instance of the vertex cover problem on cubic bridgeless graphs ($\texttt{VC} - \texttt{CBG}$).

[17] A claw in a graph is a complete bipartite subgraph $K_{1,3}$. A claw-free graph is a graph that does not have a claw as an induced subgraph.

[18] Theorem 2.4 is Theorem 2.6 when referring to the journal version of the paper in Theoretical Computer Science, Volume 1, Issue 3, February 1976, Pages 237-267.

A. Construction. Given an instance of $\texttt{VC}-\texttt{CG}$ consisting of graph $G=(V,E)$, we reduce it to an instance of $\texttt{VC}-\texttt{CBG}$ consisting of graph $G'=(V',E')$ as follows:

Vertices: We have one vertex $x_i \in X$ for each vertex $v_i \in V$ and $6m+10n$ dummy vertices $d \in D$ where m corresponds to the number of vertices in the graph G and n corresponds to the number of edges in the graph G. Specifically, we divide the dummy vertices into two types of blocks:

- Block type B_1 consists of n blocks and each block consists of ten vertices, namely, vertex $i \in B_1$, $\forall\ i \in [0,\ 9]$.
- Block type B_2 consists of m blocks and each block consists of six vertices. Specifically, for each vertex $u \in G$, we have vertices $\{u', u'', u'_1, u'_2, u''_1, u''_2\} \in B_2$.

Hence, there are $10\cdot n$ dummy vertices in blocks of type B_1 and $6\cdot m$ dummy vertices in blocks of type B_2. Thus, $|D| = 10n+6m$. Overall, we set $X = \{x_1,\ldots,x_m\}$ and the dummy vertex set $D = \{d_1,\ldots,d_{6m+10n}\}$. Hence, the vertex set $V' = X \cup D$ is of size $|V'| = |X|+|D| = (m)+(6m+10n) = 7m+10n$ vertices.

Vertex Cover Size: We set the target vertex cover size to be $k+3m+6n$.

Edges: Recall that (i) the target instance should form a cubic bridgeless graph and (ii) w.l.o.g., we have assumed the graphs being considered are simple connected, which means the given instance of $\texttt{VC}-\texttt{CG}$ contains no loops, no multiple edges, and no unconnected components[19]. Therefore, to get a bridgeless graph, we replace each edge $e=(u,v)\in E$ in graph G to ensure no bridges remain. To do so, we first create a list L consisting of each edge and its endpoints $e=(u,v)\in E$. Next, consider a snapshot of the graph G depicting an arbitrary edge $e \in E$ connecting vertices u and v in the given instance of $\texttt{VC}-\texttt{CG}$. The endpoints u and v are connected to the vertices $\{a,b,c,d\}\in V$, which are further connected to other vertices not depicted here or among themselves or both. We are given a cubic graph, hence each snapshot centered on edge e looks as follows:

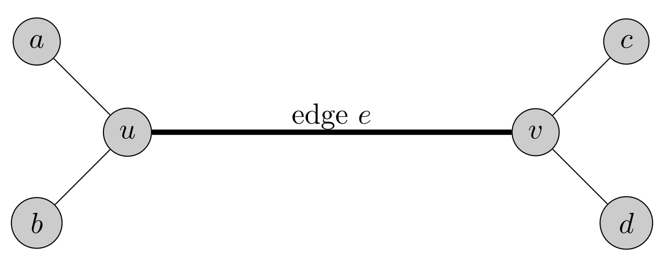

Figure 2: A snapshot of an instance of $\texttt{VC}-\texttt{CG}$ centered on an edge $e \in E$.

Next, for each edge $e=(u,v)\in L$ connecting vertices u and v in graph G of the instance of $\texttt{VC}-\texttt{CG}$, we perform the following **"atomic"** operations[20] where we (i) split the edge e by

[19] W.l.o.g., we assumed that we use simple connected graphs. This assumption is rather trivial. If one aims to overcome it, we can first prove that the $\texttt{VC}$ on cubic simple graphs is **NP**-complete, which can then be used to prove that $\texttt{VC}-\texttt{CBG}$ is **NP**-complete. The former can be proved easily by removing each multiple edge and each loop and adding an edge that is connected to a subgraph of five dummy vertices such that the graph remains cubic.

[20] Here, the term "atomic" operation is used from a mathematical perspective where it refers to an operation that cannot be simplified or further broken down (in the context of this paper) and not from a computer science perspective used in the context of concurrent programming.

connecting it to a subgraph and (ii) split each endpoint u and v into three vertices each, as discussed below. This ensures that the graph remains cubic while becoming bridgeless[21].

- **Splitting an Edge e:** The edge e connecting the vertices u and v in an instance of $\texttt{VC}-\texttt{CG}$ is split and connected via a subgraph consisting of dummy vertices from Block type B_1 (yellow vertices) as depicted below:

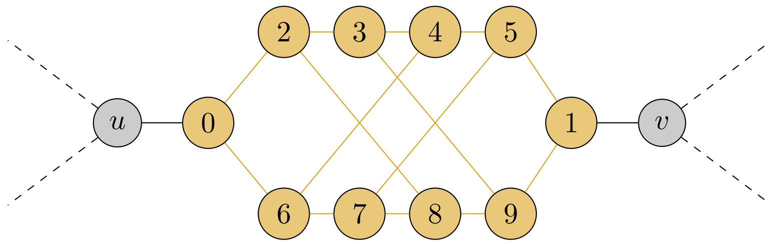

Figure 3: Splitting the edge e from the instance of $\texttt{VC}-\texttt{CG}$ by inserting a subgraph of dummy vertices from Block type B_1 in the instance of $\texttt{VC}-\texttt{CBG}$. Each dashed line denotes the existence of an edge.

- **Splitting Endpoints of Edge e:** Each endpoint u and v that is connected by the edge e in an instance of $\texttt{VC}-\texttt{CG}$ is split into three vertices each. We use dummy vertices from Block type B_2 (red vertices) and subsequently connect them as shown below:

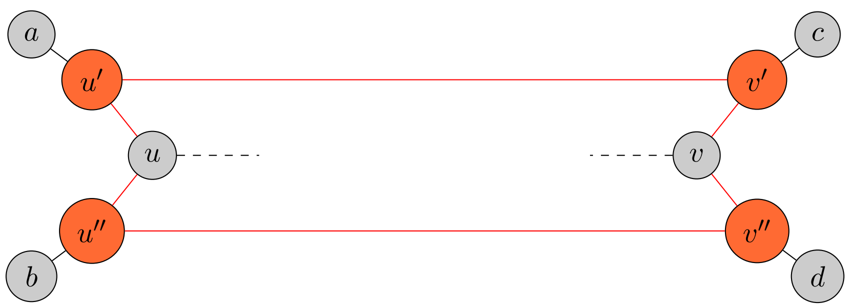

Figure 4: Splitting endpoints u and v into three vertices each in $\texttt{VC}-\texttt{CBG}$. The two dashed lines in the snapshot denote the existence of a Block Type B_1 subgraph depicted in Figure 3.

- **Updating the List L of Edges:** Remove the edge e connecting the endpoints u and v from the list L. Next, update the following edges in the list L:
  - edge connecting endpoints a and u is replaced by an edge connecting endpoints a and u' where the endpoint u' is the newly created vertex from Block type B_2, which was created by splitting the endpoint u

[21]In principle, a reduction to prove the same result can be constructed such that this exercise of splitting an edge and endpoints is done only for edges that are bridges. However, it entails (i) complications caused by the introduction of a variable, say λ, that counts the number of bridges in the given instance of graph G and (ii) needing a complex proof that encovers all cases of an edge being surrounded by r bridges where r is an integer between 0 and 4 (both inclusive) and in turn there being 5 cases encompassing $\binom{4}{r}$ possibilities about which of the four edge is a bridge. Therefore, we construct the discussed generalized reduction instance, which handles each edge, to simplify the proof.

- edge connecting endpoints b and u is replaced by an edge connecting endpoints b and u'' where the endpoint u'' is the newly created vertex from Block type B_2, which was created by splitting the endpoint u
- edge connecting endpoints c and v is replaced by an edge connecting endpoints c and v' where the endpoint v' is the newly created vertex from Block type B_2, which was created by splitting the endpoint v
- edge connecting endpoints d and v is replaced by an edge connecting endpoints d and v'' where the endpoint v'' is the newly created vertex from Block type B_2, which was created by splitting the endpoint v

The list L is updated after every split of edge e and the split of the corresponding endpoints that connect the edge e.

The above-discussed construction operations are "atomic" in that each edge e in the list L is split following the same procedure discussed above, and both its endpoints, independent of whether they are dummy vertices or not, are split following the same procedure discussed above. Each "atomic" operation transforms the given snapshot of $\texttt{VC} - \texttt{CG}$ (Figure 2) into the following snapshot of the (sub-)graph of $\texttt{VC} - \texttt{CBG}$:

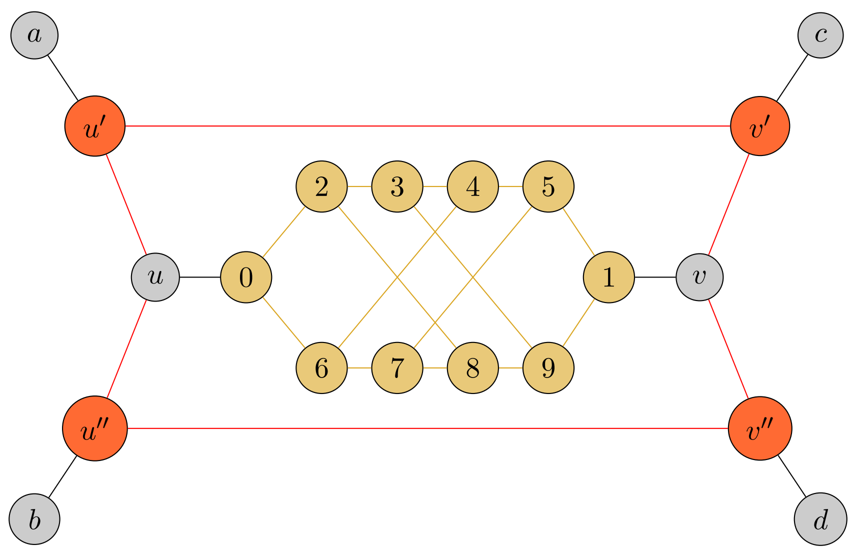

Figure 5: A snapshot of an instance of $\texttt{VC} - \texttt{CBG}$ corresponding to the snapshot of $\texttt{VC} - \texttt{CG}$ (Figure 2).

Finally, it remains to be discussed how a series of n "atomic" operations, one for each edge in the instance of $\texttt{VC} - \texttt{CG}$, leads to the complete construction of an instance of $\texttt{VC} - \texttt{CBG}$. This implies that a total of n edges are split. We also discuss the need for $10n + 6m$ dummy vertices. Next, we show that there is no edge in the constructed instance of $\texttt{VC} - \texttt{CBG}$ that is not a part of at least one cycle. In turn, it implies the constructed instance of $\texttt{VC} - \texttt{CBG}$ is bridgeless (and, of course, cubic):

- **Number of "Atomic" Operations is n:** The list L starts with n edges corresponding to the n edges in the instance of $\texttt{VC} - \texttt{CG}$. Subsequently, after each "atomic" operation, the split edge is removed from the list L. Hence, n "atomic" operations are carried out.

- **Number of Dummy Vertices in Block Type B_1 is $10n$:** This is trivial because when an edge is split, the split edge is replaced by a graph consisting of 10 vertices (Figure 3). There are n edge split operations, which result in $10n$ dummy vertices in Block type B_1.

- **Number of Dummy Vertices in Block Type B_2 is $6m$:** Each vertex has a degree of three. Hence, the operation of splitting of an endpoint occurs three times for each vertex. More specifically:
  - Initially, each endpoint u is connected to three edges that will be split.
  - Next, when the endpoint u is split (for the first time), the vertex u gets connected to two dummy vertices u' and u'' (Figure 4). By design, the vertex u is now *not* connected to any edge that will be split because (i) one of the edges it was connected to got split and (ii) each of the remaining two edges that need to be split are now connected to the dummy vertices u' and u'', respectively.
  - The dummy vertex u' is connected to one edge that needs to be split. Hence, when that edge is split, the endpoint u' is split, and in turn, it is connected to two dummy vertices u'_1 and u'_2. By design, the dummy vertex u' is now not connected to any edge that will be split. Simultaneously, the two dummy vertices u'_1 and u'_2 are also not connected to any edge that will be split.
  - The dummy vertex u'' is also connected to one edge that needs to be split. Hence, when the edge is split, vertex u'' is split and connected to two dummy vertices u''_1 and u''_2, in line with what was done for dummy vertex u'.
  - Overall, each vertex u is split into 6 dummy vertices $\{u', u'', u'_1, u'_2, u''_1, u''_2\}$. There are m vertices in total, which results in $6m$ dummy vertices.
- **Each Edge in the Constructed Instance of $\texttt{VC} - \texttt{CBG}$ is a Part of At Least One Cycle:** During an "atomic" operation, when the two endpoints of an edge is split (Figure 4), the resultant dummy vertices are connected such that (i) they form a cycle and (ii) they form a boundary around (a) the endpoints that were split and (b) the 10 dummy vertices from Block type B_1 that were inserted to split the edge. The combination of points (i) and (ii) ensures that the edges connecting the endpoints that were split and the edges connecting the 10 dummy vertices from Block type B_1 are also part of a cycle. Thus, an "atomic" operation guarantees that each new edge is part of a cycle. Consequently, after n "atomic" operations, each edge will be part of at least one cycle. The reason for this is mainly that there's at least one edge that belongs to the boundary resulting from one "atomic" operation and also to the boundary resulting from another. This fact can be interpreted in two ways: (i) two cycles share at least one edge in common or (ii) each "atomic" operation enlarges the boundary to encompass all the newly inserted edges. In either case, it means that no edge is a bridge in the constructed instance of $\texttt{VC} - \texttt{CBG}$. Therefore, the reduction ensures that the graph is bridgeless.
- **Each Vertex in the Constructed Instance of $\texttt{VC} - \texttt{CBG}$ has Degree Three:** When a vertex is split, the split vertex and the corresponding dummy vertices are connected to three other vertices. When an edge is split, there are two dummy vertices in the subgraph of Block type B_1 that are connected to the two endpoints of the split edge, and each of the remaining eight dummy vertices is connected internally with three other vertices. Hence, trivially, each vertex has a degree of three.

This completes our construction for the reduction, which is a polynomial time reduction in the size of n and m.

We refer the reader to Figure 6, which depicts a snapshot of the constructed instance of $\texttt{VC} - \texttt{CBG}$ when each of the five edges depicted in the snapshot of the instance of $\texttt{VC} - \texttt{CG}$ (Figure 2) is split.

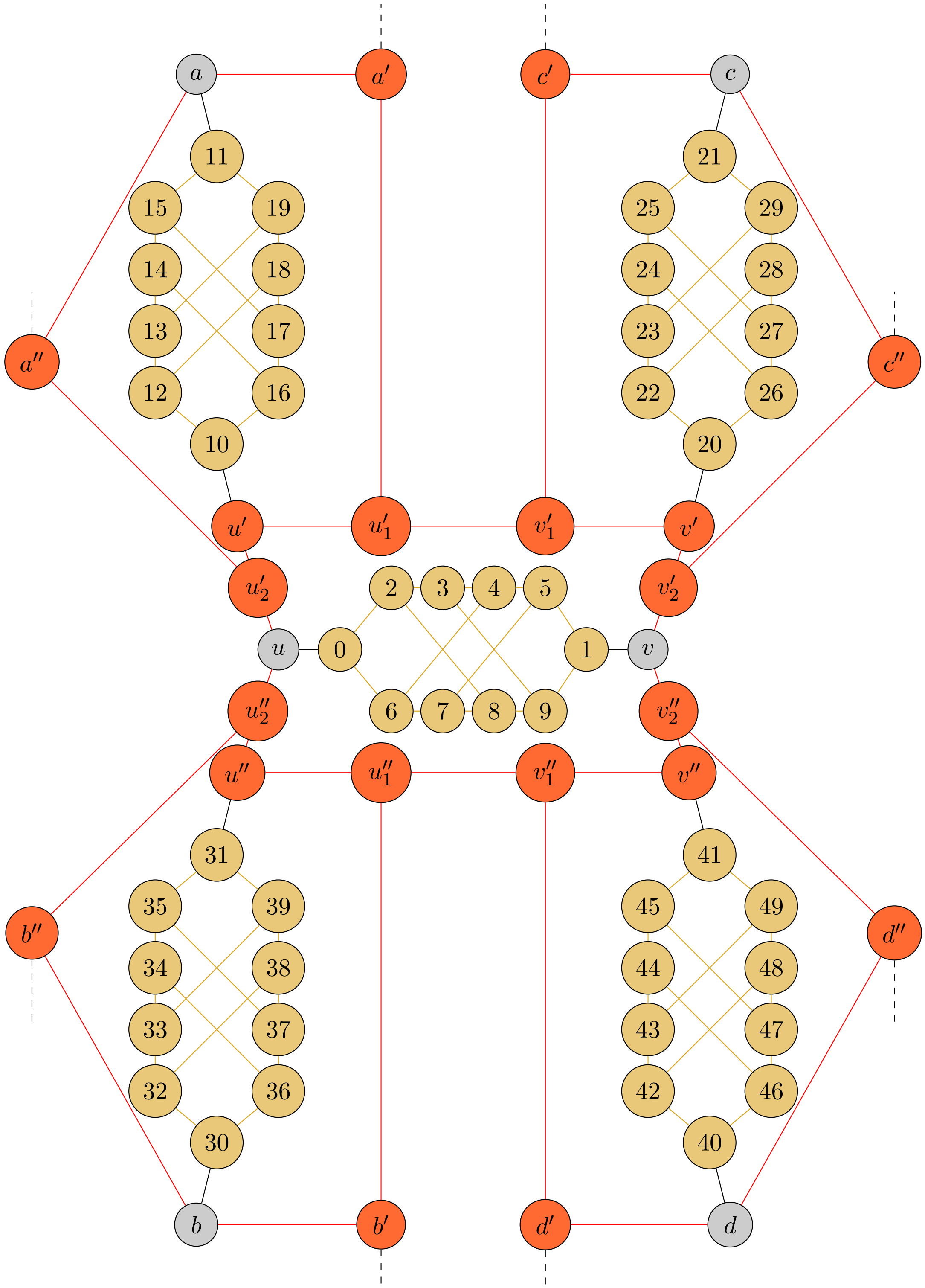

Figure 6: A snapshot of an instance of $\texttt{VC}-\texttt{CBG}$ that corresponds to an instance of $\texttt{VC}-\texttt{CG}$ (Figure 2). The construction depicts the transformation of five edges (and corresponding six vertices) shown in Figure 2; the edges that were not depicted are not transformed. Each dashed line denotes the existence of an edge.

Specifically, it denotes execution of 5 “atomic” operations. We stress that the figure is a snapshot of the reduction taking place; it is for illustrative purposes and depicts a stage of the reduction.

B. Proof of Correctness.

Claim 1. *We have a vertex cover S of size at most k that satisfies $e \cap S \neq \emptyset$ for all edges $e \in E$ if and only if we have a vertex cover S' of size at most $k + 3m + 6n$ that satisfies $e' \cap S' \neq \emptyset$ for all edges $e' \in E'$.*

($\Rightarrow$) If the instance of the $\texttt{VC} - \texttt{CG}$ problem is a yes instance, then the corresponding instance of $\texttt{VC} - \texttt{CBG}$ is a yes instance.

Six Vertices are Selected from each Block type B_1: Consider any one block of ten candidates from the n blocks of type B_1. The size of a minimum vertex cover for the subgraph consisting of those ten vertices is six. Adding an edge and a vertex to the subgraph can only increase the size of the minimum vertex cover. Hence, for each block in Block type B_1, we select six vertices, namely, vertex numbers ending with $0, 1, 3, 5, 6$, or 8, into the vertex cover S' of the instance of $\texttt{VC} - \texttt{CBG}$. This results in a total of **6n** vertices in Vertex Cover S' of the instance of $\texttt{VC} - \texttt{CBG}$.

Three Vertices are Selected from each of the $m - k$ Blocks of type B_2 that Correspond to $m - k$ Vertices Not in Vertex Cover S of $\texttt{VC} - \texttt{CG}$: For each vertex u *not* in the vertex cover S of an instance of $\texttt{VC} - \texttt{CG}$, we have three vertices in the vertex cover S' of an instance of $\texttt{VC} - \texttt{CBG}$. Specifically, for each vertex $u \notin S$, we have vertex u and corresponding dummy vertices u' and u'' in the vertex cover S'. We know that for each vertex $u \in G$, we split it and have $\{u, u', u'', u'_1, u'_2, u''_1, u''_2\} \in G'$. These seven vertices form a linear chain. Additionally, the vertices u, u', and u'' are all connected to either a vertex ending with 0 or a vertex ending with 1 from one of the corresponding blocks of Block type B_1. This means that vertices u, u', and u'' may or may not cover edges connecting them to vertices ending with 0 or vertices ending with 1 because the latter two are already in the vertex cover S'. Finally, given that vertex $u \notin S$, all three vertices connected to vertex u in graph G will be in the vertex cover S. Hence, vertices u'_1, u'_2, u''_1, and u''_2 in graph G' need not cover any edges that are not part of the linear chain (more on this in the next paragraph). Therefore, we use the following proposition on the minimum vertex cover on a linear chain graph:

Fact 1. *Given a linear chain graph G consisting of m vertices, the size of the minimum vertex cover for the graph G is $\lfloor \frac{m}{2} \rfloor$.*

The seven vertices form a linear chain. Therefore, the minimum size vertex cover for the seven vertices is of size three. We choose (central) vertices u, u', and u'' to form the minimum vertex cover. Given that there are $m - k$ vertices that are not in the vertex cover S, it corresponds to $3 \cdot (m - k)$ vertices in the vertex cover S'. This results in an additional **3m − 3k** vertices in Vertex Cover S' of the instance of $\texttt{VC} - \texttt{CBG}$.

Four Vertices are Selected from each of the k Blocks of type B_2 that Correspond to k Vertices in Vertex Cover S of $\texttt{VC} - \texttt{CG}$: For each vertex u in the vertex cover S of an instance of $\texttt{VC} - \texttt{CG}$, we have four vertices in the vertex cover S' of an instance of $\texttt{VC} - \texttt{CBG}$. Specifically, for each vertex $u \in S$, we have corresponding dummy vertices u'_1, u'_2, u''_1, and u''_2 in the vertex cover S'.

More specifically, if a vertex u is in the vertex cover S, then it covers all three edges connected to it. We call the three vertices connected to the vertex u via these three edges as *neighbors* of the vertex u. Additionally, we already discussed that corresponding vertices u, u', and u'' in the graph G' need not cover any edges (other than the edges on the linear chain) as they are connected with vertices that are already in the vertex cover S'. Simultaneously, vertices u_1', u_2', u_1'', and u_2'', in addition to being connected to the vertices in the linear chain, are also connected to other dummy vertices in graph G' that correspond to the neighbors of the vertex u in graph G. Therefore, the vertices u_1', u_2', u_1'', and u_2'' must be included in the vertex cover S'. This is required to cover the edges linking these four vertices to the "dummy" vertices that correspond to the neighbors of vertex u from the original graph G. This arrangement corresponds to (mirrors) how vertex u itself covers all the edges to its neighbors in graph G[22]. In turn, all the edges in the linear chain are also covered.

This results in the selection of 4 vertices in the vertex cover S' for each of the k vertices in the vertex cover S. Hence, there are **4k** more vertices in the vertex cover S' of the instance of $\texttt{VC} - \texttt{CBG}$.

Overall, for k vertices in the vertex cover S, we have $6n + 3(m - k) + 4k = 4k + 3m - 3k + 6n = k + 3m + 6n$ vertices in the vertex cover S'. Hence, a yes instance of the $\texttt{VC} - \texttt{CG}$ implies a yes instance of the $\texttt{VC} - \texttt{CBG}$ such that the vertex cover S' is of size at most $k + 3m + 6n$.

($\Leftarrow$) The instance of the $\texttt{VC} - \texttt{CBG}$ is a yes instance when we have $k + 3m + 6n$ vertices in the vertex cover S'. Then the corresponding instance of the $\texttt{VC} - \texttt{CG}$ is a yes instance as well. More specifically, we have the following cases when an instance of the $\texttt{VC} - \texttt{CBG}$ is a yes instance. The $\texttt{VC} - \texttt{CBG}$ is a yes instance when the minimum vertex cover S' contains:

1. **Six dummy vertices from each Block type B_1, Two dummy vertices from $m - k$ blocks of Block type B_2, Four dummy vertices from k blocks of Block type B_2, and $m - k$ vertices from set X:** This is the trivial case. The total number of vertices in the vertex cover $S' = 6n + 2(m - k) + 4k + m - k = 6n + 2m + m - 2k + 4k - k = 6n + 3m + k$. Note that this setup corresponds to the proof discussed in the forward direction. Hence, the $m - k$ vertices from the vertex set X that are in the vertex cover S' denote the vertices in set V of graph G that are not in the vertex cover S. Consequently, the k vertices in the set X that are not in the vertex cover S' denote the vertices that are in the vertex cover S. In summary, the corresponding instance of the $\texttt{VC} - \texttt{CG}$ is a yes instance because the k vertices $s \in S$ that form the vertex cover correspond to the k vertices in the set of vertices $X \setminus (S' \cap X)$.

2. **Six dummy vertices from each Block type B_1 and $k + 3m$ vertices from Block Type B_2 and set X:** The contribution of six dummy vertices from each Block type B_1 is straightforward and non-consequential in the context of this case. Nonetheless, we can easily modify the six vertices selected to include vertices ending with 0 and vertices ending with 1 in the vertex cover S'.

   Here, we focus our discussion on the selection of $k + 3m$ vertices from Block Type B_2 and set X. There are multiple ways to select the $k + 3m$ vertices that form a minimum vertex cover (in conjunction with the dummy vertices from B_1). However, among all such vertex covers, there is at least one vertex cover that is the same as Case 1, namely, has two dummy vertices from $m - k$ blocks of Block type B_2, four dummy vertices from k blocks of Block type B_2, and $m - k$ vertices from set X. The reason relies on a property of a linear chain, especially of even size: we can have multiple minimum vertex covers depending on which vertices are selected. In particular, one of the vertices on the end of the linear chain can be replaced with

[22] This is the same reason that vertices u_1', u_2', u_1'', and u_2'' in graph G' need not cover any edges that is not part of the linear chain when a vertex u is not in the vertex cover S.

its neighbor without affecting the size of the minimum vertex cover. Similarly, in our case, for each vertex $u \in V$, the corresponding vertices $\{u, u', u'', u'_1, u'_2, u''_1, u''_2\} \in V'$ form a linear chain, which allows manipulation of vertices selected in the vertex cover, even when the linear chain for us is of odd size. We discuss two points in this case:

(a) **three vertices from the linear chain are in the vertex cover S':** In the case of a linear chain of size seven (odd), the minimum vertex cover is of size three, and neither of the vertices at the end of the linear chain can be in the minimum vertex cover. There are $m - k$ such linear chains where three vertices are in the minimum vertex cover S'. The $m - k$ vertices not in the minimum vertex cover S correspond to these linear chains.

(b) **four vertices from the linear chain are in the vertex cover S':** When four vertices from a linear chain are in the minimum vertex cover S', it implies that at least one of the vertices is included to cover an edge that is not in the linear chain. W.l.o.g., let us begin with an assumption that the vertices $\{u, u', u''_1, u''_2\}$ (one vertex from the set X and three vertices from the Block type B_2) are a subset of vertices that is in the vertex cover S'. Then, these vertices can be replaced by the vertices $\{u'_1, u'_2, u''_1, u''_2\}$ (four vertices from Block type B_2) in the vertex cover S' because the vertices ending with 0 and vertices ending with 1 from Block type B_1 is in the vertex cover S' and the corresponding edges need not be covered by vertices in the linear chain. In general, when four vertices from the vertex set $\{u, u', u'', u'_1, u'_2, u''_1, u''_2\} \in V'$ are in the minimum vertex cover S', then *any* such combination of the four vertices can be replaced by vertices $\{u'_1, u'_2, u''_1, u''_2\}$ (four vertices from Block type B_2). The instance of $\texttt{VC} - \texttt{CBG}$ remains a yes instance with this modification. There are k such linear chains having four vertices in the minimum vertex cover S'. The k vertices in the minimum vertex cover S correspond to these linear chains.

Overall, there are the $3(m - k) + 4k = k + 3m$ vertices from Block Type B_2 and set X. In general, Case 2(b) shows that there is at least one minimum vertex cover S' of size $k+3m+6n$ that is the same as Case 1. Therefore, a yes instance of the $\texttt{VC} - \texttt{CBG}$ corresponds to a yes instance of the $\texttt{VC} - \texttt{CG}$ because the k vertices $s \in S$ that form the minimum vertex cover correspond to the k vertices in the set of vertices $X \setminus (S' \cap X)$.

These cases complete the other direction of the proof of correctness. In turn, it completes the overall proof that shows $\texttt{VC} - \texttt{CBG}$ is **NP**-complete. □

Part II $\texttt{VC} - \texttt{CBG} \in \mathbf{P}$

In this Part II of the paper, we discover an unconditional deterministic polynomial-time exact algorithm for the vertex cover problem on cubic bridgeless graphs ($\texttt{VC} - \texttt{CBG}$). The lack of an algorithm, and more generally, research on the $\texttt{VC} - \texttt{CBG}$ is both – quite surprising and unsurprising.

(Relative) Lack of Research on the $\texttt{VC} - \texttt{CBG}$ is Surprising: The matching theory within graph theory has received much attention. In particular, properties of a perfect matching and a maximum matching in a given graph have been studied extensively. Furthermore, matching in cubic bridgeless is also well-studied (see section 1.3). However, no analogous work that extensively discusses properties of the vertex cover[23], and in particular, properties of the vertex cover on cubic bridgeless graphs is known. This is surprising because there is a known relationship between the sizes of a maximum matching and a minimum vertex cover for a given graph (Lemma 1). Hence, it is intuitive to explore the existence of a deeper relation between the two. Additionally, a minimum vertex cover always exists (by definition), just like a maximum matching. Hence, an analog to Berge's Theorem [Ber57], which relates augmenting paths and maximum matching, should be explored.

Overall, the Blossom Algorithm [Edm65] was preceded by rich graph-theoretic work on maximum matching, perfect matching, and factorization. This facilitated a proof to show that the corresponding computational problem of finding a maximum matching is in **P** even when the problem then seemed to be similar to other typical graph optimization problems that later turned out to be "hard". Analogously, there is a need for us to better understand certain properties of the vertex cover.

No Algorithm for $\texttt{VC} - \texttt{CBG}$ is Unsurprising: While the lack of focus on understanding the properties of vertex cover, analogous to, say, Berge's Theorem for maximum matching, is surprising, the lack of an algorithm for the $\texttt{VC}$ and $\texttt{VC} - \texttt{CBG}$ is unsurprising. Karp's landmark paper on the twenty-one **NP**-complete problems brought the vertex cover problem ($\texttt{VC}$) to the attention of TCS researchers [Kar72]. Consequently, given that $\texttt{VC}$ was proven to be **NP**-complete, understanding its hardness-related computational aspects has been a focus of TCS researchers (see subsection 1.1)[24]. Additionally, one of the natural approaches to discover an algorithm for an **NP**-complete problem (and prove $\mathbf{P} = \mathbf{NP}$) directly relies on finding a polynomial-size Linear Programming or Semidefinite Programming that projects onto the polytope whose extreme points are the valid solutions. This approach was ruled out through a series of results [Yan88, FMP+15, LRS15, CLRS16, Rot17, CŽ24]. Hence, no effort on this front to find an algorithm for an **NP**-complete problem is unsurprising.

Next, we strengthen our unsurprising position about a lack of an algorithm for the $\texttt{VC} - \texttt{CBG}$, even when certain restrictions on graphs, such as bipartite and claw-free, make the $\texttt{VC}$ tractable. On the other hand, other restrictions on graphs, such as planar, do not affect the hardness of the $\texttt{VC}$. Hence, the surprisingly lack of understanding about the behavior of the vertex cover on bridgeless graphs, unsurprisingly, prohibits us from putting $\texttt{VC} - \texttt{CBG}$ in either camps (let us momentarily turn blind for this paragraph to the fact that we just proved $\texttt{VC} - \texttt{CBG}$ is **NP**-complete (Theorem 1). if not, we do not know if we can put $\texttt{VC} - \texttt{CBG}$ in both camps?). Neither do we know how the $\texttt{VC}$ behaves

[23] For the purposes of this discussion, a *"vertex cover"* (and its variants) refers to a discussion of graph-theoretic properties of the vertex covers and a "$\texttt{VC}$" (and its variants) refers to a discussion of the computational properties.

[24] These circumstances are strong reasons to speculate that the focus on the vertex cover shifted from graph theory-based results to computational complexity-based results.

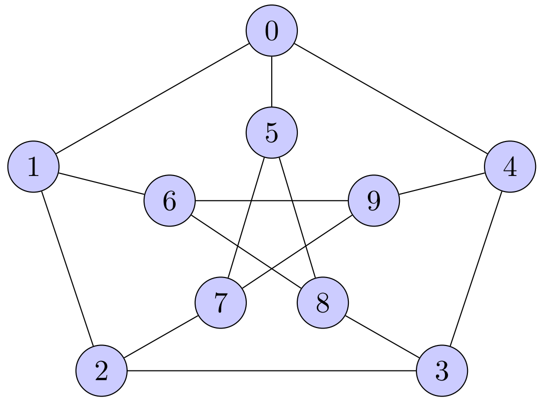

Figure 7: Petersen Graph used for the running example.

on bridgeless graphs, nor have the bridgeless graphs been studied on the other ten graph-based Karp's **NP**-complete problems. Moreover, even when we did not know much about the existence of a perfect matching in cubic graphs, Petersen showed that every cubic bridgeless graph has a perfect matching [Pet91]. Hence, just like the bridgelessness restriction on cubic graphs "easily" improved our understanding of the existence of a perfect matching, we aim to assess whether the bridgeless condition is conducive to our understanding of the vertex cover.

Overall, the absence of an algorithm for the $\texttt{VC} - \texttt{CBG}$ is unsurprising because: (i) $\texttt{VC}$ was proven **NP**-complete early and subsequent research focused on its hardness. (ii) $\texttt{VC} - \texttt{CBG}$ lacks the tools like those that helped prove maximum matching is in **P** and uses the understudied bridgeless graphs.

In summary, a lack of understanding of the non-computational aspects of the vertex cover (on cubic bridgeless graphs) is surprising. Simultaneously, the rich understanding of the computational aspects of the $\texttt{VC}$ and an absence of an algorithm to solve the $\texttt{VC}$ is unsurprising! As a result, we first improve our graph-theoretic understanding of the vertex cover on cubic bridgeless graphs. Then, we improve our understanding of the algorithmic aspects of the $\texttt{VC} - \texttt{CBG}$ (and $\texttt{VC}$)[25]. We use the Petersen Graph (Figure 7) as a running example to facilitate our discussion henceforth.

Example 1. *The Petersen Graph (Figure 7), a famous cubic bridgeless graph, is used as the given graph G throughout Part II.*

We prove the following theorem in Part II:

Theorem (2 restated). *The vertex cover problem on cubic bridgeless graphs ($\texttt{VC} - \texttt{CBG}$) is in* **P**.

Part II Contribution: We show that $\texttt{VC} - \texttt{CBG} \in \mathbf{P}$.

Part II Organization: In Section 5, we provide an overview of the three phases of an unconditional deterministic polynomial-time algorithm for the $\texttt{VC} - \texttt{CBG}$. We also discuss new (graph-theoretic) concepts and properties of the vertex cover on cubic bridgeless graphs that are needed

[25] Recall that the $\texttt{VC}$ on 2-regular graphs is in **P** and the $\texttt{VC}$ on 3-regular graphs is **NP**-complete. Hence, the $\texttt{VC}$ on 3-regular graphs is the closest known problem to a variant of the $\texttt{VC}$ in **P**. Next, if we consider this imaginative problem space between the $\texttt{VC}$ on 2-regular graphs and the $\texttt{VC}$ on 3-regular graphs, the restrictive case of cubic bridgeless lies somewhere in between these two ends. Hence, we are working on an algorithm for an **NP**-complete variant of $\texttt{VC}$ that is closest to the variant in **P** in the most conceivable way possible.

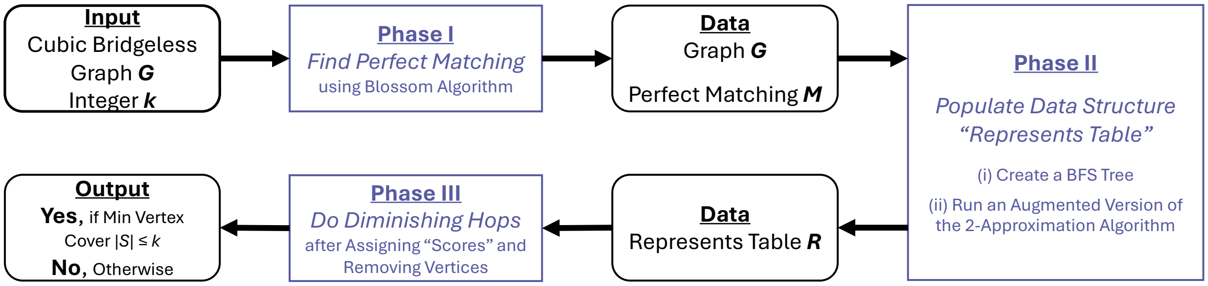

Figure 8: Figure 1 redrawn for ease of reference: A schematic representation of the three phases of the algorithm, with the corresponding input and output data for each phase clearly indicated.

for designing the algorithm and subsequently proving its correctness. In Section 6, we state the algorithm (which is abstracted into multiple algorithms for improved understanding). In Section 7, we provide the proof of correctness of the algorithm. In Section 8, we discuss the time complexity of the algorithm by providing an upper bound on its running time as a polynomial function in the size of the input. This is facilitated by a step-by-step time complexity analysis.

5 Algorithm Overview and Intermediate Results

We provide an overview of the algorithm. We also define, observe, and prove new concepts needed to prove the correctness of the algorithm. The algorithm is divided into three phases (Figure 8):

- **Phase I - Find a Perfect Matching:** The vertices are sorted lexicographically. Then, the Blossom Algorithm [Edm65] is used to find a maximum matching of the given graph. Given that we use cubic bridgeless graphs, the maximum matching is a perfect matching [Pet91][26].
- **Phase II - Populate a Novel Data Structure – Represents Table:** A breadth-first search (BFS) tree is constructed by seeding on the first vertex selected from a lexicographically sorted list of vertices. Next, an augmented version of the maximal matching algorithm (folklore 2-approximation algorithm for the vertex cover problem) is used to sequentially select edges that are part of the perfect matching. The output of this exercise is used to populate a novel data structure called the "Represents Table" (Table 2). Specifically, the data structure stores (i) the endpoints of the edges picked by the maximal matching algorithm in a row and (ii) in the same row, the neighboring vertices of the endpoints in a given iteration.
- **Phase III - Diminishing Hops:** The "Represents Table" is used to find the minimum vertex cover by: (i) assigning scores to each endpoint based on their "connectedness", (ii) removing the vertices with low scores and (iii) using a new technique called diminishing hops, analogous to the use of augmenting paths in maximum matching (Berge's Theorem [Ber57]).

[26] The time complexity of the Blossom Algorithm is $\mathcal{O}(m^2 n)$. Some algorithms are known to (i) find maximum matching faster than the Blossom Algorithm (for example, Micali and Vazirani's $\mathcal{O}(\sqrt{m} \cdot n)$ algorithm [MV80]) and (ii) specifically find a perfect matching in a cubic bridgeless graph faster than the Blossom Algorithm (for example, algorithms ranging from $\mathcal{O}(m \log^4 m)$ [BBDL01] to $\mathcal{O}(m \log m)$ [GW24]). However, the time complexity of the third phase (Diminishing Hops) dominates the complexity of the Blossom Algorithm. Hence, using a faster algorithm does not affect the overall time complexity of our algorithm. Moreover, the use of the Blossom Algorithm to find a maximum matching, instead of a specific algorithm to find a perfect matching, facilitates future work that can generalize our algorithm to other graphs.

We now discuss each of these three phases in detail.

5.1 Find a Perfect Matching

The first phase of the algorithm consists of the use of the Blossom Algorithm to find a maximum matching of the given graph. The maximum matching, in our case, is a perfect matching.

Definition 7 (Matching)**.** *Given a graph G, a matching M is a subset of the edges E such that no vertex $v \in V$ is incident to more than one edge in M.*

Alternatively, we can say that given a graph G, no two edges in a matching M have a common vertex. Consequently, a maximum matching is a matching with the highest cardinality.

Definition 8 (Maximum Matching)**.** *Given a graph G, a matching M is said to be maximum if for all other matchings M', $|M| \geq |M'|$.*

Equivalently, the size of the maximum matching M is the (co-)largest among all the matchings. A maximum matching that matches all the vertices of the graph is a perfect matching (Figure 9).

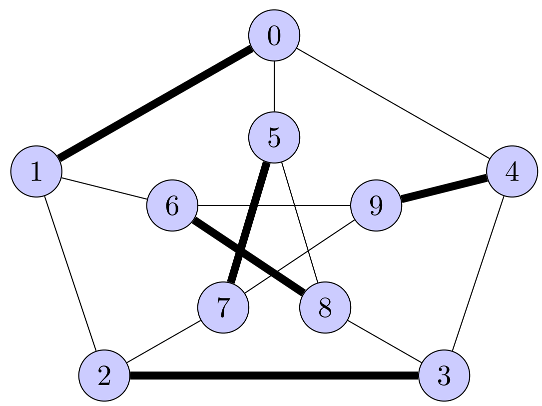

Figure 9: The bold edges of the Petersen Graph denote a perfect matching M.

Definition 9 (Perfect Matching)**.** *Given a graph G, a matching M is a perfect matching if each vertex $v \in V$ is incident to exactly one edge $e \in M$.*

In the general case, while every perfect matching is a maximum matching, every maximum matching may not be a perfect matching. However, in our case, every maximum matching found by the Blossom Algorithm is a perfect matching because we use cubic bridgeless graphs.

Theorem 3 (Petersen's Theorem [Pet91])**.** *Every cubic bridgeless graph contains a perfect matching.*

More generally, every cubic bridgeless graph contains exponentially many perfect matchings [EKK+11]. This is one of the reasons that we shall lexicographically sort the vertices as the first step of the algorithm. Sorting ensures that the Blossom Algorithm always returns the same matching for the same input. Next, there is a known relationship between the size of the maximum matching and the size of the minimum vertex cover:

Lemma 1. *In a given graph G, if M is a maximum matching and S is a minimum vertex cover, then $|S| \geq |M|$.*

Lemma 1 means that the largest number of edges in a matching does not exceed the smallest number of vertices in a cover. We use this fact to set a lower bound on the size of the minimum vertex cover. More specifically, we terminate the algorithm early if the given integer k is less than $|M|$. Moreover, given that the matching M is a perfect matching in our case, we know that $|M| = \frac{|V|}{2}$, which in turn implies that $|S| \geq \frac{|V|}{2}$.

In summary, in the Phase I of the algorithm, given a cubic bridgeless graph G and a list of lexicographically sorted vertices V_{sort}, we use the Blossom Algorithm to find and output a perfect matching M of the given graph. We refer the reader to [Edm65] for the Blossom Algorithm. In addition to the importance of Berge's Theorem in the Blossom Algorithm (discussed later in section 5.3), we note that the Blossom Algorithm does some extra work to handle the messy odd cycles, which, by transitivity, implies that our algorithm also handles the odd cycles.

5.2 Populate Represents Table

The second phase of the algorithm involves populating a novel data structure called the "Represents Table". Before populating the table, the algorithm stores the vertices at each level of a tree derived using breadth-first search (BFS):

Definition 10 (Breadth-First Search)**.** *Given a graph G, a Breadth-first Search (BFS) algorithm seeds on a root vertex $v \in V$ and visits all vertices at the current depth level of one. Then, it visits all the nodes at the next depth level. This is repeated until all vertices are visited.*

While the BFS algorithm is canonically a search algorithm, we use it here to derive a tree. This tree itself is not needed. We require the information about the level on which each vertex is in the BFS tree. It is needed for the next steps in the phase two of the algorithm.

Example 2. *We are given the Petersen graph G (Figure 7) and a seed vertex $0 \in V$. Hence, the BFS algorithm seeded on vertex 0 will return the following table regarding the level at which each vertex is in the BFS tree:*

Table 1: The vertices of the Petersen graph at each level of the BFS tree seeded on vertex 0.

Level	**Vertices**
1	$\{0\}$
2	$\{1, 4, 5\}$
3	$\{2, 3, 6, 7, 8, 9\}$

5.2.1 Augmented 2-Approximation Algorithm for the `VC`

This phase of the algorithm implements an augmented version of the 2-approximation algorithm for the `VC`. The 2-approximation algorithm is equivalent to finding the maximal matching of a given graph.

Definition 11 (Maximal Matching)**.** *Given a graph G, a matching M is said to be maximal if for all other matchings M', $M \not\subset M'$.*

In other words, a matching M is maximal if we cannot add any new edge $e \in E$ to the existing matching M. Next, recall that finding a maximal matching is equivalent to the vanilla 2-approximation algorithm, which guarantees a vertex cover of size at most twice the size of the minimum vertex cover: the algorithm picks an *arbitrary* edge $e = (u, v) \in E$, adds both the vertices u and v to the vertex cover S, removes all edges connected to either of the two vertices (u and v), and repeats until no edge remains. The vertex S is the resultant vertex cover. In this vanilla version, the method in which the edges are picked is *arbitrary* from two perspectives: (i) the *order* in which edges get picked is arbitrary, and (ii) consequently, *which* edge among the remaining edges gets picked is arbitrary. These two perspectives may seem similar but are different as outlined in the two steps discussed in the next paragraph.

We remove the above-mentioned arbitrariness in the edges that are picked. Specifically, during this phase, the edges are picked by following a two-step method:

- **Step 1 - Order in which the edges get picked:** We start with the seed vertex u on Level 1 of the BFS tree. Once an edge connected to this seed vertex is picked, the seed vertex and the other endpoint of the picked edge are marked as matched. We then move to Level 2 of the BFS tree. An edge connected to an unmatched vertex on level 2 is picked next[27]. Once all vertices on Level 2 are matched, we move to Level 3, and so on. More generally, the order in which the edges get picked is by following the levels of the BFS tree.

- **Step 2 - Which edge gets picked from a given order:** Each vertex u at level l is connected to (at most) three other vertices via (at most) three edges. Hence, among the (at most) three edges to choose from, an unpicked edge that is part of a given perfect matching M is picked. Recall that a perfect matching matches all the vertices of the graph, which means that each vertex is connected to exactly one edge in a given perfect matching M.

Example 3. *We are given the Petersen graph G, a perfect matching M (Figure 9), and the levels of a BFS tree seeded on vertex* 0 *(Table 1).*

During the first iteration of the augmented 2*-approximation algorithm, we start with vertex* 0*, because as per step 1, it is the seed vertex at level* 1 *in the BFS tree. Consequently, as per step 2, we choose the edge connecting vertex* 0 *and vertex* 1 *because that edge is in the perfect matching M (Figure 10a). We mark the two endpoints of the picked edge as matched and remove all edges that connect the two endpoints (Figure 10b).*

In the next iteration, we choose vertex 4*. This is because, as per step 1, it is the first vertex among all the unmatched lexicographically sorted vertices on level* 2 *of the BFS tree (namely, we choose vertex* 4 *from vertices* 4 *and* 5*). Next, as per step 2, we choose the edge connecting vertex* 4 *and vertex* 9 *because that edge is in the perfect matching M (Figure 10c). We mark the two endpoints of the picked edge as matched and remove all edges that connect the two endpoints (Figure 10d).*

We repeat this exercise until all the edges in the perfect matching are picked and consequently, no edge remains in the graph.

Note that the vanilla 2-approximation algorithm would have picked edges arbitrarily. Therefore, even when a given graph has a perfect matching, the vanilla 2-approximation algorithm may pick a set of edges that may not be a perfect matching. Hence, we augment the algorithm to ensure that all the edges in a perfect matching are picked. We will discuss the reason for doing so in the next section. However, our decision implies that the augmented 2-approximation algorithm always selects all edges in a perfect matching, which implies that the resultant vertex cover consists of all the vertices. Formally, this happens because of the combination of the following two known lemmas (or more formally, lemmata):

[27] The tie regarding which vertex on the same level l gets picked first is broken using the lexicographical ordering of the vertices such that a vertex at position i in the ordering is preferred over a vertex at position j, for all non-negative integers $i < j$. Similarly, all ties henceforth are broken based on the lexicographical ordering of the vertices.

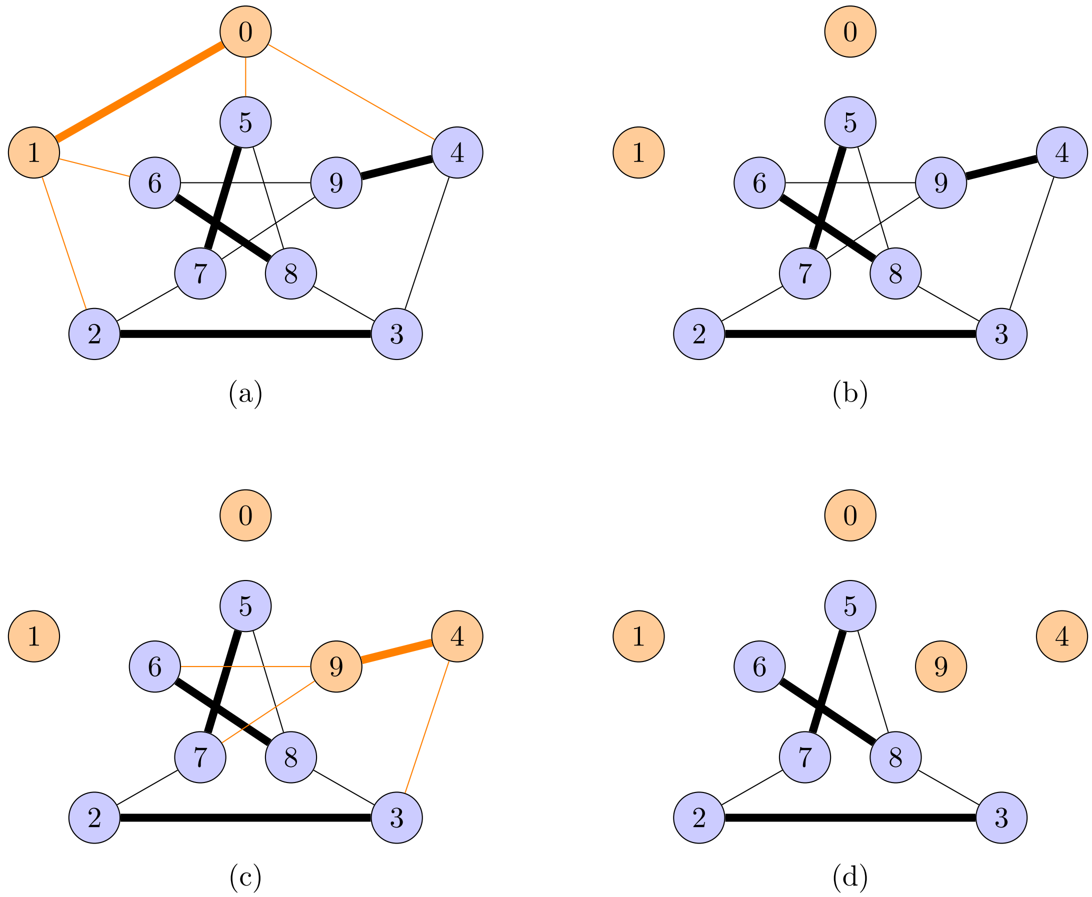

Figure 10: (a) Vertex 0 is on the Level 1 of the BFS tree. Hence, an edge in the perfect matching M that is connected to the vertex 0 is picked. Therefore, the edge connecting vertices 0 and 1 is picked. (b) All the edges connected to the two endpoints are removed. (c) Vertices 4 and 9 are the two endpoints of the second edge picked. (d) All the edges connected to the two endpoints are removed.

Lemma 2. *In a graph G, if a matching M is maximum, it implies that the matching M is also maximal. The converse does not hold.*

Note that due to Petersen's Theorem (Theorem 3), a perfect matching and a maximum matching mean the same thing in the case of cubic bridgeless graphs.

Lemma 3. *The endpoints of a maximal matching form a vertex cover.*

Overall, given that the augmented 2-approximation algorithm picks edges that are in a perfect matching, we know that the edges form a maximal matching too. Hence, its endpoints, which consist of all the vertices in the graph, form a vertex cover (trivially). Importantly, the augmented 2-approximation algorithm picks the edges in a particular order. This does not alter the above discussion but is crucial in how the represents table gets populated and affects its properties.

5.2.2 Represents Table

We discuss the novel data structure called the "Represents Table", which is a concept borrowed from its brief mention in [Rel24]. It is an augmented version of a table data structure and consists of unique properties and operations. Let us first define the property that led to the name *represents* table. It is based on a concept where a vertex u that is connected to a vertex v via an edge is said to represent[28] the other vertex.

Definition 12 (Represents)**.** *Given a graph G, a vertex $u \in V$ is said to* ***represent*** *a vertex $v \in V$ when the vertex v is connected to the vertex u by an edge $e \in E$. Conversely, the vertex v is* ***represented by*** *the vertex u.*

Observe that when a vertex u represents a vertex v, it is an alternative way of saying that an edge connects the vertices u and v. Additionally, given that we use a cubic (bridgeless) graph, each vertex represents three vertices and each vertex is represented by three vertices. The list of vertices that a vertex u represents is stored in a list called a Represents List.

Definition 13 (Represents List)**.** *Given a graph G, a vertex $u \in V$ is said to represent a set of vertices $V' \subseteq V \setminus \{u\}$ if there exists an edge between the vertex u and every vertex in V'. These vertices that the vertex u represents are in the represents list L_u such that for all vertices $u \in V$, $L_u = \bigcup_{e \in E} e \setminus \{u\} \mid u \in e$.*

In the context of this paper, we can restate the above definition as follows: Given a cubic graph G, a vertex $u \in V$ that is connected to three vertices x, y, and z by an edge each is said to represent the vertices x, y, and z. These vertices that vertex u represents are in the represents list L_u such that $L_u = \{x, y, z\}$. The size of each represents list in this paper is at most three (because we use cubic graphs). We are now ready to define the represents table:

Definition 14 (Represents Table)**.** *A represents table R is a 4-column table where a row stores the two endpoints of an edge picked during an iteration of the execution of the augmented 2-approximation algorithm, and for each endpoint u, also stores the corresponding represents list L_u, which consists of the vertices the endpoint u represents.*

Example 4. *We are given the Petersen graph G (Figure 9), a perfect matching M and the levels of a BFS tree seeded on vertex 0 (Table 1). As discussed in Example 3, the first iteration of the augmented 2-approximation algorithm picks the edge connecting the vertices 0 and 1 and removes all the edges that are connected to the two endpoints of the picked edge (Figure 10a and Figure 10b). Then, the corresponding entry in the represents table R is as shown in Table 2.*

The first endpoint, namely vertex 0 represents vertices 1, 4, and 5. The second endpoint, namely vertex 1 represents vertices 0, 2, and 6.

At this point, one may argue that the represents table is our fancy way of renaming an adjacency list. However, given the differences between their properties, we avoid using the latter term to avoid the confusion and to ensure that the represents table is visualized as a data structure that is different from an adjacency list. More specifically, unlike the adjacency list where each row enlists all the vertices connected to a vertex, the represents table stores the two endpoints of a picked edge in

[28] Informally, the term is inspired by a type of committee election where each voter approves of 2 candidates and the aim is to elect the smallest committee that represents every voter such that at least one of every voter's approved candidate is in the committee. In our context, we want to select the smallest set of vertices that covers (represents) each edge. Hence, think of vertices as candidates and edges as voters.

Table 2: A row in the Represents Table R depicts (i) the two **endpoints** of an edge picked by the augmented 2-approximation algorithm and (ii) the corresponding **represents list** of each of the two endpoints. A represents list consists of the vertices connected to an endpoint during a given iteration of the algorithm.

Endpoint 1	**Represents List 1**	**Endpoint 2**	**Represents List 2**
0	$L_0 = \{1, 4, 5\}$	1	$L_1 = \{0, 2, 6\}$

the *same* row. The corresponding represents list enlists *only* the vertices that are connected to an endpoint during a *given iteration* of the augmented 2-approximation algorithm. The stress on the words *given iteration* signifies that for a vertex to be listed in the represents list, an edge connecting the vertex and the endpoint should not have been removed during any of the previous iterations of the 2-approximation algorithm.

Example 5. *We continue the discussion from Example 4 where we had populated the first row of the represents table (Table 2).*

The second iteration of the augmented 2-approximation algorithm picks the edge connecting the vertices 4 and 9 (Figure 10c). Note that the represents list for vertex 4 enlists the vertices 9 and 3. Vertex 0 is not listed because the edge connecting vertices 0 and 4 was removed during the first iteration. The represents list for the vertex 9 is $L_9 = \{4, 6, 7\}$. Next, the algorithm now removes all the edges that are connected to the two endpoints of the picked edge (Figure 10d).

Similarly, the represents table is populated until the augmented 2-approximation algorithm terminates. Finally, the represents table R is populated completely and it looks as follows:

Table 3: A Represents Table R populated as a result of the implementation of the augmented 2-approximation algorithm for the vertex cover problem on a given instance of the Petersen graph, a corresponding perfect matching M, and a BFS tree.

Endpoint 1	**Represents List 1**	**Endpoint 2**	**Represents List 2**
0	$L_0 = \{1, 4, 5\}$	1	$L_1 = \{0, 2, 6\}$
4	$L_4 = \{9, 3\}$	9	$L_9 = \{4, 6, 7\}$
5	$L_5 = \{7, 8\}$	7	$L_7 = \{5, 2\}$
2	$L_2 = \{3\}$	3	$L_3 = \{2, 8\}$
6	$L_6 = \{8\}$	8	$L_8 = \{6\}$

The example shows that each vertex of Petersen graph is listed as an endpoint in the represents table (Table 3). Similarly, given a cubic bridgeless graph, each vertex of a given cubic bridgeless graph is listed as an endpoint in the represents table. This is because a cubic bridgeless graph has a perfect matching (Theorem 3). Therefore, each vertex is an endpoint of an edge in the perfect matching. Hence, the augmented 2-approximation algorithm will enlist each vertex as an endpoint in the represents table.

Lemma 4. *Given a cubic bridgeless graph, each vertex v of the graph is listed as an endpoint in the corresponding represents table.*

Operations and Properties of the Represents Table: We now discuss the operations that are supported by the represents table and discuss the properties relevant to this paper.

Operations The represents table supports four basic operations, namely, insert, access, freeze, and remove. The represents table does *not* support deletion of any information, as will become evident during the discussion of the diminishing hops phase of our algorithm.

- **insert:** The most basic operation supported by the table is the insertion of a vertex and its corresponding represents list. Additionally, the insertion operation does not need to access previous data. Hence, the asymptotic running time for each insert operation is $\mathcal{O}(1)$. We witnessed this operation while populating the represents table.

- **access** / search: (i) To access an endpoint u in the represents table, we do a sequential search for the given endpoint. This takes $\mathcal{O}(m)$. (ii) To access the represents list of an endpoint u, we need to first access the endpoint u, which takes $\mathcal{O}(m)$. Subsequently, accessing the represents list L_u takes $\mathcal{O}(1)$. (iii) To access each of the represents list that consists of a vertex u, we need to traverse through each of the m represents lists, each of constant size (three). This takes $\mathcal{O}(m)$. Hence, the asymptotic running time for each access operation is $\mathcal{O}(m)$.

- **freeze:** The freeze operation is used to freeze an endpoint. In the context of this paper, when we freeze an endpoint, it implies that the frozen vertex is selected as one of the vertices in the vertex cover. Next, whenever an endpoint u is frozen, it is simultaneously delisted from each of the represents lists it is a part of. This is analogous to marking every edge that touches the vertex u chosen for the vertex cover as being covered. Finally, the entire represents list L_u of the vertex u is delisted.

  Freezing an endpoint u takes $\mathcal{O}(m)$ time as we need to do a sequential search for the endpoint u. The delisting of the vertex u from each of the represents lists also takes $\mathcal{O}(m)$: for each of the m endpoints, we need to traverse through its represents list of size at most three and delist the vertex u from the represents list if present. The delisting of the represents list L_u takes $\mathcal{O}(1)$. Hence, the asymptotic running time for each freeze operation is $\mathcal{O}(m)$.

- **remove:** The remove operation is used to remove an endpoint. In the context of this paper, when we remove an endpoint u, it implies that the removed endpoint is *not* selected as one of the vertices in the vertex cover. Hence, each of the three vertices that are connected to the removed endpoint needs to be in the vertex cover to cover the edges that are connected to the removed vertex. Hence, the corresponding steps carried out in the represents table are as follows: each vertex in the represents list of the removed endpoint u *and* each vertex that represents the endpoint u is frozen.

Table 4: The time complexity of each operation carried out on the Represents Table.

Operation	**Time Complexity**
insert	$\mathcal{O}(1)$
access	$\mathcal{O}(m)$
freeze	$\mathcal{O}(m)$
remove	$\mathcal{O}(m)$
delete	operation not allowed

The removal of an endpoint u takes $\mathcal{O}(m)$ time as we need to search for the endpoint u using a sequential search. The freeze operation will be carried out three times and each one takes $\mathcal{O}(m)$. Hence, the asymptotic running time for each remove operation is $\mathcal{O}(m)$.

These are the main operations that can be carried out on the represents table (Table 4). The space complexity of the table is $\mathcal{O}(m)$.

Properties We discuss unique properties of the represents table, which are essential for our discussion on the phase three (diminishing hops) of our algorithm.

Property 1. *Given a represents table R, an endpoint u in the i^{th} row of the table R can only represent an endpoint v if the endpoint v is in row j of the table R, for all $1 \leq i \leq j \leq \frac{m}{2}$. Conversely, an endpoint u in the j^{th} row of the table R can be represented by an endpoint v only if the endpoint v is in the i^{th} row of the table R, for all $1 \leq i \leq j \leq \frac{m}{2}$.*

Property 1 discusses that an endpoint in an earlier row within the represents table can only represent endpoints in the same row or any later row. It cannot represent endpoints in rows that come before it. Conversely, an endpoint in a later row within the represents table can only be represented by endpoints in the same row or any earlier row. It cannot be represented by endpoints in rows that come after it. Overall, these describe a "directional" relationship within represents table. Endpoints in earlier rows can represent endpoints in the same or later rows, and conversely, endpoints in later rows can be represented by endpoints in the same or earlier rows.

Property 2. *Given a represents table R, endpoints u and v in the i^{th} row of the table R always represent each other, i.e., endpoint $u \in L_v$ and endpoint $v \in L_u$ if endpoints u and v are in the i^{th} row for all $1 \leq i \leq \frac{m}{2}$.*

Property 2 discusses that endpoints in the same row of the represents table *always* represent each other. Property 2 adheres to Property 1 and Property 2 can be considered a specific case of Property 1. However, neither of the properties implies the other. Moreover, when we combine these two properties, they imply three possibilities about the representations related to an endpoint u in the table R when a graph is cubic : (i) u is represented by two other endpoints in rows above itself and does not represent any endpoint in rows below itself, (ii) u is represented by one endpoint in rows above itself and it represents one endpoint in rows below itself, and (iii) u is represented by zero endpoints in rows above itself and represents two endpoints in rows below itself.

Property 3. *Given a represents table R, an endpoint $u \in R$ corresponds to a vertex $u \in V$ of the corresponding graph G, and the set of endpoints in the represent table R forms a vertex cover $S \subseteq V$ of the corresponding graph G.*

Property 3 refers to the simple one-to-one mapping of an endpoint in the represents table R and a vertex in the corresponding graph G. Additionally, in the general case, the endpoints in the represents table R form a vertex cover. This is because the endpoints of edges picked during maximal matching form a vertex cover. In our case of cubic bridgeless graphs, we always have a perfect matching and hence, all vertices of G will be an endpoint in the represents table R and these trivially form a vertex cover (because all vertices of a graph form a vertex cover).

Property 4. *Given a represents table R, if each endpoint $u \in R$ is either frozen or removed, then the frozen endpoints form a vertex cover $S \subseteq V$ of the corresponding graph G.*

Property 4 discusses the specific case when all the endpoints of the represents table R are either frozen or removed, and specifically the frozen endpoints form a vertex cover. The frozen endpoints form a vertex cover, so we focus our discussion on the removed endpoints. By design, when an endpoint is removed, all endpoints that it represents or is represented by are automatically frozen. This implies that no edge in the corresponding graph remains uncovered. Hence, the frozen endpoints will form a vertex cover when every endpoint is either frozen or removed. The frozen vertices do not form a vertex cover when at least one endpoint is neither frozen nor removed. This is because we can freeze, say, $m-2$ endpoints and not touch the remaining 2 endpoints. The frozen endpoints may not form a vertex cover because the remaining 2 endpoints may be connected via an edge. Hence, the condition that each endpoint in the represents table R be either frozen or removed is necessary for the frozen endpoints to form a vertex cover.

Overall, the operations performed on the represents table result in the above-discussed unique properties of the represents table. These operations and properties form the foundation for the discussion of the next phase of the algorithm.

In summary, in the Phase II of the algorithm, given a cubic bridgeless graph G, a list of lexicographically sorted vertices V_{sort}, and a perfect matching M as input, we create a BFS tree, run an augmented version of the 2-approximation algorithm for the `VC`, and populate a novel data structure called the represents table R. The output of this phase is the represents table R. Additionally, we discussed how the represents table was created and listed its operations and properties:

1. We began with an underlying data structure, a table.
2. We used the BFS tree and the augmented 2-approximation algorithm to collect specific information needed to populate the represents table.
3. We discussed the corresponding insert operation that is used to enter the information into the represents table. We also discussed the access, freeze, and remove operations.
4. We analyzed the time complexity of each operation[29].
5. We observed some unique properties of the represents table.

5.3 Diminishing Hops

The diminishing hops phase of the algorithm constitutes the core contribution of the paper. To understand the concept of diminishing hops and its relevance to the minimum vertex cover, we first introduce the concept of "representation scores" and use it to decide whether to freeze or remove an endpoint from the represents table. We then discuss augmenting paths and its relevance to the maximum matching (Berge's theorem [Ber57]), which serves as a motivation to finally introduce diminishing hops for the minimum vertex cover (Figure 11).

5.3.1 Representation Score

The idea for using Representation Score is to associate a score with each endpoint in the represents table R such that the score of each endpoint is used to decide whether to freeze an endpoint or remove it. The score is a quantification of the information related to each edge that connects the

[29] We may improve the time complexity of the operations by augmenting the existing data structure "represents table" with a doubly linked list or a hash table. However, we leave such improvements to future work.

Augmenting Paths for Maximum Matching

$\Downarrow \quad \Downarrow \quad \Downarrow$

Diminishing Hops for Minimum Vertex Cover

Figure 11: A high-level illustration of augmenting paths being a motivation for diminishing hops.

vertices of the graph. More specifically, the score of an endpoint is a weighted number that captures how well an endpoint is represented by other endpoints in the table.

The assignment of the score begins from the top row of the represents table R. The score assigned to an endpoint u is the sum of the scores of the endpoint that is on the same row as each endpoint that represents u. For instance, consider that the endpoint u is in the i^{th} row of the represents table R, for some integer $i > 1$. Next, if some endpoint x in row j, for all $j \in [1, i-1]$, represents the endpoint u, then the score of endpoint y that is in the same row j as endpoint x is added to the score of endpoint u plus one. The score of each endpoint is initialized to zero.

Definition 15 (Representation Score)**.** *The representation score ζ of an endpoint u in row i of the represents table R is denoted by*

$$\zeta_u = \sum(\zeta_y + 1)$$

where endpoint y is in the same row as endpoint x for all endpoints x such that $u \in L_x$ and $y \neq u$. By design, the endpoint y will be in row j for some integer j such that $1 \leq j < i$.

The higher the score of endpoint y, the higher the chance of endpoint u being frozen so that endpoint x can, in turn, be removed. Recall that both the endpoints in the first row have a score of zero each and the computation moves downward.

Example 6. *We use the populated represents table constructed in Example 5.*

The representation score of both endpoints in the first row is 0. Hence, $\zeta_0 = \zeta_1 = 0$.

There are two endpoints in the second row, namely 4 and 9. The endpoint 9 is not represented by any endpoint, which implies its score $\zeta_9 = 0$. The endpoint 4 is represented by endpoint 0. Hence, its score will be equal to the score of endpoint 1 (plus 1) because endpoint 1 is in the same row as endpoint 0. This means $\zeta_4 = \zeta_1 + 1 = 0 + 1 = 1$. Similarly, the representation score ζ will be appended to the represents table R as follows:

Table 5: The Represents Table R is appended with representation score ζ for each endpoint.

Representation Score ζ	**Endpoint 1**	**Represents List 1**	**Endpoint 2**	**Represents List 2**	**Representation Score ζ**
$\zeta_0 = 0$	0	$L_0 = \{1, 4, 5\}$	1	$L_1 = \{0, 2, 6\}$	$\zeta_1 = 0$
$\zeta_4 = 1$	4	$L_4 = \{9, 3\}$	9	$L_9 = \{4, 6, 7\}$	$\zeta_9 = 0$
$\zeta_5 = 1$	5	$L_5 = \{7, 8\}$	7	$L_7 = \{5, 2\}$	$\zeta_7 = 2$
$\zeta_2 = 3$	2	$L_2 = \{3\}$	3	$L_3 = \{2, 8\}$	$\zeta_3 = 1$
$\zeta_6 = 3$	6	$L_6 = \{8\}$	8	$L_8 = \{6\}$	$\zeta_8 = 7$

We jump to calculate the representation score of the last endpoint in the last row, namely, endpoint 8*. The endpoint* 8 *is represented by endpoints* 3 *and* 5*. Hence, its score will be equal to the sum of scores of endpoints* 2 *and* 7 *(plus 1 for each endpoint). This is because endpoint* 2 *is in the same row as endpoint* 3 *and endpoint* 7 *is in the same row as endpoint* 5*. Hence,* $\zeta_8 = (\zeta_2 + 1) + (\zeta_7 + 1) = (3 + 1) + (2 + 1) = 4 + 3 = 7$.

Finally, while computing the representation score, we traverse through the entire represents table by exploiting the fact that an endpoint u can be represented by at most two endpoints in rows above itself. This is because the vertex degree of each vertex is three and one endpoint is in the same row as endpoint u. Hence, the time complexity of computing the score is linear. We discuss the details about the algorithm to compute scores and its complexity in Section 6 and Section 8, respectively.

5.3.2 Freezing and Removing Endpoints using Representation Score

The representation scores are used to determine which endpoints to freeze or remove. The process[30] of freezing or removing an endpoint begins from the bottom row of the represents table R. There are two endpoints in the last row, u and v. We freeze the endpoint with a higher representation score ζ. Ties are broken using lexicographic ordering of the vertices. Simultaneously, we remove the other endpoint. Formally:

$$f(u, v) = \begin{cases} \text{freeze}(v) \text{ and remove}(u), & \text{if } \zeta_u < \zeta_v \\ \text{freeze}(u) \text{ and remove}(v), & \text{otherwise} \end{cases}$$

Recall that when an endpoint u is frozen, it is delisted from each represents list it is in. Therefore, for each endpoint $a \in R$, $L_a = L_a \setminus u$. Simultaneously, all entries in the represents list of u are delisted. Hence, $L_u = \emptyset$. Next, when an endpoint v is removed, all entries in the represents list of v is delisted ($L_v = \emptyset$) and each endpoint that represents the endpoint v is frozen. Finally, the representation scores of each endpoint in the remaining rows is recalculated top-down. The process now iteratively moves up row-wise of the represents table R. This process of freezing and removing endpoints of each row continues until each endpoint in the represents table R is either frozen or removed. However, during an iteration, when an endpoint in a given row is already frozen, the other endpoint is automatically removed. On the other hand, when an endpoint in a given row is already removed, the other endpoint, by design, would have been frozen. Finally, when both endpoints are frozen, no action is performed. In either case, the algorithm moves to the next iteration without the need to use the representation score. The case when both the endpoints are removed is impossible.

Example 7. *We use the represents table with representation scores constructed in Example 6.*

The process of freezing or removing an endpoint begins from the bottom row. Here, there are two endpoints, namely 6 *and* 8*, having representation scores of* $\zeta_6 = 3$ *and* $\zeta_8 = 7$.

First, freeze endpoint 8 *because it has a higher representation score* ($\zeta_8 > \zeta_6$ $(7 > 3)$)*.Then, delist the endpoint* 8 *from the represents lists* L_3 *and* L_5*. Also delist the entire represents list* L_8.

Next, remove endpoint 6*. Then, freeze the endpoints* 1 *and* 9 *because the removed endpoint* 6 *is in the represents lists* L_1 *and* L_9*. Delist the entire represents list* L_6*. Additionally, because endpoints* 1 *and* 9 *are frozen: (i) delist the endpoint* 1 *from the represents list* L_0 *and delist the endpoint* 9 *from the represents list* L_4*, and (ii) delist the entire represents lists* L_1 *and* L_9.

[30] A process here denotes a sequence of steps that are followed and should not be misinterpreted from the context of a process in operating systems.

Finally, recalculate the representation score of all the endpoints that are neither frozen nor removed. The represents table at the end of the first iteration, carried on the bottom row, looks as depicted in Table 6:

Table 6: The Represents Table R after the first iteration of freezing and removing endpoints based on the representation score. The bold red font denotes that an endpoint is frozen. The grayed-out entries denote removed / delisted endpoints.

Representation Score ζ	**Endpoint 1**	**Represents List 1**	**Endpoint 2**	**Represents List 2**	**Representation Score ζ**
$\zeta_0 = 0$	0	$L_0 = \{1, 4, 5\}$	**1**	$L_1 = \{0, 2, 6\}$	$\zeta_1 = 0$
$\zeta_4 = 1$	4	$L_4 = \{9, 3\}$	**9**	$L_9 = \{4, 6, 7\}$	$\zeta_9 = 0$
$\zeta_5 = 1$	5	$L_5 = \{7, 8\}$	7	$L_7 = \{5, 2\}$	$\zeta_7 = 0$
$\zeta_2 = 2$	2	$L_2 = \{3\}$	3	$L_3 = \{2, 8\}$	$\zeta_3 = 1$
$\zeta_6 = 3$	6	$L_6 = \{8\}$	**8**	$L_8 = \{6\}$	$\zeta_8 = 7$

At the end of all iterations, the endpoints 0*,* 3*,* 6*, and* 7 *are removed from the represents table* R*. The endpoints* 1*,* 2*,* 4*,* 5*,* 8*, and* 9 *are frozen in the represents table* R*.*

At the end of the process of freezing or removing endpoints using the represents table R, the frozen endpoints correspond to the vertices in a vertex cover S' of the graph G. We stress that at this moment, we do *not* make any claim about the size of the vertex cover.

Theorem 4. *Given a graph* G *consisting of a set* V *of* m *vertices and the corresponding represents table* R *populated by a set* W *of* m *endpoints, a set* S'' *of* l *endpoints in the represents table* R *is frozen, for some integer* $1 \leq l \leq m$*, and a disjoint set* $W \setminus S''$ *of* $m - l$ *endpoints in the represents table* R *is removed if and only if there is a vertex cover* S' *of* l *vertices in the graph* G *that correspond to the set* S'' *of frozen endpoints.*

Proof. ($\Rightarrow$) If a set S'' of l endpoints in the represents table R is frozen, for some integer $1 \leq l \leq m$, and a disjoint set $W \setminus S''$ of $m - l$ endpoints in the represents table R is removed, then there is a vertex cover S' of l vertices in the graph G that correspond to the set S'' of frozen endpoints.

When the represents table is populated, all the vertices in G are listed as endpoints. Hence, the endpoints trivially form a vertex cover (Property 3). Next, the computation of representation scores and the consequent process of freezing and removal of the endpoints results in each endpoint either being frozen or removed. Hence, the frozen endpoints form a vertex cover (Property 4).

($\Leftarrow$) If there is a vertex cover S' of l vertices in the graph G, for some integer $1 \leq l \leq m$, then a set S'' of l endpoints in the represents table R is frozen that correspond to the vertex cover S' and a disjoint set $W \setminus S''$ of $m - l$ endpoints in the represents table R is removed. This is the straightforward case, as we can simply freeze the endpoints that correspond to the vertices in the vertex cover S' and remove the rest of the endpoints. □

Again, we stress that the above-stated theorem guarantees that the frozen endpoints form a vertex cover and does not give any guarantee regarding the size of the vertex cover. We leave the analysis on the size of the vertex cover derived using representation scores for future work. Overall, here, we discussed the use of representation scores to freeze or remove each endpoint in the given represents table, and the resultant frozen endpoints form a vertex cover. We finally discuss how the represents table, representation score, and the vertex cover are used in the diminishing hops.

5.3.3 Diminishing Hops

The concept of diminishing hops is inspired by the use of augmenting paths for maximum matching (Figure 11). More specifically, we prove a theorem on the use of diminishing hops for minimum vertex cover, just like Berge's theorem is proven on the use of augmenting paths for maximum matching [Ber57]. To do so, we first discuss an augmenting path and its relation to a maximum matching proven through Berge's theorem. Then, we introduce a diminishing hop and show its relation to the minimum vertex cover through a theorem (analogous to Berge's theorem).

<u>Berge's Theorem, Augmenting Paths, and Maximum Matching</u> The Blossom Algorithm [Edm65] is a polynomial-time algorithm to find a maximum matching in a given graph. The key concept that the Blossom algorithm relies upon is Berge's theorem:

Theorem 5 (Berge's Theorem [Ber57])**.** *Given a graph G and a matching M, M is a maximum matching if and only if there is no M-augmenting path in the graph G.*

To understand this, we first define an alternating path and an augmenting path.

Definition 16 (Alternating Path)**.** *Given a graph G and a matching M, an alternating path P w.r.t. the matching M is a path that (i) starts from a vertex v that is not incident to any edge e in the matching M and (ii) whose edges alternate between not being in M and being in M (or being in M and not being in M if the path starts from a vertex v that is incident to any edge e in the matching M).*

Definition 17 (Augmenting Path)**.** *Given a graph G and a matching M, an augmenting path is an alternating path w.r.t. matching M that starts from a vertex v and ends at a vertex u such that $u \neq v$ and neither the vertex u or the vertex v is incident to any edge e in the matching M.*

An augmenting path's characteristic is that it increases the size of an existing matching. Hence, if there is an augmenting path P w.r.t. a matching M, then we can increase the size of the matching by one. To do so, we create a new matching M' by flipping the edges along the path P such that (i) if an edge is in M, then it is not in M' and (ii) if an edge is not in M, then it is in M' (Figure 12). Formally, the new matching M' can be denoted by $M' = (M \setminus P) \cup (P \setminus M)$ and hence, $|M'| = |M| + 1$. Berge's theorem uses this characteristic to prove that a matching M is maximum if and only if there is no augmenting path w.r.t. M (Theorem 5).

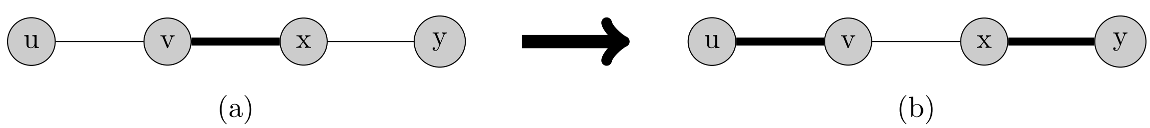

Figure 12: (a) An augmenting path from vertex u to y alternates between an edge not in a matching and an edge in the matching (thick edge). (b) It leads to an edge being augmented to the matching.

We practically described an algorithm to find a maximum matching: start with an initial matching (even a blank matching), augment the current matching with augmenting paths, and terminate when no augmenting path remains. The eventual augmented matching is a maximum matching. Subsequently, the key contribution of the Blossom algorithm is to find augmenting paths efficiently.

Similarly, this section first introduces the concept of a diminishing hop and then shows its relation to a minimum vertex cover. Finally, we discuss how to compute a diminishing hop efficiently in Section 6 (Algorithm) and analyze its time complexity in Section 8.

Diminishing Hop A represents table R consists of vertices V in graph G that are endpoints of edges in a maximum matching M found using the Blossom algorithm. In our case (of using cubic bridgeless graphs), all vertices of a given graph G will be endpoints in the represents table because a perfect matching always exists (Lemma 4). Next, if each endpoint in the represents table is either frozen or removed, then the frozen endpoints form a vertex cover (Property 4). Conversely, if a vertex cover S of a graph is given, then the corresponding endpoints in the represents table can be frozen and the rest removed (i.e., endpoints in S can be frozen and endpoints in $V \setminus S$ removed). Finally, in our case, the represents table consists of $\frac{m}{2}$ rows, which is also the lower bound on the size of the `MVC`.

Given that $\frac{m}{2}$ corresponds to the lower bound of an `MVC` and to the number of rows in the represents table, we ideally want exactly one endpoint from each row frozen and the other removed. Removing both endpoints of a given row is not possible by design. Hence, if there is a row in the represents table that consists of two frozen endpoints, called a duad, then it should be assessed if removing one of them can lead to a smaller number of frozen endpoints in the table. This corresponds to a smaller-sized vertex cover. Importantly, recall that $\frac{m}{2}$ is a lower bound and not the size of the `MVC`. Hence, it is perfectly possible for more than one row of the represents table to have both its endpoints frozen. The aim here is to assess whether decreasing the number of such rows is possible or not.

Definition 18 (Duad). *Given a represents table R where each endpoint u in the represents table R is either frozen or removed, a duad* [31] *refers to a pair of endpoints that (i) is in the same row of the represents table R and (ii) are both frozen.*

Consider a represents table R such that each endpoint is either frozen or removed and *exactly* one of its rows has both its endpoints frozen (a duad). The set of frozen endpoints corresponds to a vertex cover S. We shall generalize this discussion to when *at least* one duad exists later on by successively doing diminishing hops[32]. Hence, for now, given a represents table R where (i) each endpoint is either frozen or removed, (ii) each endpoint is marked "unvisited", and (iii) there is one duad, we carry out the following sequence of operations:

1. Endpoints u and v in row i are both frozen, for some integer $i \in [1, \frac{m}{2}]$. Both are marked "unvisited". Hence, we start the hopping phase by choosing an endpoint to remove, say u.

2. Remove endpoint u. Consequently, by design, each endpoint x is frozen such that either (i) x is represented by endpoint u or (ii) x represents endpoint u. Mark endpoints u and v of row i and each endpoint x as "visited". Enqueue endpoint u in a queue Q. Subsequently,

    (a) For all integers $j \in [i+1, \frac{m}{2}]$, consider an endpoint x in row j such that x is represented by endpoint u. If the j^{th} row has two frozen endpoints x and y, repeat Step 2 by choosing to remove endpoint y if endpoint y is marked "unvisited". Move to Step 2(b) when either an endpoint marked "visited" is encountered or endpoint u represents no endpoint or only one endpoint in row j is frozen and "visited".

[31] The term "dyad" (or dyadic) is more commonly and interchangeably used in English as compared to "duad" (or duadic), but not in a mathematical context. For instance, dyadic has a specific meaning in linear algebra. Hence, we use "duad" and "duadic" instead of "dyad" and "dyadic" to prevent confusion and emphasize that duadic and dyadic terms are unrelated.

[32] Indeed, the proof for diminishing hops should eventually seem similar to the proof connecting augmenting paths and maximum matching (Berge's Theorem: Theorem 1, [Ber57]). Hence, an understanding of the proof of Berge's Theorem will make our proof easier to follow.

(b) Dequeue an endpoint u from queue Q. For all integers $j \in [1, i-1]$, consider an endpoint x in row j such that x represents endpoint u. If the j^{th} row has two frozen endpoints x and y, repeat Step 2 by choosing to remove endpoint y if endpoint y is marked "unvisited". If no endpoint represents endpoint u or when an endpoint marked as "visited" is encountered or only one endpoint of row j is frozen and "visited", then either (i) repeat Step 2(b) if the queue Q is non-empty or (ii) move to Step 3 if the queue Q is empty.

3. Count the number of marked frozen vertices S_u resulting from removing endpoint u. Mark all the endpoints as "unvisited" and restore the represents table to the starting state (as was given before Step 1). Repeat Step 2 by removing endpoint v. Subsequently, count the number of marked frozen vertices S_v resulting from removing endpoint v.

4. The vertex cover S is the initial set of frozen endpoints in the given represents table R. Hence, if $|S| \leq |S_u|$ and $|S| \leq |S_v|$, then do nothing. Else if $|S_u| \leq |S_v|$, then the diminished vertex cover is S_u; else the diminished vertex cover is S_v.

We informally stated the algorithm for diminishing hops for the restricted case when we are given a represents table where each endpoint is either frozen or removed, and when there is exactly one duad. We formalize it in Section 6. Here, we discuss how a diminished vertex cover implies a minimum vertex cover, and when a given set of frozen endpoints is not diminishable, the given vertex cover is indeed the minimum. We first give an example:

Example 8. *We use the represents table that results after freezing or removing each endpoint discussed in Example 7. Recall that the endpoints* 0, 3, 6, *and* 7 *are removed from the represents table* R *and the endpoints* 1, 2, 4, 5, 8, *and* 9 *are frozen (gray endpoints and red endpoints, respectively, Table 7). The latter corresponds to a vertex cover. Also, observe that endpoints* 4 *and* 9 *in row* 2 *are both frozen. In other words, endpoints* 4 *and* 9 *in row* 2 *form a duad (yellow highlight).*

Table 7: The Represents Table R where each endpoint is either frozen (red font) or removed (gray font), and one row has both its endpoints frozen, i.e., a duad (yellow highlight). The representation score is not needed (entire column is gray font).

Representation Score ζ	**Endpoint 1**	**Represents List 1**	**Endpoint 2**	**Represents List 2**	Representation Score ζ
$\zeta_0 = 0$	0	$L_0 = \{1,4,5\}$	**1**	$L_1 = \{0,2,6\}$	$\zeta_1 = 0$
$\zeta_4 = 1$	**4**	$L_4 = \{9,3\}$	**9**	$L_9 = \{4,6,7\}$	$\zeta_9 = 0$
$\zeta_5 = 1$	**5**	$L_5 = \{7,8\}$	7	$L_7 = \{5,2\}$	$\zeta_7 = 0$
$\zeta_2 = 2$	**2**	$L_2 = \{3\}$	3	$L_3 = \{2,8\}$	$\zeta_3 = 1$
$\zeta_6 = 3$	6	$L_6 = \{8\}$	**8**	$L_8 = \{6\}$	$\zeta_8 = 7$

Next, as per Step 2, remove endpoint 4. *Hence, row* 2 *now has only one frozen endpoint. Mark both the endpoints as "visited" (green highlight, Table 8). Enqueue endpoint* 4 *to the queue* $Q = \{4\}$. *By design, endpoints* 0 *and* 3 *are frozen. This is because endpoint* 3 *is represented by endpoint* 4 *(see list* L_4*) and endpoint* 0 *represents endpoint* 4 *(see list* L_0*). Consequently, endpoints* 0 *and* 3 *are marked as "visited" (green highlight).*

Next, the hopping across the table begins. As depicted in Table 8, first hop to row 4 *that contains endpoint* 3 *(as per Step 2(a)). Now, as a consequence of freezing endpoint 3, row* 4 *consists of two frozen endpoints, namely,* 2 *and* 3. *It implies row* 4 *now forms a duad. Hence, we need to remove*

Table 8: Mark endpoints 4 and 9 as "visited" (green highlight). Remove endpoint 4. Consequently, endpoints 0 and 3 are frozen and marked "visited". We will first hop (red arrow) from row 2 to row 4, which corresponds to hopping from the removed endpoint 4 to endpoint 3 and then hop from row 2 to row 1, which corresponds to hopping from the removed endpoint 4 to endpoint 0.

Representation Score ζ	Endpoint 1	Represents List 1	Endpoint 2	Represents List 2	Representation Score ζ
$\zeta_0 = 0$	0	$L_0 = \{1,4,5\}$	1	$L_1 = \{0,2,6\}$	$\zeta_1 = 0$
$\zeta_4 = 1$	4	$L_4 = \{9,3\}$	9	$L_9 = \{4,6,7\}$	$\zeta_9 = 0$
$\zeta_5 = 1$	5	$L_5 = \{7,8\}$	7	$L_7 = \{5,2\}$	$\zeta_7 = 0$
$\zeta_2 = 2$	2	$L_2 = \{3\}$	3	$L_3 = \{2,8\}$	$\zeta_3 = 1$
$\zeta_6 = 3$	6	$L_6 = \{8\}$	8	$L_8 = \{6\}$	$\zeta_8 = 7$

one of the unvisited endpoints from row 4. Therefore, hop to endpoint 2 to remove it (i.e., repeat Step 2) (Table 9).

Table 9: Hop (red arrow) from frozen endpoint 3 to endpoint 2. The latter is subsequently removed.

Representation Score ζ	Endpoint 1	Represents List 1	Endpoint 2	Represents List 2	Representation Score ζ
$\zeta_0 = 0$	0	$L_0 = \{1,4,5\}$	1	$L_1 = \{0,2,6\}$	$\zeta_1 = 0$
$\zeta_4 = 1$	4	$L_4 = \{9,3\}$	9	$L_9 = \{4,6,7\}$	$\zeta_9 = 0$
$\zeta_5 = 1$	5	$L_5 = \{7,8\}$	7	$L_7 = \{5,2\}$	$\zeta_7 = 0$
$\zeta_2 = 2$	2	$L_2 = \{3\}$	3	$L_3 = \{2,8\}$	$\zeta_3 = 1$
$\zeta_6 = 3$	6	$L_6 = \{8\}$	8	$L_8 = \{6\}$	$\zeta_8 = 7$

We removed endpoint 2 (as per Step 2). Mark endpoint 2 as "visited" (green highlight, Table 10). Enqueue endpoint 2 to the queue $Q = \{4, 2\}$. Next, because endpoint 2 was removed, freeze and mark endpoints 7 and 1 as "visited" (green highlight, Table 10). Consequently, note that row 3 now has a duad because endpoint 7 was frozen. Next, we hop to rows where endpoint 2 is represented by an endpoint, namely, endpoint 1 in row 1 and endpoint 7 in row 3.

Table 10: Hop (red arrow) from row 4 to row 3 and row 1, which corresponds to hopping from removed endpoint 2 to endpoint 7 and endpoint 1, respectively.

Representation Score ζ	Endpoint 1	Represents List 1	Endpoint 2	Represents List 2	Representation Score ζ
$\zeta_0 = 0$	0	$L_0 = \{1,4,5\}$	1	$L_1 = \{0,2,6\}$	$\zeta_1 = 0$
$\zeta_4 = 1$	4	$L_4 = \{9,3\}$	9	$L_9 = \{4,6,7\}$	$\zeta_9 = 0$
$\zeta_5 = 1$	5	$L_5 = \{7,8\}$	7	$L_7 = \{5,2\}$	$\zeta_7 = 0$
$\zeta_2 = 2$	2	$L_2 = \{3\}$	3	$L_3 = \{2,8\}$	$\zeta_3 = 1$
$\zeta_6 = 3$	6	$L_6 = \{8\}$	8	$L_8 = \{6\}$	$\zeta_8 = 7$

At this point, no "unvisited" endpoint represents endpoint 2 or is represented by endpoint 2 (Table 11). Hence, as per Step 2(a), we move to Step 2(b). De-queuing queue $Q = \{4, 2\}$ gives us endpoint 4 and therefore $Q = \{2\}$. Endpoint 4 is represented by one endpoint, namely endpoint 0 *in row 1. We hop to row 1. Both the endpoints of row 1 are marked "visited" (and are frozen). Hence, we repeat Step 2(b) as queue Q is not empty.*

Table 11: Hop (red arrow) from row 2 to row 1, which corresponds to hopping from endpoint 4 extracted from queue Q to endpoint 0 (as depicted earlier in Table 8).

Representation Score ζ	**Endpoint 1**	**Represents List 1**	**Endpoint 2**	**Represents List 2**	Representation Score ζ
$\zeta_0 = 0$	**0**	$L_0 = \{1,4,5\}$	**1**	$L_1 = \{0,2,6\}$	$\zeta_1 = 0$
$\zeta_4 = 1$	4	$L_4 = \{9,3\}$	**9**	$L_9 = \{4,6,7\}$	$\zeta_9 = 0$
$\zeta_5 = 1$	**5**	$L_5 = \{7,8\}$	**7**	$L_7 = \{5,2\}$	$\zeta_7 = 0$
$\zeta_2 = 2$	2	$L_2 = \{3\}$	**3**	$L_3 = \{2,8\}$	$\zeta_3 = 1$
$\zeta_6 = 3$	6	$L_6 = \{8\}$	**8**	$L_8 = \{6\}$	$\zeta_8 = 7$

De-queuing queue $Q = \{2\}$ gives us endpoint 2 and therefore $Q = \{\}$. Endpoint 2 is represented by endpoints 1 and 7. Recall that we simply marked the endpoint 1 as "visited" because it is already frozen. Both the endpoints of row 1 are already marked "visited" (and are frozen). Note that even though row 1 is a duad, we do not perform any remove operation on either of the endpoints because each one is marked as "visited". Consequently, we hop to row 3 (Table 12). Row 3 consists of a pair of frozen endpoints, which means it is also a duad.

Table 12: Hop (red arrow) from row 4 to row 3, which corresponds to hopping from removed endpoint 2 extracted from queue Q to endpoint 7 (as depicted earlier in Table 10).

Representation Score ζ	**Endpoint 1**	**Represents List 1**	**Endpoint 2**	**Represents List 2**	Representation Score ζ
$\zeta_0 = 0$	**0**	$L_0 = \{1,4,5\}$	**1**	$L_1 = \{0,2,6\}$	$\zeta_1 = 0$
$\zeta_4 = 1$	4	$L_4 = \{9,3\}$	**9**	$L_9 = \{4,6,7\}$	$\zeta_9 = 0$
$\zeta_5 = 1$	**5**	$L_5 = \{7,8\}$	**7**	$L_7 = \{5,2\}$	$\zeta_7 = 0$
$\zeta_2 = 2$	2	$L_2 = \{3\}$	**3**	$L_3 = \{2,8\}$	$\zeta_3 = 1$
$\zeta_6 = 3$	6	$L_6 = \{8\}$	**8**	$L_8 = \{6\}$	$\zeta_8 = 7$

As previously mentioned, row 3 consists of two frozen endpoints, i.e., a duad. Among the two endpoints, one of them is marked "unvisited". Hence, hop from endpoint 7 to "unvisited" endpoint 5. Remove endpoint 5 as per Step 2 (Table 13).

Table 13: Hop (red arrow) from frozen endpoint 7 to endpoint 5. The latter is subsequently removed.

Representation Score ζ	**Endpoint 1**	**Represents List 1**	**Endpoint 2**	**Represents List 2**	Representation Score ζ
$\zeta_0 = 0$	**0**	$L_0 = \{1,4,5\}$	**1**	$L_1 = \{0,2,6\}$	$\zeta_1 = 0$
$\zeta_4 = 1$	4	$L_4 = \{9,3\}$	**9**	$L_9 = \{4,6,7\}$	$\zeta_9 = 0$
$\zeta_5 = 1$	5	$L_5 = \{7,8\}$	**7**	$L_7 = \{5,2\}$	$\zeta_7 = 0$
$\zeta_2 = 2$	2	$L_2 = \{3\}$	**3**	$L_3 = \{2,8\}$	$\zeta_3 = 1$
$\zeta_6 = 3$	6	$L_6 = \{8\}$	**8**	$L_8 = \{6\}$	$\zeta_8 = 7$

We removed endpoint 5 (as per Step 2). Mark it "visited" (Table 14) . Enqueue endpoint 5 to the queue $Q = \{5\}$. Consequently, freeze and mark endpoints 0 and 8 as "visited". Next, we first hop to the row where endpoint 5 represents an endpoint, namely, endpoint 8 in row 5. We then hop to row where endpoint 5 is represented by an endpoint, namely, endpoint 0 in row 1.

Table 14: Hop (red arrow) from row 4 to row 5 and row 1, which corresponds to hopping from removed endpoint 5 to endpoint 8 and endpoint 0, respectively.

Representation Score ζ	**Endpoint 1**	**Represents List 1**	**Endpoint 2**	**Represents List 2**	Representation Score ζ
$\zeta_0 = 0$	**0**	$L_0 = \{1,4,5\}$	**1**	$L_1 = \{0,2,6\}$	$\zeta_1 = 0$
$\zeta_4 = 1$	4	$L_4 = \{9,3\}$	**9**	$L_9 = \{4,6,7\}$	$\zeta_9 = 0$
$\zeta_5 = 1$	5	$L_5 = \{7,8\}$	**7**	$L_7 = \{5,2\}$	$\zeta_7 = 0$
$\zeta_2 = 2$	2	$L_2 = \{3\}$	**3**	$L_3 = \{2,8\}$	$\zeta_3 = 1$
$\zeta_6 = 3$	6	$L_6 = \{8\}$	**8**	$L_8 = \{6\}$	$\zeta_8 = 7$

More specifically, first hop to row 5 that contains endpoint 8 (as per Step 2(a)). Row 5 consists of one frozen endpoint, which is marked "visited". Note that endpoint 6 was already removed and hence, we need not visit it as only one endpoint from its row is frozen. Additionally, by design, each endpoint represented by and representing endpoint 6 is frozen. Hence, we move to execute Step 2(b). Next, de-queuing queue $Q = \{5\}$ gives us endpoint 5 and therefore $Q = \{\}$. Endpoint 5 is represented by one endpoint, namely endpoint 0 in row 1. We hop to row 1. Both the endpoints of row 1 are marked "visited" (and are frozen). Hence, we move to Step 3 as queue Q is empty. This completes one iteration of the hopping phase, which we call as a duadic hop.

Overall, the number of frozen endpoints is six. Hence, the corresponding vertex cover $S_4 = \{0, 1, 3, 7, 8, 9\}$ is of size six and is not smaller than S, the original vertex cover given to us. Therefore, we discard all changes made to the represents table and start with the represents table depicted in Table 7. Because the duadic hop we performed does not decrease the size of the vertex cover, it is not a diminishing hop[33]. Next, we repeat the entire exercise we discussed by removing the endpoint 9 of the duad in Table 7. Finally, at the end of the execution, we get the corresponding vertex cover S_9 of size six as well. Hence, as per Step 4, the vertex cover S is not diminishable. This is because neither of the two duadic hops is a diminishing hop. More specifically, each duadic hop resulted in a vertex cover of size that is same as the given vertex cover. Hence, the given vertex cover is indeed the minimum vertex cover, which we formally prove in the succeeding discussion.

We are now ready to define a diminishing hop and prove the corresponding theorem that relates the above-discussed restricted diminishing hops to the minimum vertex cover.

Definition 19 (Duadic Hop)**.** *Given a represents table R where each endpoint u in table R is either frozen or removed, a duadic hop H w.r.t. to the frozen endpoints in table R (equivalently w.r.t. to a given vertex cover S) is a sequence of operations that (i) starts at a row where both (duad of) the endpoints are frozen, (ii) removes one of the endpoints, (iii) freezes each endpoint that represents or is represented by the removed endpoint, and (iv) repeats each operation until no "unvisited" endpoint remains or "unvisited" but removed endpoints remain or an already frozen endpoint is encountered.*

[33] This statement is analogous to an alternating path not being an augmenting path in an instance of a maximum matching problem when an alternating path does not lead to an increase in the size of matching.

Table 15: An overview of the similarities between an augmenting path and a diminishing hop.

	Augmenting Path	**Diminishing Hop**
Where to start?	start at an exposed vertex in the graph, which means none of its edges is in the given matching	start at a row in the represents table where a duad of (i.e., both) endpoints are frozen (i.e., both endpoints of an edge are in the given vertex cover)
How to proceed?	follow an alternating path	perform a duadic hop
When to terminate?	when another exposed vertex is encountered in an alternating path, or no such vertex exists	when the number of endpoints removed > the number of endpoints frozen during a duadic hop, while satisfying the table's properties, or no such duadic hop exists
What is the use?	searching for augmenting paths is a technique to find a maximum matching (Theorem 5)	searching for diminishing hops is a technique to find a minimum vertex cover (Theorem 7)

A duadic hop is analogous to an alternating path (Definition 16), but unlike an alternating path, where the path alternates between edges in matching and not in matching, a duadic hop is a sequence of operations that alternately freezes and removes endpoints in the table R. Hence, while an alternating path exists in a graph, a duadic hop does not exist in a table R but is created. The nomenclature of a duadic hop is based on the fact that the hop starts with a duad (or pair) of frozen endpoints, removes one of them, and hops to another row, (possibly) creating a duad of frozen endpoints at the row it hopped to[34]. We now define a diminishing hop:

Definition 20 (Diminishing Hop)**.** *Given a represents table R where each endpoint u in table R is either frozen or removed, a diminishing hop H w.r.t. to the frozen endpoints in table R (equivalently w.r.t. to a given vertex cover S) is a duadic hop w.r.t. to a vertex cover S that removes at least one more endpoint than it freezes while satisfying all the properties of the table R.*

A diminishing hop is analogous to an augmenting path (Definition 17). The former diminishes the size of a given vertex cover, and the latter augments the size of a given matching. Let us elaborate on the similarity between the two concepts. To find a maximum matching, Berge [Ber57, Edm65] suggested searching for augmenting paths. Specifically, he suggested starting from an exposed vertex (i.e., a vertex which is connected to edges not in a matching). Then, walk along an alternating path by iteratively finding the path until it exists. Consequently, if this alternating path stops at an exposed vertex, then it is an augmenting path, and the size of the matching can be increased by one. On the other hand, if the path is not augmenting, then backtrack (a little), choose another edge, and continue to form an alternating path until no augmenting path exists.

Similarly, to find a minimum vertex cover, we propose searching for diminishing hops. Specifically, we are given a vertex cover S. The vertices in the vertex cover correspond to the frozen

[34]The word "possibly" is in the parenthesis because when using cubic bridgeless graphs and representation score ζ to freeze / remove endpoints as done in the paper, it is guaranteed to create a duad of frozen endpoints in at least one of the rows we hop to. However, if we do not use the representation score ζ to freeze / remove endpoints, then it is possible that such a duad may not be formed during a duadic hop. We leave this analysis to future work.

endpoints in the represents table (and vertices in $V \setminus S$ correspond to removed endpoints). If exactly one endpoint of each row is frozen, it is the minimum vertex cover, and we need not do anything. However, if we have two frozen endpoints in a row, then we start from a row where both of its endpoints are frozen. Then, move along a duadic hop by iteratively removing and freezing endpoints until possible. Consequently, when duadic hops stop, if the number of endpoints removed is greater than the number of endpoints frozen, then the duadic hop is a diminishing hop, and the size of the vertex cover is decreased (by one). On the other hand, if the duadic hop is not diminishing, then backtrack (a little), choose another endpoint to remove (or row with two frozen endpoints), and continue duadic hops until no diminishing hop exists. At this point, we stress that we do not discuss time complexity and focus on formalizing the relation between diminishing hops and minimum vertex cover.

Note that the definitions of a duadic hop and a diminishing hop are general in that they hold for diminishing hops for restricted and for the general case[35]. We now prove a theorem that associates diminishing hop and minimum vertex cover for the restricted case of represents table R where exactly one row contains two frozen endpoints. Subsequently, we prove a theorem that associates diminishing hop and minimum vertex cover for the general case of represents table R where at least one row contains two frozen endpoints. Recall that when one endpoint of each row is frozen, the frozen endpoints form a minimum vertex cover, and hence, we do not need to prove anything.

Theorem 6 (Restricted Diminishing Hop and Vertex Cover)**.** *Given a cubic bridgeless graph G, a corresponding represents table R and a vertex cover S of size $\frac{|V|}{2}+1$, S is the minimum-size vertex cover derivable from the represents table R if and only if there is no S-diminishing hop in the represents table R.*

Proof Outline: In this restricted case, the size of the minimum vertex cover can either be $\frac{|V|}{2}+1$ or $\frac{|V|}{2}$. By definition, we start with a vertex cover of size $\frac{|V|}{2}+1$. Hence, if a diminishing hop w.r.t. the given vertex cover does not exist, then it implies that the given vertex cover is the minimum vertex cover. Additionally, if there is a diminishing hop, then the resultant vertex cover is of size $\frac{|V|}{2}$, which is the minimum-size vertex cover possible in a graph having a perfect matching. On the other hand, if the given vertex cover is not a minimum vertex cover, then there exists a diminishing hop that leads to a minimum vertex cover that consists of any one of the two frozen endpoints in it.

Finally, note that the proof is designed to hold independent of the execution of the first two phases of the algorithm. More specifically, the proof holds for *any* given represents table that satisfies all its properties. Moreover, Petersen's Theorem guarantees that a locally optimal vertex cover w.r.t. the represents table is also globally optimal (Lemma 4).

Proof. We prove the contrapositive of the theorem. We start with the reverse direction, which is relatively simpler than the forward direction.

($\Leftarrow$) If there exists an S-diminishing hop, then S is not a minimum vertex cover.

Let S be a vertex cover of size $\frac{|V|}{2}+1$. Then the corresponding represents table R consists of $\frac{|V|}{2}+1$ frozen endpoints (and $\frac{|V|}{2}-1$ removed endpoints). Given that we use cubic bridgeless graphs, there is always a perfect matching, which means that there are $\frac{|V|}{2}$ rows in the represents table. This, by pigeonhole principle and by design requiring each row to have at least one frozen

[35]Given a cubic bridgeless graph, a restricted case is the case where *exactly* one row in the corresponding represents table consists of two frozen endpoints and a general case is the case where *at least* one row in the corresponding represents table consists of two frozen endpoints.

endpoint, implies that exactly one row in the represents table consists of a duad (i.e., a row with both its endpoints as frozen). Let u and v be the endpoints that form a duad.

Given that there exists an S-diminishing hop, we know that, by definition, the hop begins at the row with the duad u and v. In turn, the diminishing hop decreases the number of frozen endpoints in the table by one. The resultant frozen endpoints, one in each row, correspond to a vertex cover S'. Hence, we know that either one of u or v will be in the vertex cover S'. Formally, given that $S \cap \{u\} \neq \emptyset$ and $S \cap \{v\} \neq \emptyset$, it means that either $S' \cap \{u\} = \emptyset$ or $S' \cap \{v\} = \emptyset$. For the remaining $\frac{|V|}{2} - 1$ rows where one of its two endpoints (y and z) is frozen, we know that either $S \cap \{y\} = \emptyset$ or $S \cap \{z\} = \emptyset$; this condition holds for S' too. Here, if $y \in S$, then it does not imply $y \in S'$ (or analogously, if $z \in S$, then it does not imply $z \in S'$). We only know for a fact that either one of the two endpoints in a row will be in the vertex covers S, S'. We do not need to show *which* specific endpoint will be in the vertex covers. Overall, we are sure that $|S'| = |S| - 1$. Therefore, because we were able to find a vertex cover S' of a size smaller than the size of the vertex cover S ($|S'| < |S|$; here, specifically $|S'| = |S| - 1$), S cannot be a minimum vertex cover. In fact, for this restricted setting, we provide a stronger argument: vertex cover S' of size $\frac{|V|}{2}$ is indeed the minimum vertex cover (Lemma 1). Hence, S cannot be a minimum vertex cover. We now prove the other direction.

($\Rightarrow$) If S is not a minimum vertex cover, then there exists an S-diminishing hop.

Let S be a vertex cover that is not a minimum vertex cover. Since S is not a minimum vertex cover, there must exist at least one vertex cover S' that is smaller than S, i.e., $|S'| < |S|$. In this restricted case, we know that the vertex cover S is of size $\frac{|V|}{2} + 1$, and the vertex cover S' is of size $\frac{|V|}{2}$. We now find the association between S and S'.

Take Symmetric Difference of vertices in S and S': Let G' be a subgraph of G such that the vertices in G' is a symmetric difference between the vertex covers S and S'. Formally,

$$G' = S \Delta S' = (S \setminus S') \cup (S' \setminus S)$$

This means that the vertices in the subgraph G' are the vertices that are in the vertex cover S but not in the vertex cover S', or in S' but not in S. Vertices that are in both S and S' or that are neither in S nor in S' are omitted. The edges in subgraph G' correspond to the edges in the graph G that connect the vertices in G that correspond to the vertices in G'.

Properties of the subgraph G':

1. The edges not in G' but in G are covered by the vertices that are in both the vertex covers S and S'. Hence, the edges in G' are the edges that remain uncovered when the vertices only in S' or only in S are removed.

2. Each edge in the subgraph G' is covered by vertices that correspond to the vertices only in the vertex cover S. Simultaneously, each edge in the subgraph G' is covered by vertices that correspond to the vertices only in the vertex cover S'.

3. The number of vertices in the subgraph G' is odd. Specifically, suppose there are m' vertices in the subgraph G' that correspond to the vertices in the vertex cover S. In that case, there are $m' - 1$ vertices in the subgraph G' that correspond to the vertices in the vertex cover S'.

4. The subgraph G' can be a single vertex (singleton), a linear chain, a cycle, or a tree (or multiple connected components of one or more of the four).

Relating S and S' to subgraph G': The vertices in G' alternate between a vertex in S and a vertex in S'. This is because for every edge $e = (u, v)$ in subgraph G', if the edge is covered by vertex u in S, then it is covered by vertex v in S', or vice versa. Vertices u and v cannot be in the same vertex cover together. Otherwise, both the vertices would not be in the symmetric difference of S and S', and consequently, $e = (u, v)$ would not be an edge in the subgraph G'. Additionally, it is not possible that a vertex cover S (or S') does not contain either of the vertices u and v because if that is the case then S (or S') will not be a vertex cover as edge $e = (u, v)$ will not be covered.

Identifying a Diminishing Hop from the subgraph G': This is a critical part. Until now, the proof in the forward direction has only discussed graphs and not mentioned the represents table, which is needed for diminishing hops. Hence, we first associate the vertex covers S and S' for the given graph G with the represents tables R and R', respectively, that correspond to the given graph G. More specifically, given a graph G and a vertex cover S, there is a represents table R such that endpoints that correspond to vertices in the vertex cover S are frozen and the remainder are removed. Similarly, given a graph G and a vertex cover S', there is a represents table R' such that endpoints that correspond to vertices in the vertex cover S' are frozen and the remainder are removed. Next, in this restricted case, we know that one row of the represents table R consists of a duad of frozen endpoints. Hence, remove the endpoint from the row with a duad such that the removed endpoint is in the subgraph G'. The other endpoint in the duad will be frozen and must be in both the vertex covers S and S', and hence, not in the subgraph G'. Then, follow the sequence of remove and freeze operations, i.e., the duadic hop w.r.t. S. Consequently, the table R will become the same as table R' such that the endpoints removed from R during the duadic hop correspond to the vertices in G' that are from the vertex cover S, and the endpoints frozen in R during the duadic hop correspond to the vertices in G' that are from the vertex cover S'. Hence, after the duadic hop w.r.t. S, table R becomes equivalent to table R'. Finally, given that duadic hop w.r.t. S in the represents table R leads to a smaller number of frozen endpoints, and in turn, smaller vertex cover, it implies that the S-duadic hop is, by definition, an S-diminishing hop. Overall, because S is not a minimum vertex cover, the given table R has an S-diminishing hop.

In summary, the alternating vertices in the subgraph G' (please refer to "Relating S and S' to subgraph G'") from the vertex cover S and S' correspond to the removed and frozen endpoints in the table R, respectively. Hence, the vertices in subgraph G' correspond to the sequence of endpoints that are removed and frozen, i.e., an S-duadic hop. The duadic hop is an S-diminishing hop. Therefore, if S is not a minimum vertex cover, then there exists an S-diminishing hop.

This completes the other direction of the proof of correctness. In turn, it completes the overall contrapositive proof that shows that S is the minimum-size vertex cover derivable from the represents table R if and only if there is no S-diminishing hop in the represents table R. ☐

We established the relation between a vertex cover S and an S-diminishing hop in the represents table R for the restricted case where the represents table R has *exactly* one duad, i.e., *exactly* one row with two frozen endpoints. We now generalize this result to the case where the represents table R has *at least* one duad, i.e., *at least* one row with two frozen endpoints. As for the implementation, the steps we mentioned (starting at the end of page 39) remain the same. However, when more than one row has a duad, Step 1 is repeated for each of the existing duads unless a duadic hop beginning at each duad is guaranteed to not be a diminishing hop. The formalization on how to ensure that no diminishing hop remains and its corresponding time complexity is discussed in Section 6 (Algorithm) and Section 8 (Time Complexity), respectively. Here, we continue to focus on establishing the relation between a minimum vertex cover S and S-diminishing hop.

Theorem 7 (Diminishing Hop and Vertex Cover)**.** *Given a cubic bridgeless graph G, a corresponding represents table R and a vertex cover S, S is the minimum-size vertex cover derivable from the represents table R if and only if there is no S-diminishing hop in the represents table R.*

Proof. Before we begin the discussion of the proof, we share two observations:

Observation 1. *The size of the minimum vertex cover S will be between $\frac{|V|}{2}$ and $|V|-1$, i.e., $|S| \in [\frac{|V|}{2}, |V|-1]$*[36][37].

More specifically, when $|S| = \frac{|V|}{2}$, there are two facts: (i) S is the minimum vertex cover because the size of perfect matching is $\frac{|V|}{2}$ (Lemma 1), and (ii) there is no duad, and in turn, there will be no S-diminishing hop. Conversely, when there is no duad in a represents table R, and in turn, there is no S-diminishing hop, it means that exactly $\frac{|V|}{2}$ endpoints are frozen (one endpoint frozen for each row), which implies that S is the minimum vertex cover.

Observation 2. *When the given graph is a cubic bridgeless graph G, the minimum-size (smallest) vertex cover derivable from the represents table R implies a minimum vertex cover.*

We now discuss the main proof. We again prove the contrapositive of the theorem. This proof has subtle variations from the proof for Theorem 6, which necessitates a detailed discussion. We start with the reverse direction, which is relatively simpler than the forward direction. Also, recall that m denotes the number of vertices in the given graph ($m = |V|$).

($\Leftarrow$) If there exists an S-diminishing hop, then S is not a minimum vertex cover.

Let S be a vertex cover of size $\frac{m}{2}+1 \le |S| \le m-1$. Then the corresponding represents table R consists of x frozen endpoints (and remainder $m-x$ removed endpoints) where x is an integer such that $x \in [\frac{m}{2}+1, m-1]$. Given that we use cubic bridgeless graphs, there is always a perfect matching, which means that there are $\frac{m}{2}$ rows in the represents table. This, by pigeonhole principle and by design requiring each row to have at least one frozen endpoint, implies that at least one row in the represents table R consists of a duad (i.e., a row with both its endpoints as frozen).

Let u and v be the endpoints that form a duad. Given that there exists an S-diminishing hop, we know that, by definition, the hop begins at the row with a duad, say, row i containing endpoints u and v. In turn, the diminishing hop decreases the number of frozen endpoints in the table by at least one because one of the endpoints from u or v will be removed. The resultant frozen endpoints in table R' correspond to a vertex cover S'. Hence, we know that either one of u or v will be in the vertex cover S'. Formally, given that $S \cap \{u\} \neq \emptyset$ and $S \cap \{v\} \neq \emptyset$, it means that either $S' \cap \{u\} = \emptyset$ or $S' \cap \{v\} = \emptyset$. For the remaining $\frac{m}{2}-1$ rows, we do not need to show *which* specific endpoints will be in each of the vertex covers S and S'. It suffices to show that, by definition, the total number of frozen endpoints in the remaining $\frac{m}{2}-1$ rows of table R will be at least equal to the total number of frozen endpoints in the remaining $\frac{m}{2}-1$ rows of table R'. Hence, $|S \setminus \{u,v\}| \ge |S' \setminus \{u\}|$ or $|S \setminus \{u,v\}| \ge |S' \setminus \{v\}|$. Consequently, $|S| > |S'|$. Therefore, because we were able to find a vertex cover S' of a size smaller than the size of the vertex cover S ($|S'| < |S|$), S cannot be a minimum vertex cover. We now prove the other direction.

($\Rightarrow$) If S is not a minimum vertex cover, then there exists an S-diminishing hop.

[36]The upper bound of $|V|-1$ on the size of the minimum vertex cover is a relaxed one. We can prove a tighter upper bound that can be provided by the use of representation scores. We omit a detailed analysis as it is not within the scope of this paper.

[37]A closed interval $[a,b]$ denotes all integers in the range of integers a and b, both inclusive. Formally, $[a,b]$ denotes all integers x such that $a \le x \le b$.

Let S be a vertex cover that is not a minimum vertex cover. Since S is not a minimum vertex cover, there must exist at least one vertex cover S' that is smaller than S, i.e., $|S'| < |S|$. We now find the association between S and S'.

Take Symmetric Difference of vertices in S and S': Let G' be a subgraph of G such that the vertices in G' is a symmetric difference between the vertex covers S and S'. Formally,

$$G' = S \Delta S' = (S \setminus S') \cup (S' \setminus S)$$

The edges in subgraph G' correspond to the edges in the graph G that connect the vertices in G that correspond to the vertices in G'.

Properties of the subgraph G':

1. The edges not in G' but in G are covered by the vertices that are in both the vertex covers S and S'. Hence, the edges in G' are the edges that remain uncovered when the vertices only in S' or only in S are removed.

2. Each edge in the subgraph G' is covered by vertices that correspond to the vertices only in the vertex cover S. Simultaneously, each edge in the subgraph G' is covered by vertices that correspond to the vertices only in the vertex cover S'.

3. The subgraph G' can be a single vertex (singleton), a linear chain, a cycle, or a tree (or multiple connected components of one or more of the four, or a bipartite graph).

Relating S and S' to G': If the graph G' is a singleton or consists of a singleton component, then that single vertex must come from the vertex cover S. For the remaining cases, the vertices in (each connected component of) G' alternate between a vertex in S and a vertex in S'. In other words, G' is a bipartite graph. This is because for every edge $e = (u, v)$ in subgraph G', if the edge is covered by vertex u in S, then it is covered by vertex v in S', or vice versa. Vertices u and v cannot be in the same vertex cover together. Additionally, a vertex cover S (or S') must contain either of the vertices u and v.

Identifying a Diminishing Hop from the subgraph G': Until now, this proof in the forward direction discussed graphs and did not mention the represents table, which is needed for diminishing hops. Hence, we first associate the vertex covers S and S' for the given graph G with the represents tables R and R', respectively, that correspond to the given graph G. More specifically, given a graph G and a vertex cover S, there is a represents table R such that endpoints that correspond to vertices in the vertex cover S are frozen and the remainder are removed. Similarly, there is a represents table R' for a given graph G and a vertex cover S'. This mapping from a graph to a represents table is general because:

- how a represents table is populated is not used in the proof. Hence, the logic holds for any represents table that can be derived in the preceding two phases. Hence, the one represents table that will be derived by the algorithm is inherently covered by our proof.

- additionally, just like an adjacency list is a form of a direct and complete representation of the underlying graph's structure, a represents table is also a direct and complete representation of the underlying graph. Hence, operations on represents table are considered equivalent to operations on the corresponding graph. More specifically, every operation performed on the represents table has a one-to-one mapping to a conceptually equivalent operation on the graph itself. Moreover, even if we were to consider the latter two representations to be different, a represents table is implicitly isomorphic to the graph because of a bijection that preserves the operations carried out. Finally, if we were working on an online graph problem, then the represents table would not have been isomorphic, which is not the case here[38].

- given that the represents table enlists each vertex of the graph G as an endpoint (Lemma 4), this discussion is globalized, in line with Berge's Theorem [Ber57]. Specifically, Berge showed that checking the existence of a localized structure, namely augmenting path, guarantees a global maximum matching. Similarly, because each endpoint is in the represents table, checking the existence of a localized structure, namely diminishing hop, guarantees a global minimum vertex cover.

Next, we discuss the existence of an S-diminishing hop in each of the four types of graph G':

- **Singleton:** A singleton in graph G' consists of a vertex u from the vertex cover S. Because there is no edge connected to this single vertex in G', removing u from S keeps the vertex cover intact and results in a smaller vertex cover S'. Correspondingly, there exists an S-diminishing hop that removes the endpoint u from the duad in the represents table R, which results in a represents table R' with fewer frozen endpoints that correspond to the vertex cover S'.

For the remaining three graph types, each is a special case of a bipartite graph. By design, each component of the graph G' must be a bipartite graph because each edge in G' connects a vertex from S and a vertex from S'. We now make the following common observation (that holds for a general bipartite graph): we know that at least one row of the represents table R consists of a duad of frozen endpoints. Hence, to perform an S-diminishing hop, remove the endpoint from a row with a duad such that (i) the removed endpoint is in the subgraph G' and (ii) the other endpoint in the duad is frozen and must be in both the vertex covers S and S', and hence, not in the subgraph G'. Then, follow the sequence of remove and freeze operations, i.e., the duadic hop w.r.t. S. For a singleton, we discussed that the hopping stops after the removal of one endpoint. We now discuss the cases for the remaining graph types:

- **Even Cycle:** When there is an even cycle consisting of c vertices, then an S-diminishing hop cannot be carried on the even cycle. The cycle consists of $\frac{c}{2}$ vertices each from the vertex covers S and S'. Hence, a hop simply changes the vertices but does not affect the count. Additionally, an even cycle can only exist with another component in the graph G'. Without another component, the existence of only an even cycle implies that the vertex covers S and S' are of the same size, which contradicts our assumption. Hence, another component in G' must exist.

- **Odd Cycle:** There is never an odd cycle in graph G'. Assume that there is an odd cycle in G'. This implies that the cycle consists of at least one vertex more from the vertex cover S than from the vertex cover S'. In turn, this means that there exists an edge that connects two vertices from the same vertex cover S (or S'). However, this is not possible because G' is a symmetric difference of vertices in S and S', and hence, there can be no edge in G' that connects two vertices from S (or S'). This is because if such an edge exists, then S' (or S)

[38] Just like the sequence in which we draw the vertices of the graph does not matter when thinking about selecting or not selecting vertices that are in a minimum vertex cover, the sequence in which the represents table gets populated does not matter, and the operations carried out for a duadic hop on the represents table are not affected.

cannot be a vertex cover. This contradicts our assumption about the presence of an odd cycle in G'.

- **Even Linear Chain:** A linear chain of even length cannot exist by itself without another component in G' for reasons that are the same as even cycles. Hence, we focus our discussion on linear chains of odd length.

- **Odd Linear Chain:** When the graph G' is (has) a linear chain of odd length, the linear chain consists of at least one vertex more from the vertex cover S than from the vertex cover S'. Hence, an S-diminishing hop can begin at any row with a duad in the represents table R such that one of the endpoints in the duad corresponds to a vertex from S in the linear chain of G'. Subsequently, the table R will become the same as table R' such that the endpoints removed from R during the S-diminishing hop correspond to the vertices in G' that are from the vertex cover S, and the endpoints frozen in R correspond to the vertices from the vertex cover S'. Finally, the S-diminishing hop in the represents table R leads to a smaller number of frozen endpoints, and in turn, a smaller vertex cover. Therefore, because S is not a minimum vertex cover, the given table R has an S-diminishing hop.

  Here, we note that such a mapping between the graph structure (G') and the table operations on R (hopping) exist because, by design and due to table properties, each vertex removed from G' correspond to a removal of an endpoint in the table, which in turn implies that each endpoint that represents and is represented by the removed endpoint will be frozen. In turn, this implies that at least one of the vertices in G' will be frozen too that correspond to the vertex from vertex cover S'. Overall, the aim is not to find an odd linear chain in the represents table. Rather we simply needed to prove that when such an odd linear chain exists in graph G', there exists at least one duadic hop in represents table R, independent of how the table is derived, such that it mimics the odd linear chain in graph G' through its sequence of remove and freeze operations, which correspond to removing the vertices from vertex cover S and freezing the vertices from vertex cover S' in graph G', respectively. This discussion of mapping between the graph structure (G') and the table operations (hopping) is generalizable and holds for each graph structure as discussed.

- **Even Tree:** When G' is (has) a tree, it must be a binary tree because it is derived from a cubic graph G. When the length of the longest path between each pair of leaf vertices is even, it results in the same case as an even linear chain, and there is no change in the number of vertices in S and S'.

- **Odd Tree:** When there is an odd path, an S-diminishing hop begins at a row with a duad in the table R such that one endpoint in the duad corresponds to a vertex from S in G'. Subsequently, the table R becomes equivalent to R' such that the endpoints removed from R correspond to the vertices in G' from the vertex cover S, and the endpoints frozen correspond to the vertices in G' from the vertex cover S'. More generally, because G' *must* be bipartite, this discussion (regarding an S-diminishing hop beginning at a row with a duad in the R) holds for the general case when G' is a bipartite graph.

In summary, for each graph type, the alternating vertices in the subgraph G' from the vertex cover S and S' correspond to the removed and frozen endpoints in the table R, respectively. Hence, the vertices in subgraph G' correspond to the sequence of endpoints that are removed and frozen, i.e., an S-duadic hop. The duadic hop is an S-diminishing hop because the number of endpoints removed is greater than the number of endpoints frozen (across all components when even components are present). Therefore, when S is not a minimum vertex cover, there exists an S-diminishing hop.

This completes the other direction of the proof of correctness. In turn, it completes the overall contrapositive proof that shows that S is the minimum-size vertex cover derivable from the represents table R if and only if there is no S-diminishing hop in the represents table R. □

Theorem 7 is designed to hold for a cubic bridgeless graph G, which always consists of a perfect matching (Theorem 3). However, if we are not given a cubic bridgeless graph, then the graph may not consist of a perfect matching. Consequently, at least one vertex won't be listed as an endpoint in the represents table R. Hence, the theorem that guarantees a minimum vertex cover does not hold anymore. However, by design, we can generalize this result (for future use) by stating a new corollary, which states that the vertex cover derived from the represents table R is the minimum-size derivable from the endpoints listed in the represents table R. In other words, the vertex cover is the minimum-size derivable from subset of vertices that are the endpoints of the edges in the maximum matching found using the Blossom Algorithm during Phase I of the algorithm.

Corollary 1. *Given a graph G, a corresponding represents table R and a vertex cover S (derived using R), S is the minimum-size vertex cover derivable from the endpoints in represents table R if and only if there is no S-diminishing hop in the represents table R.*

Proof. A cubic bridgeless graph always consists of a perfect matching (Theorem 3). Hence, all the vertices of a given graph are always listed as endpoints in the represents table R. Therefore, Theorem 7 guaranteed a minimum vertex cover. In other words, by design, an S-diminishing hop finds the smallest vertex cover derivable from the endpoints in the represents table R. This implies a minimum vertex cover when there exists a perfect matching, such as in a cubic bridgeless graph. Therefore, when an arbitrary graph is given, this "generalized" corollary follows from Theorem 7. □

Finally, we stress that while Theorem 7 proves that a vertex cover S is a minimum vertex cover if and only if there is no S-diminishing hop, the discussion on (i) performing a diminishing hop and (ii) bounding the number of duadic hops needed to ensure there remains no diminishing hops is done later (Lemma 7 and Lemma 8)[39].

5.4 Summary

We provided an overview of each of the three phases of the algorithm. We also introduced a data structure "represents table" and a technique called "diminishing hop". Consequently, we presented a few intermediate results needed for the proof of correctness of the algorithm. Overall, the algorithm we discovered is a three-phase algorithm, which, when given a cubic bridgeless graph G and a non-negative integer k, returns "**Yes**" if there is a vertex cover S of size at most k, and "**No**" otherwise. The three phases are implemented sequentially (Figure 8):

I **Find a Perfect Matching:** Use the Blossom Algorithm to find a perfect matching.

II **Populate Represents Table:** Create a BFS tree and use it with the perfect matching to implement an augmented version of the 2-approximation algorithm for the vertex cover problem

[39] Berge's Theorem established the relation between augmenting paths and maximum matching. Subsequently, the Blossom Algorithm used this theorem to find a maximum matching by searching for an augmenting path. Similarly, in our paper, Theorem 7 simply establishes a relation between diminishing hops and minimum vertex cover. Subsequently, Section 6 uses this theorem to find a minimum vertex cover by searching for a diminishing hop.

to populate a novel data structure called the "represents table". We also discussed operations and properties of the represents table.

III **Diminishing Hops:** The input to the third phase is the data structure represents table. Foremost, assign a weighted number to each vertex (also known as an endpoint) in the table, called the representation score ζ, which captures how well an endpoint is represented in the table. Next, use the representation score ζ to freeze or remove each endpoint in the represents table. The frozen endpoints correspond to a vertex cover. Finally, motivated by the use of augmenting paths (a specific case of an alternating path) w.r.t. a given matching to find a maximum matching, the third phase introduces diminishing hops (a specific case of a duadic hop) w.r.t. a given vertex cover to find a minimum vertex cover. A diminishing hop differs from the Vertex Cover Reconfiguration problem because we do not (i) stipulate that each remove / freeze operation of a hop should result in a vertex cover or (ii) bound the number of operations needed to complete the hopping.

Overall, the combination of all these phases implies that we get an unconditional deterministic polynomial-time algorithm for the vertex cover problem on cubic bridgeless graphs.

6 Algorithm

We now present the core contribution of this paper, an algorithm to solve the $\texttt{VC} - \texttt{CBG}$ problem. In the algorithm, all ties are broken and all ordering (sorting) of vertices is done based on lexicographic ordering unless noted otherwise. The ordering does not impact the correctness but ensures that for the same input, the output remains the same.

Algorithm 1: `VERTEX_COVER`(G, k)

Data: Cubic Bridgeless Graph $G = (V, E)$
non-negative integer k
Result: returns **Yes** if there is a Vertex Cover S of size at most k, **No** otherwise

V_s = lexicographically sorted set of vertices
`// PHASE I`
M = a set of edges in a perfect matching found using the Blossom Algorithm [Edm65]
if $k < |M|$ **then**
    return **No**
end
`// PHASE II`
R = `POPULATE_REPRESENTS_TABLE`(G, M, V_s) `// Algorithm 2`
`// PHASE III`
S = `DIMINISHING_HOP_PHASE`(R) `// Algorithm 3`
if $|S| \leq k$ **then**
    return **Yes**
end
return **No**

Algorithm 2: POPULATE_REPRESENTS_TABLE(G, M, V_S)

Data: Cubic Bridgeless Graph $G = (V, E)$
Edges in Perfect Matching M
Lexicographically Sorted Vertices V_S

Result: returns Represents Table R

T = an array of arrays storing sorted vertices at each level of a breadth-first search tree seeded on the first vertex in V_S, and each vertex in T is marked unvisited `// Table 1`

R = a four-column table, Represents Table, that stores the endpoints of an edge selected during the for loop discussed below and the corresponding vertices each endpoint is connected to through an edge `// Definition 14, Table 2`

`// The following loop traverses the BFS-tree table top-down`

for each *level* in T **do**
- **for each** unvisited vertex u in *level* **do**
  - **if** *there exists an edge that connects vertex u with another vertex on the same level and the edge is in M* **then**
    - select the edge
  - **else if** *there exists an edge that connects vertex u with another vertex on the next level and the edge is in M* **then**
    - select the edge
  - **end**
  - Mark the two endpoints of the selected edge as visited in T
  - Insert a new row after the last row in R: the two endpoints of the selected edge and the respective vertices each endpoint is connected to through an edge
  - Remove from graph G the selected edge and all the edges that are connected to the two endpoints
  - If any vertex becomes edgeless in G, mark the vertex as visited in T
- **end**

end

return R

Algorithm 3: DIMINISHING_HOP_PHASE(R)

Data: Represents Table R

Result: returns a Minimum Vertex Cover S

$S = \emptyset$

Augment the represents table R with two new columns corresponding to the two endpoints in each row

For each endpoint u in the represents table R, insert into the new columns of the table a representation score ζ such that $\zeta_u = -\infty$

R = COMPUTE_REPRESENTATION_SCORE(R) `// Algorithm 4`

R, S = VERTEX_ELIMINATION(R, S) `// Algorithm 5`

for each integer a in $[1, \frac{m}{2}]$ **do**
- R, S = DIMINISHING_HOPS(R, S) `// Algorithm 7`

end

return S

Algorithm 4: `COMPUTE_REPRESENTATION_SCORE`(R)

Data: Represents Table R

Result: returns Represents Table R with updated representation scores ζ

```
// The following loop traverses through the table R top-down
for each row in R do
    // The following loop executes exactly twice
    for each endpoint u in row do
        if u is frozen then
            ζ_u = -1
            continue
        else if u is removed then
            ζ_u = -1
            continue
        else
            ζ_u = 0
            for each row_j in R that is above row do
                x, y = two endpoints in row_j
                // L_x denotes the list of endpoints that the endpoint x in row_j represents
                if vertex u ∈ L_x then
                    ζ_u = ζ_u + max(0, ζ_y) + 1
                else if vertex u ∈ L_y then
                    ζ_u = ζ_u + max(0, ζ_x) + 1
                else
                    do nothing
                end
            end
        end
    end
end
return R
```

Algorithm 5: `VERTEX_ELIMINATION`(R, S)

Data: Represents Table R
Vertex Cover S

Result: returns updated Represents Table R, Vertex Cover S

```
// The following loop traverses through the table R bottom-up
for each row in R do
    R = COMPUTE_REPRESENTATION_SCORE(R) // Algorithm 4
    if both endpoints in row are either frozen or removed then
        continue
    else if endpoint u in row remains and endpoint v in row is frozen then
        R, S = FREEZE_AND_REMOVE(R, S, ∅, u) // Algorithm 6.  ∅ denotes a null value
    else
        // at this point, both endpoints u and v in row are neither frozen nor removed, and
           represent exactly one endpoint, namely each other
        if ζ_u ≥ ζ_v then
            R, S = FREEZE_AND_REMOVE(R, S, u, v) // Algorithm 6
        else
            R, S = FREEZE_AND_REMOVE(R, S, v, u) // Algorithm 6
        end
    end
end
return R, S
```

Algorithm 6: `FREEZE_AND_REMOVE`(R, S, ψ, ω)

Data: Represents Table R
Vertex Cover S
Endpoint to be Frozen ψ
Endpoint to be Removed ω

Result: returns updated Represents Table R, Vertex Cover S

```
// Freeze Operation of Represents Table
Freeze endpoint ψ in R
Append endpoint ψ to S
Set the represents list L_ψ of endpoint ψ in R to null
Delist endpoint ψ from every represents list in R
// Remove Operation of Represents Table
Remove endpoint ω from R
Remove endpoint ω from S (if present)
for each non-frozen and unremoved endpoint u in R such that ω ∈ L_u do
    R, S = FREEZE_AND_REMOVE(R, S, u, ∅)
end
for each non-frozen and unremoved endpoint u in L_ω do
    R, S = FREEZE_AND_REMOVE(R, S, u, ∅)
end
Set the represents list L_ω of endpoint ω in R to null
return R, S
```

Algorithm 7: DIMINISHING_HOPS(R, S)

Data: Represents Table R
Vertex Cover S

Result: returns a diminished or the same Represents Table R and
a smaller or same Vertex Cover S

```
λ = ∅ // an array of endpoints (vertices) visited during diminishing hops
R_diminished=R
S_diminished=S
// The loop traverses through the table R top-down
for each row in R do
    R_original=R
    S_original=S
    λ_original=λ
    // a duadic hop exists only if both endpoints u and v in row are frozen, and form a
       duad
    if both endpoints in row are frozen then
        for each endpoint u in row do
            R, S, λ = DUADIC_HOP(R, S, ∅, u, λ) // Algorithm 8
            // this holds if the number of vertices removed > number of vertices frozen
            if |S| < |S_diminished| then
                R_diminished=R
                S_diminished=S
                λ_diminished=λ
            end
            R=R_original
            S=S_original
            λ=λ_original
        end
        R=R_diminished
        S=S_diminished
        λ=λ_diminished
    end
end
return R, S
```

Algorithm 8: `DUADIC_HOP`(R, S, ψ, ω, λ)

Data: Represents Table R, Vertex Cover S
Endpoint to be Frozen ψ, Endpoint to be Removed ω, List of Visited Endpoints λ

Result: returns updated Represents Table R, Vertex Cover S, Visited Endpoint List λ

```
// During each execution of this algorithm, either ψ = ∅ or ω = ∅
if ω ≠ ∅ then
    if ω ∈ λ then
        return R, S, λ
    end
    Add ω to λ
    Remove endpoint ω from R
    Remove endpoint ω from S
    Q = ∅ // a queue storing the endpoints to be frozen
    for each endpoint u in L_ω do
        if u ∉ λ then
            Add u to λ
            if u ∉ S then
                Q = Q ∪ {u} // enqueue endpoint u
            end
        end
    end
    for each endpoint u in R such that ω ∈ L_u do
        if u ∉ λ then
            Add u to λ
            if u ∉ S then
                Q = Q ∪ {u}
            end
        end
    end
    for each endpoint u in Q do
        Q = Q \ {u} // dequeue endpoint u
        R, S, λ = DUADIC_HOP(R, S, u, ∅, λ)
    end
end
// the following condition will be true only when an endpoint has been removed
if ψ ≠ ∅ then
    Freeze endpoint ψ in R
    Append endpoint ψ to S
    // the following condition is equivalent to checking for the existence of a duad
    u=the other endpoint that is in the same row of R as ψ
    if u ∈ S then
        R, S, λ = DUADIC_HOP(R, S, ∅, u, λ)
    end
end
return R, S, λ
```

7 Proof of Correctness

We proved the relation between diminishing hops and minimum vertex cover in Section 5. Hence, showing that the algorithm successfully searches for diminishing hops, which, combined with already proven results, implies the correctness of the algorithm. Overall, we prove the following theorem:

Theorem 8. *Algorithm 1 returns* ***Yes*** *if and only if the given instance of* $\texttt{VC} - \texttt{CBG}$ *is a* ***Yes*** *instance.*

More specifically, we prove the theorem through a sequence of lemmas. Foremost, in the reverse direction, we have the following lemma:

Lemma 5. *If the given instance of* $\texttt{VC} - \texttt{CBG}$ *is a* ***Yes*** *instance, then the Algorithm 1 returns* ***Yes****.*

Proof. When the given instance of $\texttt{VC} - \texttt{CBG}$ is a **Yes** instance, it implies that there is a vertex cover S of size at most k ($|S| \leq k$). Additionally, it implies that $k \geq \frac{m}{2}$. This is because a cubic bridgeless graph always consists of a perfect matching (Theorem 3), which means the size of a maximum matching M (equivalently, a perfect matching for our paper) is $\frac{m}{2}$. Therefore, by Lemma 1, we know that the size of a minimum vertex cover S' is $|S'| \geq |M|$, which means $|S'| \geq \frac{m}{2}$. Hence, each vertex cover S in a set of vertex covers $\mathcal{S}$, $S \in \mathcal{S}$, will be of size $|S| \geq \frac{m}{2}$. Consequently, because

$$k \geq |S| \text{ and } |S| \geq \frac{m}{2}$$

we know that

$$k \geq \frac{m}{2}$$

Therefore, Line 5 of Algorithm 1 cannot return **No**.

Next, by design, we know that Line 5 of Algorithm 3 consists of a vertex cover S. This is because the operations of the represents table R are designed to ensure that the frozen endpoints in table R correspond to a vertex cover (Theorem 4). Finally, at the end of the execution of the loop in Line 6 of Algorithm 3, the final vertex cover S in Line 7 of Algorithm 3 consists of a minimum vertex cover because there is no S-diminishing hop in the given represents table (Theorem 7). This will be returned by Line 9 of Algorithm 3. This implies that no vertex smaller than S can exist. Therefore, if the given instance of $\texttt{VC} - \texttt{CBG}$ is a **Yes** instance, then each vertex cover $S' \in \mathcal{S}$ that can be a **Yes** instance must be of size greater than or equal to the minimum vertex cover and less than or equal to k. Formally, $|S| \leq |S'| \leq k$. Hence, the condition in Line 11 of Algorithm 1 must be true ($|S| \leq k$), which means Line 12 of Algorithm 1 must return **Yes**. The execution of Algorithm 1 will never reach Line 14, and hence, it cannot return **No**. □

Next, in the forward direction, we have the following lemma:

Lemma 6. *If the Algorithm 1 returns* ***Yes****, then the given instance of* $\texttt{VC} - \texttt{CBG}$ *is a* ***Yes*** *instance.*

Proof. If the Algorithm 1 returns **Yes**, then it can do so only if Line 12 of Algorithm 1 returns **Yes**. This implies that Line 5 of Algorithm 1 cannot return **No**. Hence, the value of non-negative integer k must be greater than the size of the perfect matching M found in Line 3 of Algorithm 1; formally, $k \geq \frac{m}{2}$. This also implies that Lines 8 and 10 of Algorithm 1 must be executed, which are the two main phases of the algorithm. We first discuss the execution of Line 8.

Line 8 of Algorithm 1 (Populate Represents Table): The Line 8 of Algorithm 1 invokes Algorithm 2. The output of Algorithm 2 is a data structure called the represents table R. Line 12 of Algorithm 2 inserts a new row to the table such that the endpoints of an edge selected in Lines 7 or 9 are listed. Because the algorithm only selects the edges that are in the perfect matching M (Lines 6 or 8), it implies that the algorithm enlists endpoints of edges in a matching, which means the endpoints form a vertex cover (Lemma 3). More specifically, in our case, the matching is a perfect matching, and each perfect matching is a maximum matching, which in turn is a maximal matching (Lemma 2). Hence, each vertex of graph G is listed as an endpoint in table R, which trivially forms a vertex cover (Property 3). The Lines 1, 13, and 14 of Algorithm 2 are important for the next phase as they ensure the table R has certain properties (Property 1, Property 2, Property 4). The key output of Algorithm 2 is that the endpoints of the resultant represents table R form a vertex cover.

Line 10 of Algorithm 1 (Diminishing Hop Phase): The Line 10 of Algorithm 1 invokes Algorithm 3 with the represents table R as its input. The first three lines of Algorithm 3 are initialization steps that are consequential for the next lines. Hence, we do not discuss them. Line 4 of Algorithm 3 invokes Algorithm 4 (Compute Representation Score). Algorithm 4 assigns a score to each endpoint. It does not alter the structure of the table or its endpoints and hence, it does not need further discussion. Next, Line 5 of Algorithm 3 invokes Algorithm 5.

- **Algorithm 5 (Vertex Elimination)**: The overarching goal of this algorithm is to freeze or remove each endpoint in the represents table R using the representation score such that the frozen endpoints correspond to a vertex cover S such that $S \subset V$ (Property 4, Theorem 4)[40].

  More specifically, Algorithm 5 traverses through the represents table R bottom-up. For each row, it recomputes the representation score by invoking Algorithm 4. Then, it carries out a sequence of freeze and remove operations while adhering to the properties of the represents table R. If both endpoints in a row of the table are frozen, then the algorithm does nothing. By design, both the endpoints in a row cannot be removed. If one of the endpoints in a row is frozen (and the other is neither frozen nor removed), then the other is removed (Line 7 of Algorithm 5 invokes Algorithm 6 (Freeze and Remove)). Next, if neither of the endpoints in a row is frozen or removed, one endpoint is frozen and the other one is removed based on the representation score (Line 11 or 13 of Algorithm 5 invokes Algorithm 6 as appropriate). Finally, the case when one endpoint is removed and the other is neither frozen nor removed cannot happen (by design). Hence, Algorithm 5 does not need to cover that case.

  Algorithm 6 is specifically designed to freeze or remove an endpoint. When an endpoint ψ needs to be frozen, Algorithm 6 freezes the endpoint in table R (Line 2), adds it to the vertex cover S (Line 3), and updates the corresponding represents lists (Lines 4 and 5). Lines 4 and 5 set the represents list of endpoint ψ to null and remove the endpoint ψ from the represents lists of other endpoints, respectively. This operation denotes that the vertex ψ in the vertex cover S covers each edge that connects ψ to its neighbors. Next, when an endpoint ω needs to be removed, Algorithm 6 removes the endpoint from table R (Line 7), removes it from the vertex cover S (if present; Line 8), and freezes each endpoint it represents or it is represented by (Lines 9 to 14). The last set of operations denotes that the vertex ω is not in the vertex cover S, and hence, each of its neighbors must be in S to cover each edge connected to ω.

[40] We leave the following discussion to future work: (i) how is the representation score used to decide whether to freeze or remove an endpoint and (ii) the guarantees on the upper bound of the number of endpoints being frozen. In particular, guarantees on the upper bound can be used to make the diminishing hops phase more efficient.

In summary, Algorithm 5, through Algorithm 6, carries out a deterministic sequence of freeze and remove operations such that each endpoint in the represents table R is either frozen or removed. Consequently, the frozen endpoints correspond to the vertices in a vertex cover S.

Finally, Line 7 of Algorithm 3 invokes Algorithm 7 for a total of $\frac{m}{2}$ times. Each iteration of an S-diminishing hop corresponds to one row of the represents table R.

- **Algorithm 7 (Diminishing Hops)**: There are three aspects to be proven for Algorithm 7: (i) establish the relation between a vertex cover S and an S-diminishing hop, (ii) prove that the algorithm performs an S-diminishing hop when one exists, and (iii) $\frac{m}{2}$ calls to the algorithm ensures that there exists is no S-diminishing hop when it terminates. Theorem 7 already established that a vertex cover S is minimum if and only if there is no S-diminishing hop. Hence, it remains to be discussed that Algorithm 7 is an algorithm that uses S-duadic hops to search and perform an S-diminishing hop in the represents table R. Hence, when an S-diminishing hop does not exist, S is a minimum vertex cover.

Lemma 7. *Given a represents table R where each endpoint is either frozen or removed and a vertex cover S that corresponds to the frozen endpoints in the table R, Algorithm 7 is an algorithm to perform an S-diminishing hop if it exists.*

Proof. When Line 7 of Algorithm 3 invokes Algorithm 7, input to Algorithm 7 is a represents table R[41] where each endpoint is either frozen or removed and a vertex cover S[42]. Note that during each iteration, the updated values of represents table R and the vertex cover S are passed. Line 1 of Algorithm 7 initializes an empty list that will store each endpoint that is visited during a hop. Lines 2 and 3 keep a record of the represents table R with the smallest number of frozen endpoints and of the smallest vertex cover S, respectively, during a given iteration. Line 5 ensures a top-down row-wise traversal of the represents table R. Lines 6 to 8 store the concerned data as it was when an iteration of the loop begins. This is needed because an S-duadic hop removes each of the two endpoints turn-wise to assess which one leads to an S-diminishing hop. Line 10 ensures the presence of a duad without which an S-duadic hop is not needed. Consequently, Line 11 does an S-duadic hop for each of the endpoints in a duad. Line 12 invokes Algorithm 8 (Duadic Hop). Line 14 assesses whether the vertex cover returned by Algorithm 8 (Duadic Hop) is smaller than the smallest one, and if so, updates the relevant variables (Lines 15-17). Lines 19 to 21 restore the relevant variables for the other endpoint of the duad to undergo a duadic hop. Lines 23 to 25 ensure that after each endpoint of the duad is traversed, the smallest vertex cover is used as input for the next row.

Next, we mentioned that Line 12 of Algorithm 7 invokes Algorithm 8 (Duadic Hop). Its inputs are the represents table R, a vertex cover S, an endpoint to be frozen ψ, an endpoint to be removed ω, and a list of visited endpoints. By design, when Algorithm 8 is invoked, either ψ or ω will be null. An S-duadic hop always begins with the removal of an endpoint, as evident from Line 12 of Algorithm 7. Foremost, an endpoint ω can be removed only if it is not visited (Line 3). If the endpoint ω is marked as visited, it implies it was frozen during the removal of another endpoint in another row. If not visited, ω is now marked as visited

[41] The represents list L_u for each endpoint u in table R is given in the table such that no endpoint is removed from any list. In other words, the represents list of each endpoint is the same as it was during the output of Algorithm 2. This information is needed to hop to different rows.

[42] During each iteration, the values of R and S that are provided as inputs may be different.

(Line 6), removed from represents table R (Line 7), and removed from the vertex cover S (Line 8). A queue is maintained to ensure that for each endpoint ω that is removed, the endpoints that correspond to the neighboring vertices in the graph are marked as visited (if not already done so) and added to the queue if not already frozen (Lines 10 to 25). For each endpoint u in the queue, we hop from the row containing ω to the row containing u, which needs to be frozen (Lines 26 to 29). An endpoint ψ is frozen in table R (Line 33) and added to the vertex cover S (Line 34) only when an endpoint was removed earlier. Next, if the neighboring endpoint of ψ is in the vertex cover, it needs to be removed (if possible). Finally, an S-duadic hop will terminate only when removing or freezing an endpoint in a duad or in another row, respectively, is not possible. If the size of S decreases from the time when Line 12 of Algorithm 7 invoked Algorithm 8, then an S-duadic hop is an S-diminishing hop. $\square$

We proved that Algorithm 7, along with Algorithm 8, performs an S-diminishing hop when one exists. Next, we prove that $\frac{m}{2}$ calls by Algorithm 3 to Algorithm 7 guarantees that there is no S-diminishing hop when the loop in Algorithm 3 terminates.

Lemma 8. *Given a represents table R where each endpoint is either frozen or removed and a vertex cover S that corresponds to the frozen endpoints in the table R, it takes at most m^2 S-duadic hops to ensure that there is no S-diminishing hop.*

Proof. The proof for this lemma is divided into two parts: (i) to show the algorithm executes at most m^2 S-duadic hops, and (ii) an S-diminishing hop cannot exist after at most m^2 S-duadic hops.

(i) Line 7 of Algorithm 3 invokes Algorithm 7 $\frac{m}{2}$ times. Next, in the worst case, during each of the $\frac{m}{2}$ iterations, there can be at most $\frac{m}{2} - 1$ rows with a duad. Therefore, Line 12 of Algorithm 7 invokes Algorithm 8 at most $\frac{m}{2} \cdot 2 \cdot (\frac{m}{2} - 1) \approx m^2$ times. Finally, Algorithm 8 can call itself at most m times, which is not considered in this analysis because the recursive calls are part of an ongoing S-duadic hop, but not a new hop in itself. Hence, we showed that the algorithm executes at most m^2 S-duadic hops.

(ii) It remains to be proven that an S-diminishing hop cannot exist after at most m^2 S-duadic hops[43]. Firstly, we know that at least one of the $2 \cdot (\frac{m}{2} - 1)$ S-duadic hops invoked in Line 12 of Algorithm 7 is an S-diminishing hop, assuming an S-diminishing hop exists. More specifically, if there exists an S-diminishing hop, then at least one of the endpoints in at least one of the rows with a duad will be removed such that the remaining frozen endpoints in the represents table R still form a vertex cover. Removal of such an endpoint during an S-duadic hop implies an S-diminishing hop. Next, in a given represents table R, there can be at most $\frac{m}{2} - 1$ rows consisting of a duad. Hence, each time Line 7 of Algorithm 3 invokes Algorithm 7, at least one row of the represents table R will become duad-less, again assuming an S-diminishing hop exists. In other words, each S-diminishing hop implies that the number of duads in the represents table R is decreasing (by at least one in the worst case)[44]. Therefore, in the worst

[43] The idea behind having $\frac{m}{2}$ iterations of Algorithm 7 is inspired by bubble sort. More specifically, in bubble sort, after each iteration, an element's position is fixed. Similarly, after each S-diminishing hop, the number of rows with a duad in the represents table R decreases by at least one.

[44] Alternative explanation: Line 7 of Algorithm 3 invokes Algorithm 7 $\frac{m}{2}$ times. Each iteration consists of $2 \cdot (\frac{m}{2} - 1)$ S-duadic hops invoked in Line 12 of Algorithm 7. Moreover, during each iteration, if an S-diminishing hop exists, then the number of duads goes down by at least one. Consequently, given that a given represents table R can have at most $\frac{m}{2} - 1$ rows with a duad to begin with, the $\frac{m}{2}$ invokes to Algorithm 7 by Algorithm 3 guarantees that an S-diminishing hop does not exist because either (i) the represents table R will have no rows with a duad or (ii) no possible S-duadic hop reduces the number of frozen endpoints in the represents table R.

case, after at most $\frac{m}{2}$ calls to Algorithm 7 and consequently, at most m^2 S-duadic hops, there cannot be an S-diminishing hop.

Overall, we showed that after $\frac{m}{2}$ iterations of Lines 6 to 8 in Algorithm 3, there cannot exist an S-diminishing hop in the represents table R. □

In summary, we proved that Algorithm 7, along with Algorithm 8, is an algorithm (i) to perform an S-diminishing hop (Lemma 7) when one exists, (ii) that guarantees that no S-diminishing hop exists when it terminates after performing at most m^2 S-duadic hops across $\frac{m}{2}$ calls to Algorithm 7 by Algorithm 3 (Lemma 8), and (iii) that is based on the fact that the non-existence of an S-diminishing hop implies S is a minimum vertex cover (Theorem 7).

After the termination of the loop in Line 8 of Algorithm 3, Line 9 of the algorithm returns a minimum vertex cover S to Line 10 of Algorithm 1 because there will be no S-diminishing hop (Lemma 7, Lemma 8, Theorem 7). This discussion effectively completes the proof of this lemma.

In summary, recall that we posited that for the Algorithm 1 to return **Yes**, Lines 8 and 10 of Algorithm 1 must be executed. We subsequently discussed the execution of these lines. We proved that for frozen endpoints in the resultant represents table R and the corresponding vertex cover S, there exists no S-diminishing hop after the execution of Line 10 of Algorithm 1. This implies that S is a minimum vertex cover (Theorem 7). Subsequently, when Line 12 of Algorithm 1 returns **Yes**, the given instance of $\texttt{VC} - \texttt{CBG}$ must be a **Yes** instance. Hence, if Algorithm 1 returns **Yes**, then the given instance of $\texttt{VC} - \texttt{CBG}$ is a **Yes** instance. This completes the proof in the forward direction. □

The Lemma 5 and Lemma 6 complete both directions of the proof of correctness, which completes the proof of Theorem 8.

8 Time Complexity Analysis

In this section, we discuss the time complexity of the algorithm (Table 16, Table 17, Table 18, Table 19, Table 20, Table 21, Table 22, Table 23). Each table corresponds to each algorithm (ranging from Algorithm 1 to Algorithm 8). m denotes the number of vertices V and n denotes the number of edges E. However, for cubic graphs, we know that $n = \frac{3m}{2} = \mathcal{O}(m)$. Hence, for simplicity and in line with the literature, we compute time complexity with respect to m.

In each table, we give the complexity of each line (each operation), the complexity of the loop (complexity of line multiplied by the number of loop iterations) and the dominant complexity. For convenience, the beginning of a loop, specifically the number of loop iterations, is highlighted (e.g., Line 4 in Table 17). Each statement within the loop is prefixed with a pointer (▶). In the case of nested loops, an additional pointer (▷, >) is used. Whenever an algorithm calls another algorithm, the latter's worst-case time complexity becomes the former's line complexity, which is denoted by square brackets ([Table x]; e.g., Line 8 in Table 16).

Theorem 9. *The asymptotic running time of Algorithm 1 is* $\mathcal{O}(m^5)$.

Proof. Line 10 in Algorithm 1 dominates the complexity of all other lines as shown in Table 16. This dominant complexity is $\mathcal{O}(m^5)$. Hence, the time complexity of the entire algorithm is $\mathcal{O}(m^5)$. □

Time Complexity of Algorithms with Recursive Calls We elaborate upon the time complexity of Algorithm 6 (Table 21) and Algorithm 8 (Table 23) because the time complexity of the remainder of the algorithms is self-explanatory from the respective tables. Both of these algorithms consist of recursive calls that require discussion.

- Algorithm 6 has recursive calls in line 10 and line 13. However, by design, Algorithm 6 executes at most m times because each time it is executed, at least one vertex is either removed or frozen. Hence, after at most m calls, no unfrozen or unremoved vertex exists. Each call takes $\mathcal{O}(m)$ time. Overall, in the worst case, the height of the recursion tree is m and each level has one subproblem taking $\mathcal{O}(m)$. Thus, total complexity is $\mathcal{O}(m^2)$.
- Algorithm 8 has recursive calls in line 28 and line 38. Again, by design, Algorithm 8 executes at most m times. This is because each time the algorithm is executed, at least one endpoint is marked as visited using the variable λ. Hence, in the worst case, if one endpoint is marked as visited during each call, then there can be at most m calls. After this, no unvisited endpoint will exist. Each call takes $\mathcal{O}(m^2)$ time (each operation is $\mathcal{O}(m)$ and there are at most m operations). Thus, total complexity is $\mathcal{O}(m) \cdot \mathcal{O}(m^2) = \mathcal{O}(m^3)$. Notably, during each call, the algorithm never executes both, line 28 and line 38, together. This is by design as each hop within an S-duadic hop can either result in freezing of an endpoint or removal of an endpoint.

Table 16: Line wise time complexity of Algorithm 1. W.l.o.g., we assume the average length of vertex names is a constant and hence, ignore it in time complexity analysis of Line 1.

Line Number	Line complexity	Loop complexity	Dominant complexity
1	$\mathcal{O}(m \cdot \log m)$	-	$\mathcal{O}(m \cdot \log m)$
2	-	-	$\mathcal{O}(m \cdot \log m)$
3	$\mathcal{O}(m^3)$	-	$\mathcal{O}(m^3)$
4	$\mathcal{O}(1)$	-	$\mathcal{O}(m^3)$
5	$\mathcal{O}(1)$	-	$\mathcal{O}(m^3)$
6	-	-	$\mathcal{O}(m^3)$
7	-	-	$\mathcal{O}(m^3)$
8	$\mathcal{O}(m^3)$ [Table 17]	-	$\mathcal{O}(m^3)$
9	-	-	$\mathcal{O}(m^3)$
10	$\mathcal{O}(m^5)$ [Table 18]	-	$\mathcal{O}(m^5)$
11	$\mathcal{O}(1)$	-	$\mathcal{O}(m^5)$
12	$\mathcal{O}(1)$	-	$\mathcal{O}(m^5)$
13	-	-	$\mathcal{O}(m^5)$
14	$\mathcal{O}(1)$	-	$\mathcal{O}(m^5)$

Table 17: Line wise time complexity of Algorithm 2. A highlight denotes the number of loop iterations. A pointer (▶) denotes that a line is within a loop. An additional pointer (▷) denotes a nested loop. Note that the BFS-tree is traversed at most m times (by design) but asymptotically it may traverse m^2 times. Hence, we keep the latter time complexity as it does not impact the overall complexity of the algorithm.

Line Number	Line complexity	Loop complexity	Dominant complexity
1	$\mathcal{O}(m+m)$	-	$\mathcal{O}(m)$
2	$\mathcal{O}(1)$	-	$\mathcal{O}(m)$
3	-	-	$\mathcal{O}(m)$
4	$\mathcal{O}(1)$	$\mathcal{O}(m)$	$\mathcal{O}(m)$
5	$\mathcal{O}(1)$	▶ $\mathcal{O}(m^2)$	$\mathcal{O}(m^2)$
6	$\mathcal{O}(m)$	▶ ▷ $\mathcal{O}(m^3)$	$\mathcal{O}(m^3)$
7	$\mathcal{O}(1)$	▶ ▷ $\mathcal{O}(m^2)$	$\mathcal{O}(m^3)$
8	$\mathcal{O}(m)$	▶ ▷ $\mathcal{O}(m^3)$	$\mathcal{O}(m^3)$
9	$\mathcal{O}(1)$	▶ ▷ $\mathcal{O}(m^2)$	$\mathcal{O}(m^3)$
10	-	-	$\mathcal{O}(m^3)$
11	$\mathcal{O}(m)$	▶ ▷ $\mathcal{O}(m^3)$	$\mathcal{O}(m^3)$
12	$\mathcal{O}(1)$	▶ ▷ $\mathcal{O}(m^3)$	$\mathcal{O}(m^3)$
13	$\mathcal{O}(m)$	▶ ▷ $\mathcal{O}(m^3)$	$\mathcal{O}(m^3)$
14	$\mathcal{O}(m)$	▶ ▷ $\mathcal{O}(m^3)$	$\mathcal{O}(m^3)$
15	-	-	$\mathcal{O}(m^3)$
16	-	-	$\mathcal{O}(m^3)$
17	$\mathcal{O}(1)$	-	$\mathcal{O}(m^3)$

Table 18: Line wise time complexity of Algorithm 3. A highlight denotes the number of loop iterations. A pointer (▶) denotes that a line is within a loop.

Line Number	Line complexity	Loop complexity	Dominant complexity
1	$\mathcal{O}(1)$	-	$\mathcal{O}(1)$
2	$\mathcal{O}(1)$	-	$\mathcal{O}(1)$
3	$\mathcal{O}(m)$	-	$\mathcal{O}(m)$
4	$\mathcal{O}(m^2)$ [Table 19]	-	$\mathcal{O}(m^2)$
5	$\mathcal{O}(m^3)$ [Table 20]	-	$\mathcal{O}(m^3)$
6	$\mathcal{O}(1)$	$\mathcal{O}(m)$	$\mathcal{O}(m^3)$
7	$\mathcal{O}(m^4)$ [Table 22]	▶ $\mathcal{O}(m^5)$	$\mathcal{O}(m^5)$
8	-	-	$\mathcal{O}(m^5)$
9	$\mathcal{O}(1)$	-	$\mathcal{O}(m^5)$

Table 19: Line wise time complexity of Algorithm 4. A highlight denotes the number of loop iterations. A pointer (▶) denotes that a line is within a loop. Each additional pointer (▷, >) denotes a nested loop.

Line Number	Line complexity	Loop complexity	Dominant complexity
1	-	-	-
2	$\mathcal{O}(1)$	$\mathcal{O}(m)$	$\mathcal{O}(m)$
3	-	-	$\mathcal{O}(m)$
4	$\mathcal{O}(1)$	▶ $\mathcal{O}(m)$	$\mathcal{O}(m)$
5	$\mathcal{O}(1)$	▶ ▷ $\mathcal{O}(m)$	$\mathcal{O}(m)$
6	$\mathcal{O}(1)$	▶ ▷ $\mathcal{O}(m)$	$\mathcal{O}(m)$
7	-	-	$\mathcal{O}(m)$
8	$\mathcal{O}(1)$	▶ ▷ $\mathcal{O}(m)$	$\mathcal{O}(m)$
9	$\mathcal{O}(1)$	▶ ▷ $\mathcal{O}(m)$	$\mathcal{O}(m)$
10	-	-	$\mathcal{O}(m)$
11	-	-	$\mathcal{O}(m)$
12	$\mathcal{O}(1)$	▶ ▷ $\mathcal{O}(m)$	$\mathcal{O}(m)$
13	$\mathcal{O}(1)$	▶ ▷ $\mathcal{O}(m^2)$	$\mathcal{O}(m^2)$
14	$\mathcal{O}(1)$	▶ ▷ > $\mathcal{O}(m^2)$	$\mathcal{O}(m^2)$
15	-	-	$\mathcal{O}(m^2)$
16	$\mathcal{O}(1)$	▶ ▷ > $\mathcal{O}(m^2)$	$\mathcal{O}(m^2)$
17	$\mathcal{O}(1)$	▶ ▷ > $\mathcal{O}(m^2)$	$\mathcal{O}(m^2)$
18	$\mathcal{O}(1)$	▶ ▷ > $\mathcal{O}(m^2)$	$\mathcal{O}(m^2)$
19	$\mathcal{O}(1)$	▶ ▷ > $\mathcal{O}(m^2)$	$\mathcal{O}(m^2)$
20	-	-	$\mathcal{O}(m^2)$
21	-	-	$\mathcal{O}(m^2)$
22	-	-	$\mathcal{O}(m^2)$
23	-	-	$\mathcal{O}(m^2)$
24	-	-	$\mathcal{O}(m^2)$
25	-	-	$\mathcal{O}(m^2)$
26	-	-	$\mathcal{O}(m^2)$
27	$\mathcal{O}(1)$	-	$\mathcal{O}(m^2)$

Table 20: Line wise time complexity of Algorithm 5. A highlight denotes the number of loop iterations. A pointer (▶) denotes that a line is within a loop.

Line Number	Line complexity	Loop complexity	Dominant complexity
1	-	-	-
2	$\mathcal{O}(1)$	$\mathcal{O}(m)$	$\mathcal{O}(m)$
3	$\mathcal{O}(m^2)$ [Table 19]	▶ $\mathcal{O}(m^3)$	$\mathcal{O}(m^3)$
4	$\mathcal{O}(1)$	▶ $\mathcal{O}(m)$	$\mathcal{O}(m^3)$
5	-	-	$\mathcal{O}(m^3)$
6	$\mathcal{O}(1)$	▶ $\mathcal{O}(m)$	$\mathcal{O}(m^3)$
7	$\mathcal{O}(m^2)$ [Table 21]	▶ $\mathcal{O}(m^3)$	$\mathcal{O}(m^3)$
8	-	-	$\mathcal{O}(m^3)$
9	-	-	$\mathcal{O}(m^3)$
10	$\mathcal{O}(1)$	▶ $\mathcal{O}(m)$	$\mathcal{O}(m^3)$
11	$\mathcal{O}(m^2)$ [Table 21]	▶ $\mathcal{O}(m^3)$	$\mathcal{O}(m^3)$
12	-	-	$\mathcal{O}(m^3)$
13	$\mathcal{O}(m^2)$ [Table 21]	▶ $\mathcal{O}(m^3)$	$\mathcal{O}(m^3)$
14	-	-	$\mathcal{O}(m^3)$
15	-	-	$\mathcal{O}(m^3)$
16	-	-	$\mathcal{O}(m^3)$
17	$\mathcal{O}(1)$	-	$\mathcal{O}(m^3)$

Table 21: Line wise time complexity of Algorithm 6. A highlight denotes the number of loop iterations. A pointer (▶) denotes that a line is within a loop.

Line Number	Line complexity	Loop complexity	Dominant complexity
1	-	-	-
2	$\mathcal{O}(m)$	-	$\mathcal{O}(m)$
3	$\mathcal{O}(1)$	-	$\mathcal{O}(m)$
4	$\mathcal{O}(m)$	-	$\mathcal{O}(m)$
5	$\mathcal{O}(m)$	-	$\mathcal{O}(m)$
6	-	-	$\mathcal{O}(m)$
7	$\mathcal{O}(m)$	-	$\mathcal{O}(m)$
8	$\mathcal{O}(m)$	-	$\mathcal{O}(m)$
9	$\mathcal{O}(m)$	$\mathcal{O}(m)$	$\mathcal{O}(m)$
10	$\mathcal{O}(m)$	▶ $\mathcal{O}(m^2)$	$\mathcal{O}(m^2)$
11	-	-	$\mathcal{O}(m^2)$
12	$\mathcal{O}(m)$	$\mathcal{O}(m)$	$\mathcal{O}(m^2)$
13	$\mathcal{O}(m)$	▶ $\mathcal{O}(m^2)$	$\mathcal{O}(m^2)$
14	-	-	$\mathcal{O}(m^2)$
15	$\mathcal{O}(m)$	-	$\mathcal{O}(m^2)$
16	$\mathcal{O}(1)$	-	$\mathcal{O}(m^2)$

Table 22: Line wise time complexity of Algorithm 7. A highlight denotes the number of loop iterations. A pointer (▶) denotes that a line is within a loop. An additional pointer (▷) denotes a nested loop.

Line Number	Line complexity	Loop complexity	Dominant complexity
1	$\mathcal{O}(1)$	-	$\mathcal{O}(1)$
2	$\mathcal{O}(m)$	-	$\mathcal{O}(m)$
3	$\mathcal{O}(m)$	-	$\mathcal{O}(m)$
4	-	-	$\mathcal{O}(m)$
5	$\mathcal{O}(1)$	$\mathcal{O}(m)$	$\mathcal{O}(m)$
6	$\mathcal{O}(m)$	▶ $\mathcal{O}(m^2)$	$\mathcal{O}(m^2)$
7	$\mathcal{O}(m)$	▶ $\mathcal{O}(m^2)$	$\mathcal{O}(m^2)$
8	$\mathcal{O}(m)$	▶ $\mathcal{O}(m^2)$	$\mathcal{O}(m^2)$
9	-	-	$\mathcal{O}(m^2)$
10	$\mathcal{O}(1)$	▶ $\mathcal{O}(m)$	$\mathcal{O}(m^2)$
11	$\mathcal{O}(1)$	▶ $\mathcal{O}(m)$	$\mathcal{O}(m^2)$
12	$\mathcal{O}(m^3)$ [Table 23]	▶ ▷ $\mathcal{O}(m^4)$	$\mathcal{O}(m^4)$
13	-	-	$\mathcal{O}(m^4)$
14	$\mathcal{O}(1)$	▶ ▷ $\mathcal{O}(m)$	$\mathcal{O}(m^4)$
15	$\mathcal{O}(m)$	▶ ▷ $\mathcal{O}(m^2)$	$\mathcal{O}(m^4)$
16	$\mathcal{O}(m)$	▶ ▷ $\mathcal{O}(m^2)$	$\mathcal{O}(m^4)$
17	$\mathcal{O}(m)$	▶ ▷ $\mathcal{O}(m^2)$	$\mathcal{O}(m^4)$
18	-	-	$\mathcal{O}(m^4)$
19	$\mathcal{O}(m)$	▶ ▷ $\mathcal{O}(m^2)$	$\mathcal{O}(m^4)$
20	$\mathcal{O}(m)$	▶ ▷ $\mathcal{O}(m^2)$	$\mathcal{O}(m^4)$
21	$\mathcal{O}(m)$	▶ ▷ $\mathcal{O}(m^2)$	$\mathcal{O}(m^4)$
22	-	-	$\mathcal{O}(m^4)$
23	$\mathcal{O}(m)$	▶ $\mathcal{O}(m^2)$	$\mathcal{O}(m^4)$
24	$\mathcal{O}(m)$	▶ $\mathcal{O}(m^2)$	$\mathcal{O}(m^4)$
25	$\mathcal{O}(m)$	▶ $\mathcal{O}(m^2)$	$\mathcal{O}(m^4)$
26	-	-	$\mathcal{O}(m^4)$
27	-	-	$\mathcal{O}(m^4)$
28	$\mathcal{O}(1)$	-	$\mathcal{O}(m^4)$

Table 23: Line wise time complexity of Algorithm 8. A highlight denotes the number of loop iterations. A pointer (▶) denotes that a line is within a loop.

Line Number	Line complexity	Loop complexity	Dominant complexity
1	-	-	-
2	$\mathcal{O}(1)$	-	$\mathcal{O}(1)$
3	$\mathcal{O}(m)$	-	$\mathcal{O}(m)$
4	$\mathcal{O}(1)$	-	$\mathcal{O}(m)$
5	-	-	$\mathcal{O}(m)$
6	$\mathcal{O}(1)$	-	$\mathcal{O}(m)$
7	$\mathcal{O}(m)$	-	$\mathcal{O}(m)$
8	$\mathcal{O}(m)$	-	$\mathcal{O}(m)$
9	$\mathcal{O}(1)$	-	$\mathcal{O}(m)$
10	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(m)$
11	$\mathcal{O}(m)$	▶ $\mathcal{O}(m)$	$\mathcal{O}(m)$
12	$\mathcal{O}(1)$	▶ $\mathcal{O}(1)$	$\mathcal{O}(m)$
13	$\mathcal{O}(m)$	▶ $\mathcal{O}(m)$	$\mathcal{O}(m)$
14	$\mathcal{O}(m)$	▶ $\mathcal{O}(m)$	$\mathcal{O}(m)$
15	-	-	$\mathcal{O}(m)$
16	-	-	$\mathcal{O}(m)$
17	-	-	$\mathcal{O}(m)$
18	$\mathcal{O}(m)$	$\mathcal{O}(1)$	$\mathcal{O}(m)$
19	$\mathcal{O}(m)$	▶ $\mathcal{O}(m)$	$\mathcal{O}(m)$
20	$\mathcal{O}(1)$	▶ $\mathcal{O}(1)$	$\mathcal{O}(m)$
21	$\mathcal{O}(m)$	▶ $\mathcal{O}(m)$	$\mathcal{O}(m)$
22	$\mathcal{O}(m)$	▶ $\mathcal{O}(m)$	$\mathcal{O}(m)$
23	-	-	$\mathcal{O}(m)$
24	-	-	$\mathcal{O}(m)$
25	-	-	$\mathcal{O}(m)$
26	$\mathcal{O}(1)$	$\mathcal{O}(m)$	$\mathcal{O}(m)$
27	$\mathcal{O}(m)$	▶ $\mathcal{O}(m^2)$	$\mathcal{O}(m^2)$
28	$\mathcal{O}(m^2)$	▶ $\mathcal{O}(m^3)$	$\mathcal{O}(m^3)$
29	-	-	$\mathcal{O}(m^3)$
30	-	-	$\mathcal{O}(m^3)$
31	-	-	$\mathcal{O}(m^3)$
32	$\mathcal{O}(1)$	-	$\mathcal{O}(m^3)$
33	$\mathcal{O}(m)$	-	$\mathcal{O}(m^3)$
34	$\mathcal{O}(1)$	-	$\mathcal{O}(m^3)$
35	-	-	$\mathcal{O}(m^3)$
36	$\mathcal{O}(m)$	-	$\mathcal{O}(m^3)$
37	$\mathcal{O}(m)$	-	$\mathcal{O}(m^3)$
38	$\mathcal{O}(m^2)$	-	$\mathcal{O}(m^3)$
39	-	-	$\mathcal{O}(m^3)$
40	-	-	$\mathcal{O}(m^3)$
41	$\mathcal{O}(1)$	-	$\mathcal{O}(m^3)$

9 Concluding Remarks

In this two-part study on the vertex cover problem on cubic bridgeless graphs ($\texttt{VC} - \texttt{CBG}$), we discovered that: (i) $\texttt{VC} - \texttt{CBG}$ is **NP**-complete (Theorem 1) and (ii) $\texttt{VC} - \texttt{CBG} \in \mathbf{P}$ (Theorem 2)[45]. As a consequence of these two theorems combined with Proposition 1(c) in [Coo00], which states that if some language L is **NP**-complete and $L \in \mathbf{P}$, then $\mathbf{P} = \mathbf{NP}$, we get the following corollary:

Corollary 2. $\mathbf{P} = \mathbf{NP}$.

9.1 Brief Additional Remarks

9.1.1 Practical Consequences + Ethical Implications

We presented a polynomial-time algorithm for an **NP**-complete problem. However, the problem space for this paper has been narrowed down to cubic bridgeless graphs ($\texttt{VC} - \texttt{CBG}$). Hence, the algorithm's practical utility depends on how generalizable it is for various graph types. Moreover, even for the narrowed-down problem of $\texttt{VC} - \texttt{CBG}$, the time complexity is a higher-order polynomial. Therefore, the time complexity will only worsen for the general case (and in turn, for using the algorithm for other **NP**-complete problems; see Footnote 11). Hence, the practical impact of our work is (none to) limited until more efficient versions of this (galactic) algorithm are found, for $\texttt{VC} - \texttt{CBG}$, for $\texttt{VC} - \texttt{CG}$, for $\texttt{VC}$, or for other **NP**-complete problems.

Given that we do not expect any immediate practical consequences of our work, we do not expect any ethical implications either. That said, our work may eventually lead to algorithms that, for example, may break certain variants of cryptography. Hence, we stress the need for a study to understand the immediate and long-term implications of the algorithm to various fields. This study should at least (i) provide appropriate quantification (e.g., how long is "long-term", or what order of the polynomial is not "practical" for how big a size of data in which field) and (ii) suggest alternative solutions wherever applicable (e.g., use information-theoretic security).

9.1.2 Existence of Verifier-Solver Gap?

One of the implications of $\mathbf{P} = \mathbf{NP}$ is that finding a solution is as easy as verifying one. However, even with $\mathbf{P} = \mathbf{NP}$, we conjecture that for a *given* problem, there always exists a gap between the time needed to verify a given solution and the time needed to solve the *given* problem, in presence of certain standard conditions. Informally, we ask the following question: *"Is there a computational problem such that the time taken to verify a given solution for the problem is the* ***exact*** *same as the time taken to solve the problem?"*.

The three standard conditions are:

1. By a computational problem, we strictly mean the decision version of the problem.
2. A problem may not be stated such that verification of a given solution requires solving the entire problem again.[46]

[45] Theorem 2 is proven via Theorem 8 (algorithm's proof of correctness) and Theorem 9 (time complexity).

[46] There is a relaxed version of the condition: "The problem is in **NP**". This condition leads to a larger space of problems to search from. However, we propose the use of the enumerated condition. One of the aims is to omit every decision problem stated in a way where the time needed to verify a given solution is strictly greater than the time needed to find a solution, which may be caused by the framing of the question or by the certificate(s) provided. For instance, we omit a problem in **NP** that is stated analogous to the following problem statement: "Given a graph G and a vertex cover S, is S a minimum-size vertex cover of graph G?".

3. The size of output is not asymptotically larger than the size of input. If it were larger, verifying the output will need the same time as solving the problem in many cases.

We stress that for the aforementioned question, we depart from an asymptotical comparison of running times to a comparison of exact running times unless mentioned otherwise. A relaxed variant of the question using the same set of conditions is: *"Is there a computational problem such that the time taken to verify a given solution for the problem is the* ***exact*** *same as the time taken to solve the problem* ***and*** *the time taken to solve the problem is asymptotically larger than the size of the input?"*.

References

[Aar16] Scott Aaronson. P=\limitsˆ? np. *Open problems in mathematics*, pages 1–122, 2016.

[ABLT06] Sanjeev Arora, Béla Bollobás, László Lovász, and Iannis Tourlakis. Proving integrality gaps without knowing the linear program. *Theory OF Computing*, 2:19–51, 2006.

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Annals of mathematics*, 160(2):781–793, 2004.

[AKS11] Per Austrin, Subhash Khot, and Muli Safra. Inapproximability of vertex cover and independent set in bounded degree graphs. *Theory of Computing*, 7(1):27–43, 2011.

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM (JACM)*, 45(3):501–555, 1998.

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of np. *Journal of the ACM (JACM)*, 45(1):70–122, 1998.

[AW09] Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *ACM Transactions on Computation Theory (TOCT)*, 1(1):1–54, 2009.

[BBDL01] Therese C Biedl, Prosenjit Bose, Erik D Demaine, and Anna Lubiw. Efficient algorithms for petersen's matching theorem. *Journal of Algorithms*, 38(1):110–134, 2001.

[Ber57] Claude Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences*, 43(9):842–844, 1957.

[BGS75] Theodore Baker, John Gill, and Robert Solovay. Relativizations of the p=?np question. *SIAM Journal on computing*, 4(4):431–442, 1975.

[BIP19] Peter Bürgisser, Christian Ikenmeyer, and Greta Panova. No occurrence obstructions in geometric complexity theory. *Journal of the American Mathematical Society*, 32(1):163–193, 2019.

[BJS22] Marin Bougeret, Bart MP Jansen, and Ignasi Sau. Bridge-depth characterizes which minor-closed structural parameterizations of vertex cover admit a polynomial kernel. *SIAM Journal on Discrete Mathematics*, 36(4):2737–2773, 2022.

[BLW86] Norman Biggs, E Keith Lloyd, and Robin J Wilson. *Graph Theory, 1736-1936*. Oxford University Press, 1986.

[BM08] John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph theory*. Springer Publishing Company, Incorporated, 2008.

[BW15] Samuel R Buss and Ryan Williams. Limits on alternation trading proofs for time–space lower bounds. *computational complexity*, 24:533–600, 2015.

[CLRS16] Siu On Chan, James R Lee, Prasad Raghavendra, and David Steurer. Approximate constraint satisfaction requires large lp relaxations. *Journal of the ACM (JACM)*, 63(4):1–22, 2016.

[CLRS22] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[Cob65] Alan Cobham. The intrinsic computational difficulty of functions. In Yehoshua Bar-Hillel, editor, *Logic, methodology and philosophy of science*, pages 24–30. North-Holland Pub. Co., 1965.

[Coo71] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971.

[Coo00] Stephen Cook. The p versus np problem. *Clay Mathematics Institute*, 2(6):3, 2000.

[Coo03] Stephen Cook. The importance of the p versus np question. *Journal of the ACM (JACM)*, 50(1):27–29, 2003.

[CS12] Maria Chudnovsky and Paul Seymour. Perfect matchings in planar cubic graphs. *Combinatorica*, 32(4):403–424, 2012.

[CŽ24] Lorenzo Ciardo and Stanislav Živnỳ. Semidefinite programming and linear equations vs. homomorphism problems. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 1935–1943, 2024.

[DF95] Rod G Downey and Michael R Fellows. Fixed-parameter tractability and completeness i: Basic results. *SIAM Journal on computing*, 24(4):873–921, 1995.

[DF12] Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.

[DGKR03] Irit Dinur, Venkatesan Guruswami, Subhash Khot, and Oded Regev. A new multilayered pcp and the hardness of hypergraph vertex cover. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 595–601, 2003.

[Din07] Irit Dinur. The pcp theorem by gap amplification. *Journal of the ACM (JACM)*, 54(3):12–es, 2007.

[DIP20] Julian Dörfler, Christian Ikenmeyer, and Greta Panova. On geometric complexity theory: Multiplicity obstructions are stronger than occurrence obstructions. *SIAM Journal on Applied Algebra and Geometry*, 4(2):354–376, 2020.

[DS05] Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of mathematics*, pages 439–485, 2005.

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

[EKK+11] Louis Esperet, František Kardoš, Andrew King, Daniel Král', and Sergey Norine. Exponentially many perfect matchings in cubic graphs. *Advances in Mathematics*, 227(4):1646–1664, 2011.

[EKK12] Louis Esperet, František Kardoš, and Daniel Král'. A superlinear bound on the number of perfect matchings in cubic bridgeless graphs. *European Journal of Combinatorics*, 33(5):767–798, 2012. EuroComb '09.

[EKŠŠ10] Louis Esperet, Daniel Král', Petr Škoda, and Riste Škrekovski. An improved linear bound on the number of perfect matchings in cubic graphs. *European Journal of Combinatorics*, 31(5):1316–1334, 2010.

[EPL82] Jack Edmonds, WR Pulleyblank, and L Lovász. Brick decompositions and the matching rank of graphs. *Combinatorica*, 2:247–274, 1982.

[Eul36] Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, pages 128–140, 1736.

[Fei03] Uriel Feige. Vertex cover is hardest to approximate on regular graphs. *Technical Report MCS03–15*, 2003.

[FFR97] Ralph Faudree, Evelyne Flandrin, and Zdeněk Ryjáček. Claw-free graphs—a survey. *Discrete Mathematics*, 164(1-3):87–147, 1997.

[FGL+96] Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM (JACM)*, 43(2):268–292, 1996.

[FMP+15] Samuel Fiorini, Serge Massar, Sebastian Pokutta, Hans Raj Tiwary, and Ronald De Wolf. Exponential lower bounds for polytopes in combinatorial optimization. *Journal of the ACM (JACM)*, 62(2):1–23, 2015.

[For09] Lance Fortnow. The status of the p versus np problem. *Communications of the ACM*, 52(9):78–86, 2009.

[For21] Lance Fortnow. Fifty years of p vs. np and the possibility of the impossible. *Communications of the ACM*, 65(1):76–85, 2021.

[Gib85] Alan Gibbons. *Algorithmic graph theory.* Cambridge university press, 1985.

[GJ02] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[GJS74] Michael R Garey, David S Johnson, and Larry Stockmeyer. Some simplified np-complete problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 47–63, 1974.

[GNW07] Jiong Guo, Rolf Niedermeier, and Sebastian Wernicke. Parameterized complexity of vertex cover variants. *Theory of Computing Systems*, 41:501–520, 2007.

[GW24] Paweł Gawrychowski and Mateusz Wasylkiewicz. Finding perfect matchings in bridgeless cubic multigraphs without dynamic (2-) connectivity. In *32nd Annual European Symposium on Algorithms (ESA 2024)*, pages 59–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

[Hal35] P Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 1(1):26–30, 1935.

[Hås01] Johan Håstad. Some optimal inapproximability results. *Journal of the ACM (JACM)*, 48(4):798–859, 2001.

[IP17] Christian Ikenmeyer and Greta Panova. Rectangular kronecker coefficients and plethysms in geometric complexity theory. *Advances in Mathematics*, 319:40–66, 2017.

[Kar72] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

[Kho02] Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 767–775, 2002.

[Kho19] Subhash Khot. On the proof of the 2-to-2 games conjecture. *Current Developments in Mathematics*, 2019(1):43–94, 2019.

[KMS23] Subhash Khot, Dor Minzer, and Muli Safra. Pseudorandom sets in grassmann graph have near-perfect expansion. *Annals of Mathematics*, 198(1):1–92, 2023.

[Kon31] Denés Konig. Gráfok és mátrixok. matematikai és fizikai lapok, 38: 116–119, 1931.

[KR08] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2- ε. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.

[KSS09] Daniel Král', Jean-Sébastien Sereni, and Michael Stiebitz. A new lower bound on the number of perfect matchings in cubic graphs. *SIAM Journal on Discrete Mathematics*, 23(3):1465–1483, 2009.

[KT06] Jon Kleinberg and Eva Tardos. *Algorithm design*. Pearson Education India, 2006.

[Lev73] Leonid Anatolevich Levin. Universal sequential search problems. *Problemy peredachi informatsii*, 9(3):115–116, 1973.

[LP09] László Lovász and Michael D Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.

[LRS15] James R Lee, Prasad Raghavendra, and David Steurer. Lower bounds on the size of semidefinite programming relaxations. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 567–576, 2015.

[LV99] Richard J Lipton and Anastasios Viglas. On the complexity of sat. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 459–464. IEEE, 1999.

[Min80] George J Minty. On maximal independent sets of vertices in claw-free graphs. *Journal of Combinatorial Theory, Series B*, 28(3):284–304, 1980.

[MS01] Ketan D Mulmuley and Milind Sohoni. Geometric complexity theory i: An approach to the p vs. np and related problems. *SIAM Journal on Computing*, 31(2):496–526, 2001.

[MS08] Ketan D Mulmuley and Milind Sohoni. Geometric complexity theory ii: Towards explicit obstructions for embeddings among class varieties. *SIAM Journal on Computing*, 38(3):1175–1206, 2008.

[Mul99] Ketan Mulmuley. Lower bounds in a parallel model without bit operations. *SIAM Journal on Computing*, 28(4):1460–1509, 1999.

[Mul11] Ketan D Mulmuley. On p vs. np and geometric complexity theory: Dedicated to sri ramakrishna. *Journal of the ACM (JACM)*, 58(2):1–26, 2011.

[Mul12] Ketan D Mulmuley. The gct program toward the p vs. np problem. *Communications of the ACM*, 55(6):98–107, 2012.

[MV80] Silvio Micali and Vijay V Vazirani. An $\mathcal{O}(\sqrt{|V|}\cdot|\mathrm{E}|)$ algoithm for finding maximum matching in general graphs. In *21st Annual symposium on foundations of computer science (Sfcs 1980)*, pages 17–27. IEEE, 1980.

[Oum11] Sang-il Oum. Perfect matchings in claw-free cubic graphs. *The Electronic Journal of Combinatorics*, 18-1:P62–1–P62–6, 2011.

[Pan23] Greta Panova. Computational complexity in algebraic combinatorics, appeared in current developments in mathematics, 2023, 2023.

[Pet91] Julius Petersen. Die theorie der regulären graphs. *Acta Mathematica*, 1891.

[Raz85a] Alexander Razborov. Lower bounds on the monotone complexity of some boolean function. In *Soviet Math. Dokl.*, volume 31, pages 354–357, 1985.

[Raz85b] Alexander A Razborov. Lower bounds on monotone complexity of the logical permanent. *Mathematical Notes of the Academy of Sciences of the USSR*, 37:485–493, 1985.

[Rel24] Kunal Relia. On efficient computation of dire committees. *arXiv preprint arXiv:2402.19365*, 2024.

[Rot17] Thomas Rothvoß. The matching polytope has exponential extension complexity. *Journal of the ACM (JACM)*, 64(6):1–19, 2017.

[RR94] Alexander A Razborov and Steven Rudich. Natural proofs. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 204–213, 1994.

[Sbi80] Najiba Sbihi. Algorithme de recherche d'un stable de cardinalité maximum dans un graphe sans étoile. *Discrete Mathematics*, 29(1):53–76, 1980.

[Tur36] Alan M Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(1):230–265, 1936.

[Tur38] Alan M Turing. On computable numbers, with an application to the entscheidungsproblem. a correction. *Proceedings of the London Mathematical Society*, 2(1):544–546, 1938.

[Tut47] William T Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, 1(2):107–111, 1947.

[Val79a] Leslie G Valiant. Completeness classes in algebra. In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 249–261, 1979.

[Val79b] Leslie G Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2):189–201, 1979.

[Var10] Moshe Y Vardi. on p, np, and computational complexity. *Communications of the ACM*, 53(11):5–5, 2010.

[VL91] Jan Van Leeuwen. *Handbook of theoretical computer science (vol. A) algorithms and complexity.* Mit Press, 1991.

[Voo79] Marc Voorhoeve. A lower bound for the permanents of certain (0, 1)-matrices. In *Indagationes Mathematicae (Proceedings)*, volume 82-1, pages 83–86. Elsevier, 1979.

[Wes01] Douglas Brent West. *Introduction to graph theory*, volume 2. Pearson Prentice hall Upper Saddle River, 2001.

[Wig06] Avi Wigderson. P, np and mathematics–a computational complexity perspective. In *Proceedings of the ICM*, volume 6, pages 665–712, 2006.

[Wig09] Avi Wigderson. Knowledge, creativity and p versus np. *URL http://www. math. ias. edu/ avi/PUBLICATIONS/MYPAPERS/AW09/AW09. pdf. Circulated manuscript*, 2009.

[Wil14] Ryan Williams. Nonuniform acc circuit lower bounds. *Journal of the ACM (JACM)*, 61(1):1–32, 2014.

[Wil19] R Ryan Williams. Some estimated likelihoods for computational complexity. *Computing and Software Science: State of the Art and Perspectives*, pages 9–26, 2019.

[Wil25] R Ryan Williams. Simulating time with square-root space. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, pages 13–23, 2025.

[Yan88] Mihalis Yannakakis. Expressing combinatorial optimization problems by linear programs. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 223–228, 1988.

A Selection of Simple Connected Graphs

We hand-waved the selection of simple connected graphs for our study (e.g., see Footnote 2 and Footnote 19). Hence, we extensively discuss why this selection was worthy of hand-waving rather than a deeper discussion. Consequently, we reiterate that all our results are unconditional[47].

Let us zoom out to discuss our choice of the graph used in this paper from the family of graphs (2-uniform hypergraphs). Foremost, it is clear by now that we use cubic bridgeless graphs. As is the norm in literature, all graphs are unweighted undirected finite graphs. Additionally, w.l.o.g., we stated that the considered graphs are simple connected graphs. We clarify the last choice.

Connected Graphs: We first discuss the requirement that graphs are connected.

- **Graph Theory:** The assumption on the graph being connected is standard in graph theory literature about papers on matching theory. For instance, Line 35 on Page Number 843 (just below Theorem 1) in [Ber57] assumes the graph to be connected. This is because in the context of such work (and our paper), a technique that works for one connected component

[47] The use (of variations) of the words "assume" and "trivial" in Footnote 2 and Footnote 19 was bothersome. While their use in the context of the paper is appropriate, we add this discussion to the appendix to provide the reasons for our choice of simple connected graphs. This discussion reinforces that our results are indeed unconditional.

implicitly works for each connected component of a given graph, and in turn, for the entire graph, assuming all components share the common properties. Hence, when a theorem holds for a connected component of a given graph, it implies that it holds for the entire graph.

- **Computational Complexity:** Our results in both parts of the paper hold when we have a connected graph. Hence, if an unconnected graph is given, our results will hold for each connected component of the graph. We simply need to take the union of the outcomes of each connected component to get the overall outcome. For instance, executing Lines 1 to 10 of Algorithm 1 for each connected component and eventually taking the union of the vertex covers S we get in Line 10 indeed results in a minimum vertex cover.

Simple Graphs: We now discuss the requirement that graphs are simple. A simple graph, by definition, is undirected.

- **Graph Theory:** If a matching theory (graph theory) result holds for general graphs, it implies it will hold for the special case of simple graphs. Hence, each known result we use in the paper holds for simple graphs, too. Our results are particularly designed to hold for simple graphs.

- **Computational Complexity:** From the computational complexity perspective, we discussed how $\texttt{VC} - \texttt{CBG}$ can alternatively be proven to be **NP**-complete by reducing from the $\texttt{VC}$ on cubic simple graphs by using the same construction we discussed in Part I. Before that, the $\texttt{VC}$ on cubic simple graphs can be proven to be **NP**-complete by reducing from $\texttt{VC} - \texttt{CG}$.

  More specifically, if the given graph is not simple, we first remove each multiple edge and each loop. Then, we add an edge to each vertex that has a degree less than three, such that the edge connects the vertex to a subgraph of five dummy vertices (Figure 13). This ensures the graph remains cubic while becoming simple[48]. Each subgraph needs 3 vertices to form an MVC.

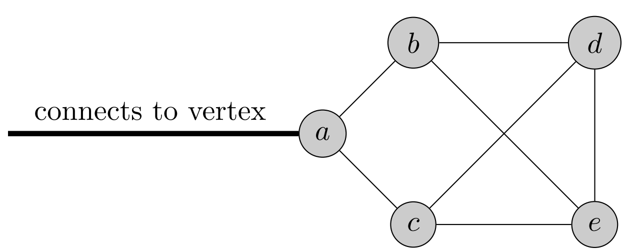

Figure 13: A subgraph of dummy vertices used to make a cubic graph simple.

In summary, in the context of this paper, all our computational complexity and graph-theoretic results hold when we use simple connected graphs. Overall, for the sake of completeness, the unconditional results in the paper use finite unweighted simple connected cubic bridgeless graphs.

[48] Alternatively, for each edge, split it and insert Block Type B_1 (but not Type B_2) as discussed in Part I.