

₁ On the Computational Complexity of the Vertex Cover Problem on
₂ Cubic Bridgeless Graphs

₃ Kunal Relia*

Independent Scientist

kunal.relia91@gmail.com / krelia@nyu.edu

₄ **Abstract**

₅ In this two-part study on the vertex cover problem on cubic bridgeless graphs ($\text{VC} - \text{CBG}$),
₆ we discover that: (I) $\text{VC} - \text{CBG}$ is NP -complete and (II) $\text{VC} - \text{CBG} \in \mathbf{P}$.

₇ **Part I** - We prove that $\text{VC} - \text{CBG}$ is NP -complete by showing that (i) $\text{VC} - \text{CBG}$ is in NP and (ii)
₈ $\text{VC} - \text{CBG}$ is NP -hard via a polynomial-time reduction from a known NP -hard problem, namely,
₉ the vertex cover problem on cubic graphs.

₁₀ **Part II** - In this core contribution of the paper, we discover an unconditional deterministic
₁₁ polynomial-time exact algorithm for $\text{VC} - \text{CBG}$. The algorithm is divided into three phases:

- ₁₂ • 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
₁₃ a perfect matching because we use a cubic bridgeless graph [Petersen, Acta Mathematica
₁₄ 1891].
- ₁₅ • 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
₁₆ the vertex cover problem by (i) selecting edges that are in the perfect matching and (ii)
₁₇ ordering the selection of the edges based on the tree-level of the vertices. This augmented
₁₈ 2-approximation algorithm populates a novel data structure called the “represents table”,
₁₉ which stores the information of each endpoint picked by the augmented algorithm and the
₂₀ neighbors of each endpoint. The endpoints form a vertex cover.
- ₂₁ • 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
₂₂ 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
₂₃ 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
₂₄ maximum), we establish the relation between the absence of a diminishing hop (local
property) and a minimum vertex cover (global minimum). Consequently, this relation
₂₅ forms the basis of Phase III of the algorithm, in line with Berge’s Theorem forming the
basis of the Blossom Algorithm.

*Blank for now.

33 Contents

| | | |
|----|--|-----------|
| 34 | 1 Introduction | 1 |
| 35 | 1.1 Vertex Cover and its Computational Aspects | 1 |
| 36 | 1.2 $P \stackrel{?}{=} NP$, Lower Bounds, Barriers, and the MVC | 3 |
| 37 | 1.3 Some Non-Computational Aspects of Graph Theory | 7 |
| 38 | 1.4 Section Summary | 9 |
| 39 | 2 High-level Overview of Results | 10 |
| 40 | 2.1 Part I: NP-completeness | 10 |
| 41 | 2.2 Part II: Algorithm | 11 |
| 42 | 3 Notation and Preliminaries | 12 |
| 43 | I VC – CBG IS NP-COMPLETE | 14 |
| 44 | 4 Proof of NP-completeness of VC – CBG | 14 |
| 45 | II VC – CBG $\in P$ | 23 |
| 46 | 5 Algorithm Overview and Intermediate Results | 25 |
| 47 | 5.1 Find a Perfect Matching | 26 |
| 48 | 5.2 Populate Represents Table | 27 |
| 49 | 5.3 Diminishing Hops | 34 |
| 50 | 5.4 Summary | 52 |
| 51 | 6 Algorithm | 53 |
| 52 | 7 Proof of Correctness | 59 |
| 53 | 8 Time Complexity Analysis | 63 |
| 54 | 9 Concluding Remarks | 70 |
| 55 | 9.1 Brief Additional Remarks | 70 |
| 56 | A Selection of Simple Connected Graphs | 76 |

57 1 Introduction

58 The $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ question [Coo71, Lev73, Kar72], directly and indirectly, is arguably one of the
59 biggest curiosities in computer science and mathematics. Informally, the question asks if every
60 computational problem whose solution can be *verified* in polynomial time can also be *solved* in
61 polynomial time. Formally, the complexity class \mathbf{P} consists of computational problems whose
62 solution can be computed in time polynomial in the size of input, which is considered “efficient”¹.
63 Alternatively, the problems in \mathbf{P} are a set of all languages that can be decided by a *deterministic*
64 Turing Machine [Tur36, Tur38] in polynomial time. On the other hand, the class \mathbf{NP} consists of
65 computational problems that, when given a candidate solution, we can verify in time polynomial in
66 the size of the input whether the candidate solution is correct or not. Hence, the problems in \mathbf{NP}
67 are a set of all languages that can be decided by a *non-deterministic* Turing Machine in polynomial
68 time (or that can only be verified by a *deterministic* Turing Machine in polynomial time). The
69 question of whether the problems in \mathbf{NP} can be computed efficiently or not forms the basis of $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$.
70

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

80 1.1 Vertex Cover and its Computational Aspects

81 Given an unweighted undirected graph (specifically, a 2-uniform hypergraph)², the vertex cover of
82 the graph is a set of vertices that includes at least one endpoint of every edge of the graph. Formally,
83 given a graph $G = (V, E)$ consisting of a set of vertices V and a collection E of 2-element subsets of
84 V called edges, the vertex cover of the graph G is a subset of vertices $S \subseteq V$ that includes at least one
85 endpoint of every edge of the graph, i.e., for all $e \in E$, $e \cap S \neq \emptyset$. The corresponding computational
86 problem of finding the minimum-size vertex cover (**MVC**) is \mathbf{NP} -complete³ (Node Cover, Problem
87 5 in [Kar72]). However, the hardness meant that there is no *known* unconditional deterministic
88 polynomial-time algorithm to solve **MVC** unless $\mathbf{P} = \mathbf{NP}$. Hence, a rich line of research ensued
89 that improved our general understanding related to the complexity of the **MVC** problem, especially
90 around its (in)approximability and parameterized complexity.

91 1.1.1 Approximation Algorithm and Inapproximability

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

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

²For 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).

³Strictly speaking, the decision version (**VC**) of the minimum-size vertex cover problem is \mathbf{NP} -complete whereas the **MVC** itself (search version) is \mathbf{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.

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

111 1.1.2 Parameterized Complexity

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

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

127 1.1.3 Tractability under Restricted Graphs

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

133
 134 *In summary*, there is neither a study to *significantly* improve the 2-approximation algorithm for
 135 the MVC nor a study on an algorithm for a restricted setting of the MVC that is **NP**-complete.
 136 Consequently, we shift our discussion to understanding how the research on resolving the **P** = **NP**

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

¹³⁷ question relates to the MVC. This is important especially because all **NP**-complete problems are
¹³⁸ “equivalent” in a certain technical sense. In particular, we assess if existing research either prohibits
¹³⁹ or limits the discovery of an algorithm for an **NP**-complete variant of the MVC.

¹⁴⁰ 1.2 $\stackrel{?}{=} \text{NP}$, Lower Bounds, Barriers, and the MVC

¹⁴¹ The $\stackrel{?}{=} \text{NP}$ question has arguably attracted unparalleled research in the number of approaches
¹⁴² to solving it. On the surface, there are three possible outcomes: (i) $\text{P} = \text{NP}$, (ii) $\text{P} \neq \text{NP}$, or (iii)
¹⁴³ $\text{P} \stackrel{?}{=} \text{NP}$ is unsolvable or undecidable. On zooming in, each outcome, especially the first two, is
¹⁴⁴ associated with multiple approaches. The third outcome has not received particular attention.

¹⁴⁵ To prove $\text{P} = \text{NP}$ using a constructive proof, we can either (a) discover a polynomial-time
¹⁴⁶ algorithm for an **NP**-complete problem or (b) prove that a problem in **P** is **NP**-complete. While
¹⁴⁷ the technique used for both approaches may be the same, the problem space being targeted is
¹⁴⁸ different. A constructive proof for $\text{P} = \text{NP}$, especially an algorithm with a lower order polynomial
¹⁴⁹ time complexity, would fundamentally reshape how we study complexity theory, as not only will
¹⁵⁰ all “hard” problems be in **P**, but there will be an algorithm to solve them all! A non-constructive
¹⁵¹ proof for $\text{P} = \text{NP}$ may not have similar major practical consequences. On the other hand, the aim
¹⁵² to prove $\text{P} \neq \text{NP}$, which is widely believed to be the case and would imply that the problems in
¹⁵³ **NP** cannot be computed efficiently, has led to multiple important breakthroughs in how and more
¹⁵⁴ importantly, how not to approach a proof for $\text{P} \neq \text{NP}$.

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

¹⁵⁶ There is an amazing breadth and depth of research focused towards proving $\text{P} \neq \text{NP}$. Yet,
¹⁵⁷ each major approach, ranging from the earlier logic-based techniques to the most recent Geometric
¹⁵⁸ Complexity Theory, has either hit at least one of the barriers in complexity theory or been stagnated.
¹⁵⁹ We discuss some of these approaches as arguments against a proof of $\text{P} \neq \text{NP}$ and then provide
¹⁶⁰ an overview of its relevance to the MVC.

¹⁶¹ Approaches that Hit a Barrier We enlist approaches that were negated by one of the three bar-
¹⁶² riers in complexity theory, namely, relativization [BGS75], natural proofs [RR94], or algebrization
¹⁶³ [AW09].

- ¹⁶⁴ • **Logical Techniques:** Initial research in TCS that aimed at separating the classes **P** and
¹⁶⁵ **NP** used techniques borrowed from logic and computability theory, especially previously
¹⁶⁶ successful techniques that were used for separation results. One such strong candidate was
¹⁶⁷ the diagonalization technique. However, Baker, Gill, and Solovay [BGS75] showed that these
¹⁶⁸ techniques that relativize cannot be used to solve the $\stackrel{?}{=} \text{NP}$ question. This is because
¹⁶⁹ a relativizing proof for $\text{P} \neq \text{NP}$ would mean that there exists another “relativized world”
¹⁷⁰ where **P** and **NP** Turing machines can compute a problem in polynomial time, even in a single
¹⁷¹ time stamp. Hence, there are relativized worlds where $\text{P} = \text{NP}$, and other relativized worlds
¹⁷² where $\text{P} \neq \text{NP}$. Therefore, any solution to the $\stackrel{?}{=} \text{NP}$ problem will require non-relativizing
¹⁷³ techniques.

- ¹⁷⁴ • **Proof Complexity and Circuit Lower Bounds:** The need for a non-relativizing approach
¹⁷⁵ led the researchers to turn to proof complexity and circuit complexity. The proof complexity
¹⁷⁶ approach would lead to counterintuitive results such that, with some additional work, one
¹⁷⁷ could prove $\text{P} \neq \text{NP}$, and even $\text{NP} \neq \text{coNP}$. 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 problem⁵ by a then-graduate student Razborov [Raz85a]. However, this hit a wall when an exponential lower bound for the matching problem⁶ 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 $\mathbf{P} = \mathbf{NP}$.

⁵Given 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.

⁶Given 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**.

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

221 “Stagnated” Approaches We now enlist approaches that have made progress but are stagnated.

- 222 • **Ironic Complexity Theory (ICT)**⁷: The term “ironic” in the name is apt - the ICT
223 program aims to assess whether an efficient algorithm for one problem can be used to show that
224 an efficient algorithm cannot exist for another problem. Conversely, is it possible that proving
225 the non-existence of an efficient algorithm for one problem implies that an efficient algorithm
226 solves another problem? At a high level, the ICT program aims to discover algorithms to
227 prove lower bounds. However, theoretically, such surprising results depend on collapse(s) in
228 the Time Hierarchy Theorem. Previous examples of positive results involve understanding,
229 say time-space complexity tradeoffs [LV99], to discover surprising algorithms and collapses
230 that do occur to establish new lower bounds! While examples of such amazing results are
231 there, especially by Williams (e.g., [Wil14]), the common denominator across all approaches
232 is that it will still require years of work.
- 233 • **Arithmetic Complexity Theory (ACT)**: The ACT program, a generalization of the tra-
234 ditional Turing Machine and Boolean circuits using Boolean values, uses arithmetic circuits,
235 which consider computer programs that use some larger field of values, such as real or com-
236 plex numbers instead of Boolean. Then, the task here is to find the minimum number of
237 operations needed to compute some polynomial over the chosen field of values. To that end,
238 the arithmetic complexity world analog of the $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ question is the permanent versus
239 determinant question for an $n \times n$ matrix⁸. It is known that the determinant is computable
240 in polynomial time but the permanent is $\#P$ -complete [Val79b]. This led to a remarkable line
241 of research on the study of the lower bounds for arithmetic circuits concerning this question.
242 However, all approaches fell short of resolving the question, mainly the Valiant Conjecture.
243 The reasons include the absence of a technique that works for permanent but fails for deter-
244 minant and a technique that circumvents the arithmetic variant of the natural proof barrier,
245 if there is one.
- 246 • **Geometric Complexity Theory (GCT)**: The GCT program was a once-promising ap-
247 proach started by Muliuley [Mul99] and forwarded along with Sohoni [MS01, MS08] and
248 others⁹. At a high level, it aims to resolve the $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ question via a resolution of Valiant’s
249 algebraic analog, the \mathbf{VP} vs \mathbf{VNP} conjecture [Val79a]. Importantly, GCT had the potential,
250 in part, because it overcame the three barriers. However, Panova recently discussed that the
251 study of Kronecker and plethysm coefficients has effectively stagnated the progress of the
252 GCT program [IP17, BIP19, DIP20]. In particular, for the GCT to progress, asymptotic
253 representation theoretic multiplicities need to be studied, which can then be used to under-
254 stand the computational complexity lower bounds [Pan23]. In summary, the GCT program,
255 as of 2025 and except for the *general* approach of resolving the permanent versus deter-
256 minant question (borrowed from previous approaches), has almost stagnated and is not a strong
257 contender to separate the complexity classes \mathbf{P} and \mathbf{NP} in the near future.

⁷We 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.

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

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

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

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

265 **1.2.2 Relevance to the MVC**

266 The research on proving $\mathbf{P} \neq \mathbf{NP}$ is connected to the MVC in two ways: (i) barriers in complexity
267 theory do not prevent an algorithm for the MVC and (ii) given that all \mathbf{NP} -complete problems are
268 “equivalent” in a certain technical sense, an exponential (or more generally, a super-polynomial)
269 lower bound for one of them (or for any analogous lower bound programs) would imply that the
270 MVC cannot be solved efficiently. However, no such lower bounds are known. In the other direction,
271 our proof of $\mathbf{P} = \mathbf{NP}$ wouldn't be a total disaster for lower bounds research, too. This is because
272 our result would adhere to the known theories as it will provide a polynomial upper bound to the
273 lower bounds instead of the current exponential upper bound. Also, while there are such indirect
274 implications, none of the studies, to the best of our knowledge, provide insight to *directly* improve
275 our understanding of the vertex cover problem¹¹.

276
277 *In summary*, given the state of research, we do not have a technical reason (“barrier”) that would
278 prohibit (“lower bound”) us from having a polynomial-time algorithm to solve an \mathbf{NP} -complete
279 problem, namely the MVC, let alone the VC – CBG. On the contrary, an algorithm for the VC – CBG
280 would validate our arguments and explain many current theories in computational complexity
281 literature! Hence, we now shift our discussion to some non-computational aspects of graph theory
282 with a focus on the research relevant to vertex covers and, specifically, on research that facilitates
283 the discovery of an algorithm for an \mathbf{NP} -complete variant of the MVC in Part II of the paper.

¹⁰We 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.

¹¹We stress that we are aware of some of the results that shed light on some of the \mathbf{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 VC – CBG is $\mathcal{O}(\text{poly}(m, n))$ where $\text{poly}(m, n)$ 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 $\mathcal{O}(68, 719, 476, 736 \cdot \text{poly}(m, n))$, (ii) 5-regular graphs would be $\mathcal{O}(322, 687, 697, 779 \cdot \text{poly}(m, n))$, (iii) 6-regular graphs would be $\mathcal{O}(3.108710029642957 \cdot 10^{16} \cdot \text{poly}(m, n))$, and so on. However, we leave this analysis for future work.

284 **1.3 Some Non-Computational Aspects of Graph Theory**

285 Since Euler (formally) introduced graphs to solve the Königsberg Bridge problem¹² in 1736 [Eul36],
286 the field of Graph Theory has evolved and found applications in various areas, including computer
287 science, medicine, and social science. Here, we discuss aspects of graph theory relevant to this
288 paper.

289 **1.3.1 Matching Theory**

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

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

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

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

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

¹²The 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.

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

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

333 1.3.2 Graph Theory and Vertex Cover

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

337 Matching and Vertex Cover Matching of a graph and the vertex cover of a graph are closely
338 related. Just like maximum matching, a minimum vertex cover always exists (by definition). Ad-
339 ditionally, given an arbitrary graph, the size of the minimum vertex cover is at least the size of the
340 maximum matching. In case of the existence of a perfect matching, the size of the minimum vertex
341 cover is at least $\frac{m}{2}$. If the graph is bipartite, then the size of the maximum matching is equal to
342 the size of the minimum vertex cover (Konig's Theorem [Kon31]). However, despite such numerical
343 relations between maximum matching and minimum vertex cover, there is no known structural
344 relation between the two¹³. Moreover, like Berge's Theorem is to maximum matching, there is no
345 such analog to the minimum vertex cover.

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

353
354 *In summary*, we focused our discussion on the existence of matchings in graphs (especially perfect
355 matching in a cubic bridgeless graph) and the theories that help us count the number of matchings
356 and the (non-computational, graph-theoretic) results that are used to find a matching (especially
357 Berge's Theorem for finding a maximum matching)¹⁴. Next, in the context of vertex cover, we
358 observed that the otherwise well studied cubic bridgeless graphs are not studied for the vertex
359 cover. Additionally, there is no analog of Berge's Theorem for the minimum vertex cover problem.
360 Hence, in this paper, we focus on understanding the non-computational aspects of the minimum
361 vertex cover in cubic bridgeless graphs. In turn, we study the complexity of the corresponding
362 computational problem of finding the minimum vertex cover in cubic bridgeless graphs.

¹³By 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.

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

363 **1.4 Section Summary**

364 We summarize the key observations:

- 365 • **Vertex Cover:** There is an extremely exciting line of work to understand the computational
366 complexity of the minimum vertex cover problem (MVC), especially around its (in)approximability.
367 However, no work aims to either find an algorithm for an **NP**-complete variant of the MVC
368 or aims to *significantly* reduce the factor 2 approximation successfully. (Also, given that we
369 dive deep into the differences between the variants of the MVC, we omitted the discussion on
370 the research progress of every other **NP**-complete problem as it is beyond the scope of this
371 paper, even when all **NP**-complete problems are “equivalent” in a certain technical sense¹⁵.)
- 372 • **P $\stackrel{?}{=}$ NP:** The rich breadth and depth of research surrounding an answer for the **P $\stackrel{?}{=}$ NP** question has resulted in (i) three barriers in complexity theory that do not allow easy
373 separation of complexity classes **P** and **NP**. Rather, a (constructive) proof for **P = NP** may
374 explain the existence of these barriers! (ii) The programs that explore the lower bounds and
375 that also overcome the barriers do not prohibit a polynomial-time for the MVC, let alone for
376 the restricted case of the MVC on cubic bridgeless graphs.
- 377 • **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.

383 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:

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

389 More specifically, we work on an understudied problem, the vertex cover problem on cubic
390 bridgeless graphs (VC – CBG). In Part I, we show that VC – CBG is **NP**-complete by reducing from
391 a known **NP**-complete problem, namely, the vertex cover problem on cubic graphs [GJS74, GJ02].
392 Next, in Part II, we present the core contribution of the paper: an unconditional deterministic
393 polynomial-time algorithm for the VC – CBG, which is spread over three phases ([Figure 1](#)). Phase I
394 leverages the knowledge of the existence of a perfect matching in every cubic bridgeless graph [Pet91]
395 to find a perfect matching for the given graph using the Blossom algorithm [Edm65]. Phase II uses
396 the perfect matching and a breadth-first search tree to create an augmented version of the (vanilla)
397 2-approximation algorithm for the vertex cover problem. This augmented algorithm populates a
398 novel data structure called the “represents table”, which stores the information of each endpoint
399 picked by the augmented algorithm and the neighbors of each endpoint. Phase III introduces a novel
400 technique called the “diminishing hops”. The use of a diminishing hop to find a minimum vertex
401 cover is analogous to the use of an augmenting path to find a maximum matching [Ber57]. The

¹⁵Given 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.

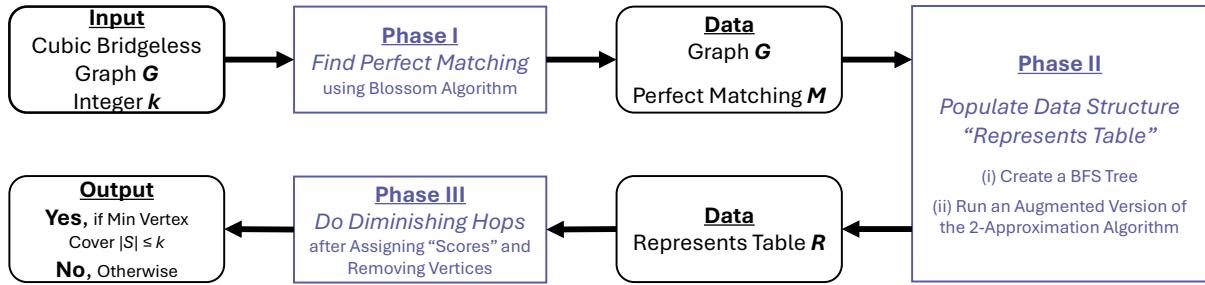


Figure 1: A schematic representation of the three phases of the algorithm, with the corresponding input and output data for each phase clearly indicated.

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

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

414 2 High-level Overview of Results

415 This paper is divided into two main results:

416 2.1 Part I: NP-completeness

417 In this part, we prove the following theorem:

418 **Theorem 1.** *The vertex cover problem on cubic bridgeless graphs (VC – CBG) is NP-complete.*

419 We prove the NP-completeness of VC – CBG in two steps: (i) we show that VC – CBG ∈ NP
 420 and (ii) we show that VC – CBG is NP-hard via a polynomial-time reduction from the vertex cover
 421 problem on cubic graphs, a known NP-hard problem. For the latter, the key idea in the reduction
 422 is the replacement of each edge, whether a bridge or not, with two blocks of dummy vertices.
 423 Specifically, we split the edge to insert one block of dummy vertices and we split each endpoint
 424 of the edge to insert another block of dummy vertices. We call this an “atomic operation”. The
 425 constructed gadget with the inclusion of dummy vertices (and corresponding edges that are part
 426 of at least one cycle by design) ensures that the edge from the given graph is also part of at least
 427 one cycle, which in turn implies, it is not a bridge. Each atomic operation results in one edge of
 428 the given graph guaranteed to be transformed into a bridgeless edge. Hence, we repeat an atomic
 429 operation for each edge in the graph. This ensures that the graph becomes cubic bridgeless. The

430 reduction is polynomial in the size of input. The proof of correctness ensues because of the mapping
431 between the edge and an atomic operation. We defer a more detailed discussion of the reduction
432 and the proof to [Section 4](#), which uses visualizations for easier understanding.

433 **2.2 Part II: Algorithm**

434 In this part, we prove the following main theorem:

435 **Theorem 2.** *The vertex cover problem on cubic bridgeless graphs (VC – CBG) is in P.*

436 We prove this theorem through discovery of an unconditional deterministic polynomial-time
437 exact algorithm for VC – CBG, which is a sequential implementation of three phases ([Figure 1](#)):

438 **2.2.1 Phase I: Find a Perfect Matching**

439 **Input:** a cubic bridgeless graph G

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

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

446 **2.2.2 Phase II: Populate Represents Table**

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

448 **Execution:** Phase II first constructs a tree from graph G by doing a breadth-first search of its
449 lexicographically sorted vertices. Then, it uses the tree and the perfect matching to augment the
450 2-approximation algorithm for the vertex cover problem. More specifically, instead of randomly
451 choosing the edges during each iteration of the the 2-approximation algorithm, Phase II selects
452 an edge (i) that is in perfect matching M and (ii) based on the order determined by the level on
453 which an endpoint of the edge is in the BFS tree. The output of this augmented 2-approximation
454 algorithm is a novel ordered unique data structure called the “Represents Table”.

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

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

467 **2.2.3 Phase III: Diminishing Hops**

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

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

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

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

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

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

502 **Output:** a minimum vertex cover S

503 **3 Notation and Preliminaries**

504 We kindly refer the reader to standard texts in theoretical computer science (TCS) for the definitions
505 of complexity classes **P** and **NP** and other definitions in complexity theory such as polynomial-
506 time reductions and **NP**-completeness. For example, for a lucid overview, please refer to Sections
507 2.1 and 2.2 in [[Kho19](#)], which is an interesting paper discussing foundational work leading to the

508 proof of the 2-to-2 Games Theorem. For comprehensive definitions and discussion, please refer to
509 a handbook of TCS [VL91] or to some of the standard TCS textbooks [KT06, GJ02, CLRS22].
510 We treat these standard definitions as done in the literature. Additionally, our algorithm treats a
511 graph as a set of vertices and a set of edges. Hence, throughout the paper, we perform standard
512 set operations on a given graph. Therefore, graph operations implicitly follow all standard set
513 theory laws (e.g., associative, commutative, etc.). See Appendix B of [CLRS22] for details on set
514 operations and laws. We now define the computational problems related to finding the vertex cover
515 of a given graph. First, we define the search/optimization problem:

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

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

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

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

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

528 **Definition 4** (Bridgeless Graphs). *A bridgeless graph, also known as a 2-edge-connected graph or
529 an isthmus-free graph or a graph with no cutedge, is a graph that does not contain an edge, called
530 a bridge¹⁶, whose deletion increases the number of connected components in the graph.*

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

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

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

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

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

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

I VC – CBG IS NP-COMPLETE

551 In this Part I of the paper, we establish the **NP**-completeness of the vertex cover problem on cubic
 552 bridgeless graphs (**VC – CBG**) by reducing from a known **NP**-complete problem, namely, the vertex
 553 cover problem on cubic graphs (**VC – CG**).

554 The vertex cover problem on cubic graphs is trivially known to be **NP**-complete because the
 555 vertex cover problem on graphs with vertex degree at most three is **NP**-complete [GJS74] (GJS74
 556 calls vertex cover as node cover). However, the hardness of the **VC** on cubic bridgeless graphs does
 557 not follow the hardness of the **VC** on cubic graphs. This is because the stipulation that the graph is
 558 bridgeless introduces a restriction to the family of cubic graphs being considered. Such structural
 559 restrictions are known to make the **VC** problem tractable. For example, the **VC** on claw-free graphs¹⁷
 560 is in **P** as a consequence of the maximum independent set problem on claw-free graphs being in
 561 **P** [Sbi80, Min80] (we refer the reader to a survey on claw-free graphs for more details [FFR97]).
 562 Hence, restricting the cubic graphs to being bridgeless requires us to establish its computational
 563 complexity. Furthermore, the reduction used to prove the hardness of the **VC** on graphs with vertex
 564 degree at most three in Theorem 2.4¹⁸ in [GJS74] does not necessarily consist of a bridgeless graph.
 565 Thus, its reduction cannot be used to establish the hardness of the **VC** on cubic bridgeless graphs.

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

572 **Part I Contribution and Organization:** We show that **VC – CBG** is **NP**-complete (Section 4).

573 4 Proof of NP-completeness of VC – CBG

574 For consistency, we restate the theorem we prove:

575 **Theorem (1 restated).** *The vertex cover problem on cubic bridgeless graphs (**VC – CBG**) is **NP**-
 576 complete.*

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

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

584 **$\text{VC} - \text{CG} \leq_P \text{VC} - \text{CBG}$** We reduce an instance of the vertex cover problem on cubic graphs (**VC – CG**)
 585 to an instance of the vertex cover problem on cubic bridgeless graphs (**VC – CBG**).

¹⁷A 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.

¹⁸Theorem 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.

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

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

- 591 • Block type B_1 consists of n blocks and each block consists of ten vertices, namely, vertex
592 $i \in B_1, \forall i \in [0, 9]$.
- 593 • Block type B_2 consists of m blocks and each block consists of six vertices. Specifically, for
594 each vertex $u \in G$, we have vertices $\{u', u'', u'_1, u'_2, u''_1, u''_2\} \in B_2$.

595 Hence, there are $10 \cdot n$ dummy vertices in blocks of type B_1 and $6 \cdot m$ dummy vertices in blocks of
596 type B_2 . Thus, $|D| = 10n + 6m$. Overall, we set $X = \{x_1, \dots, x_m\}$ and the dummy vertex set $D =$
597 $\{d_1, \dots, d_{6m+10n}\}$. Hence, the vertex set $V' = X \cup D$ is of size $|V'| = |X| + |D| = (m) + (6m + 10n) =$
598 $7m + 10n$ vertices.

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

600 **Edges:** Recall that (i) the target instance should form a cubic bridgeless graph and (ii) w.l.o.g.,
601 we have assumed the graphs being considered are simple connected, which means the given instance
602 of $\text{VC} - \text{CG}$ contains no loops, no multiple edges, and no unconnected components¹⁹. Therefore, to
603 get a bridgeless graph, we replace each edge $e = (u, v) \in E$ in graph G to ensure no bridges remain.
604 To do so, we first create a list L consisting of each edge and its endpoints $e = (u, v) \in E$. Next,
605 consider a snapshot of the graph G depicting an arbitrary edge $e \in E$ connecting vertices u and v in
606 the given instance of $\text{VC} - \text{CG}$. The endpoints u and v are connected to the vertices $\{a, b, c, d\} \in V$,
607 which are further connected to other vertices not depicted here or among themselves or both. We
608 are given a cubic graph, hence each snapshot centered on edge e looks as follows:

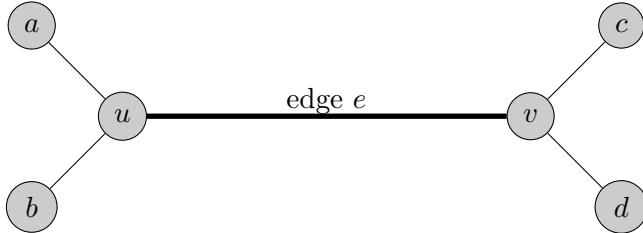


Figure 2: A snapshot of an instance of $\text{VC} - \text{CG}$ centered on an edge $e \in E$.

609 Next, for each edge $e = (u, v) \in L$ connecting vertices u and v in graph G of the instance
610 of $\text{VC} - \text{CG}$, we perform the following “atomic” operations²⁰ where we (i) split the edge e by

¹⁹W.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 VC on cubic simple graphs is NP -complete, which can then be used to prove that $\text{VC} - \text{CBG}$ 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.

²⁰Here, 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.

611 connecting it to a subgraph and (ii) split each endpoint u and v into three vertices each, as discussed
 612 below. This ensures that the graph remains cubic while becoming bridgeless²¹.

- 613 • **Splitting an Edge e :** The edge e connecting the vertices u and v in an instance of VC – CG is
 614 split and connected via a subgraph consisting of dummy vertices from Block type B_1 (yellow
 615 vertices) as depicted below:

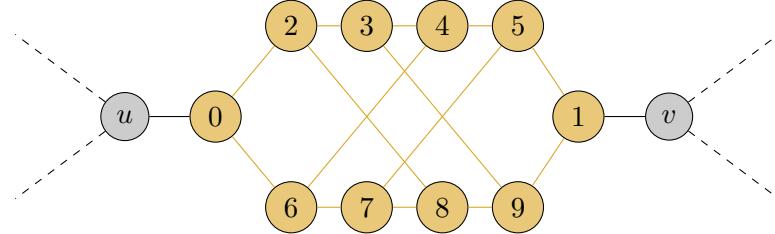


Figure 3: Splitting the edge e from the instance of VC – CG by inserting a subgraph of dummy vertices from Block type B_1 in the instance of VC – CBG. Each dashed line denotes the existence of an edge.

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



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

- 619 • **Updating the List L of Edges:** Remove the edge e connecting the endpoints u and v from
 620 the list L . Next, update the following edges in the list L :

- 621 – edge connecting endpoints a and u is replaced by an edge connecting endpoints a and
 622 u' where the endpoint u' is the newly created vertex from Block type B_2 , which was
 623 created by splitting the endpoint u

²¹In 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 encloses 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 VC – CG (Figure 2) into the following snapshot of the (sub-)graph of VC – CBG:

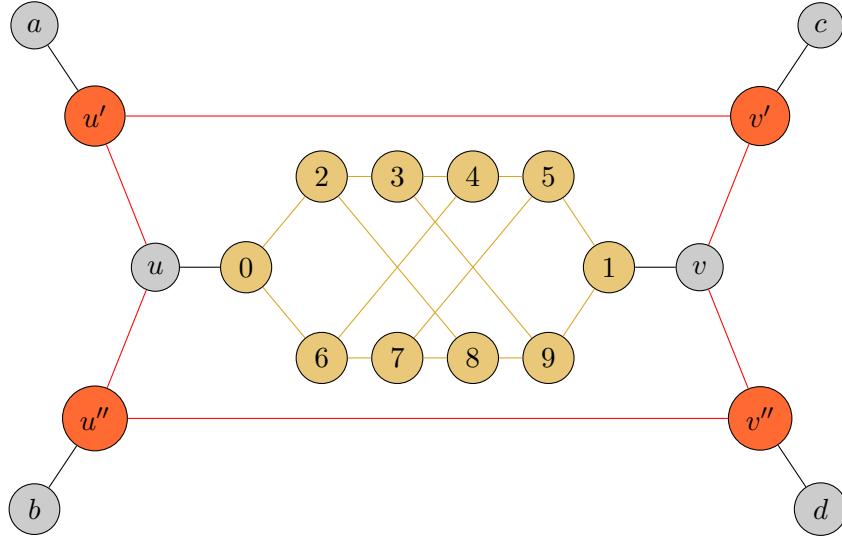


Figure 5: A snapshot of an instance of VC – CBG corresponding to the snapshot of VC – CG (Figure 2).

Finally, it remains to be discussed how a series of n “atomic” operations, one for each edge in the instance of VC – CG, leads to the complete construction of an instance of VC – 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 VC – CBG that is not a part of at least one cycle. In turn, it implies the constructed instance of VC – 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 VC – 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 .

- 651 • **Number of Dummy Vertices in Block Type B_2 is $6m$:** Each vertex has a degree of
 652 three. Hence, the operation of splitting of an endpoint occurs three times for each vertex.
 653 More specifically:

- 654 – Initially, each endpoint u is connected to three edges that will be split.
- 655 – Next, when the endpoint u is split (for the first time), the vertex u gets connected to
 656 two dummy vertices u' and u'' ([Figure 4](#)). By design, the vertex u is now *not* connected
 657 to any edge that will be split because (i) one of the edges it was connected to got split
 658 and (ii) each of the remaining two edges that need to be split are now connected to the
 659 dummy vertices u' and u'' , respectively.
- 660 – The dummy vertex u' is connected to one edge that needs to be split. Hence, when that
 661 edge is split, the endpoint u' is split, and in turn, it is connected to two dummy vertices
 662 u'_1 and u'_2 . By design, the dummy vertex u' is now not connected to any edge that will
 663 be split. Simultaneously, the two dummy vertices u'_1 and u'_2 are also not connected to
 664 any edge that will be split.
- 665 – The dummy vertex u'' is also connected to one edge that needs to be split. Hence, when
 666 the edge is split, vertex u'' is split and connected to two dummy vertices u''_1 and u''_2 , in
 667 line with what was done for dummy vertex u' .
- 668 – Overall, each vertex u is split into 6 dummy vertices $\{u', u'', u'_1, u'_2, u''_1, u''_2\}$. There are m
 669 vertices in total, which results in $6m$ dummy vertices.

- 670 • **Each Edge in the Constructed Instance of VC – CBG is a Part of At Least One
 671 Cycle:** During an “atomic” operation, when the two endpoints of an edge is split ([Figure 4](#)),
 672 the resultant dummy vertices are connected such that (i) they form a cycle and (ii) they form
 673 a boundary around (a) the endpoints that were split and (b) the 10 dummy vertices from
 674 Block type B_1 that were inserted to split the edge. The combination of points (i) and (ii)
 675 ensures that the edges connecting the endpoints that were split and the edges connecting the
 676 10 dummy vertices from Block type B_1 are also part of a cycle. Thus, an “atomic” operation
 677 guarantees that each new edge is part of a cycle. Consequently, after n “atomic” operations,
 678 each edge will be part of at least one cycle. The reason for this is mainly that there’s at least
 679 one edge that belongs to the boundary resulting from one “atomic” operation and also to the
 680 boundary resulting from another. This fact can be interpreted in two ways: (i) two cycles
 681 share at least one edge in common or (ii) each “atomic” operation enlarges the boundary to
 682 encompass all the newly inserted edges. In either case, it means that no edge is a bridge
 683 in the constructed instance of VC – CBG. Therefore, the reduction ensures that the graph is
 684 bridgeless.
- 685 • **Each Vertex in the Constructed Instance of VC – CBG has Degree Three:** When a
 686 vertex is split, the split vertex and the corresponding dummy vertices are connected to three
 687 other vertices. When an edge is split, there are two dummy vertices in the subgraph of Block
 688 type B_1 that are connected to the two endpoints of the split edge, and each of the remaining
 689 eight dummy vertices is connected internally with three other vertices. Hence, trivially, each
 690 vertex has a degree of three.

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

693 We refer the reader to [Figure 6](#), which depicts a snapshot of the constructed instance of VC – CBG
 694 when each of the five edges depicted in the snapshot of the instance of VC – CG ([Figure 2](#)) is split.

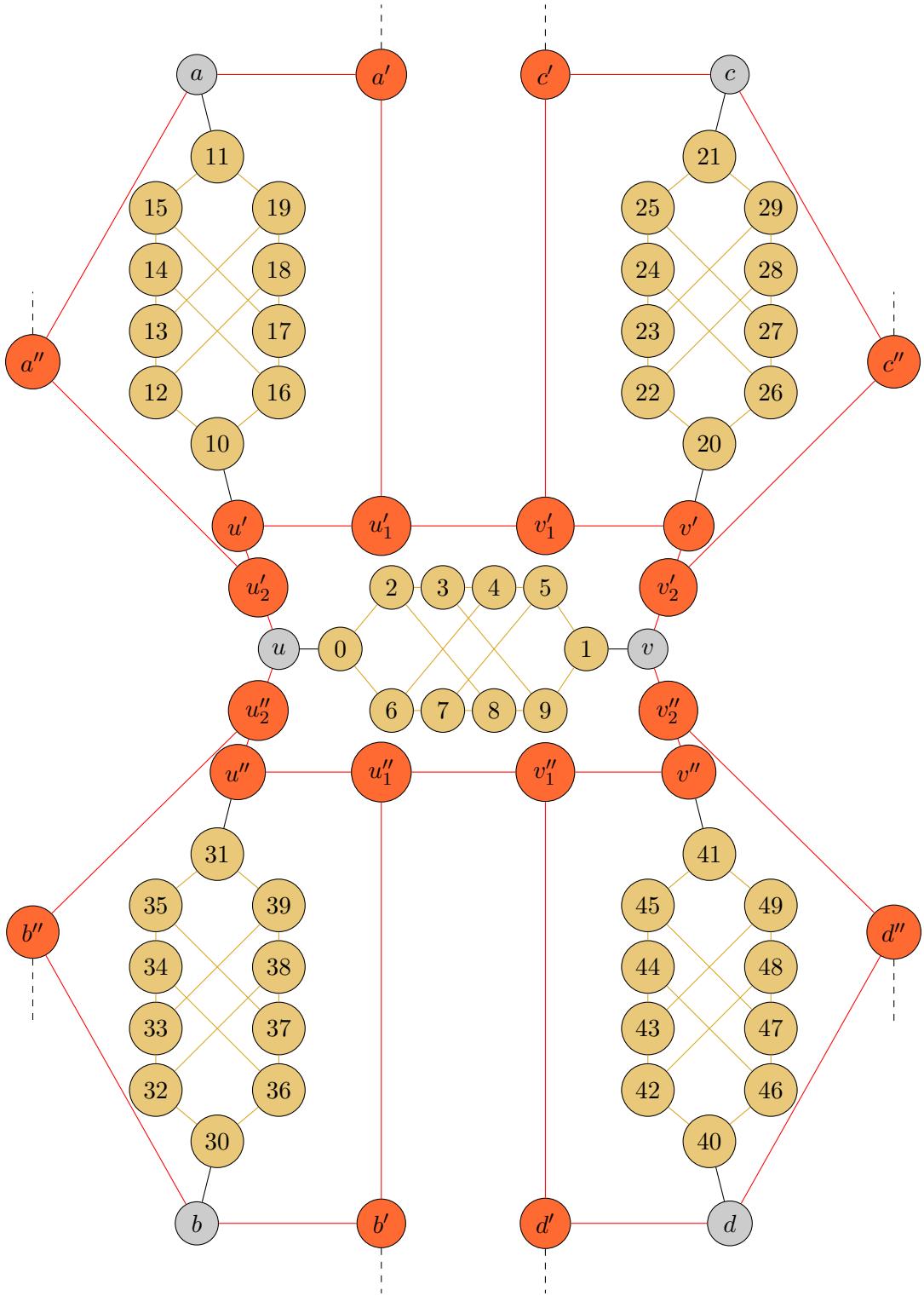


Figure 6: A snapshot of an instance of VC – CBG that corresponds to an instance of VC – CG (Figure 2). The construction depicts the transformation of five edges (and corresponding six vertices) shown in Figure 2; the edges that were not depicted are not transformed. Each dashed line denotes the existence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

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

697 **B. Proof of Correctness.**

698 **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
699 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
700 edges $e' \in E'$.*

701 (\Rightarrow) If the instance of the VC – CG problem is a yes instance, then the corresponding instance
702 of VC – CBG is a yes instance.

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

709 **Three Vertices are Selected from each of the $m - k$ Blocks of type B_2 that Correspond
710 to $m - k$ Vertices Not in Vertex Cover S of VC – CG:** For each vertex u not in the vertex
711 cover S of an instance of VC – CG, we have three vertices in the vertex cover S' of an instance of
712 VC – CBG. Specifically, for each vertex $u \notin S$, we have vertex u and corresponding dummy vertices
713 u' and u'' in the vertex cover S' . We know that for each vertex $u \in G$, we split it and have
714 $\{u, u', u'', u'_1, u'_2, u''_1, u''_2\} \in G'$. These seven vertices form a linear chain. Additionally, the vertices
715 u , u' , and u'' are all connected to either a vertex ending with 0 or a vertex ending with 1 from one
716 of the corresponding blocks of Block type B_1 . This means that vertices u , u' , and u'' may or may
717 not cover edges connecting them to vertices ending with 0 or vertices ending with 1 because the
718 latter two are already in the vertex cover S' . Finally, given that vertex $u \notin S$, all three vertices
719 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
720 in graph G' need not cover any edges that are not part of the linear chain (more on this in the next
721 paragraph). Therefore, we use the following proposition on the minimum vertex cover on a linear
722 chain graph:

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

725 The seven vertices form a linear chain. Therefore, the minimum size vertex cover for the seven
726 vertices is of size three. We choose (central) vertices u , u' , and u'' to form the minimum vertex
727 cover. Given that there are $m - k$ vertices that are not in the vertex cover S , it corresponds to
728 $3 \cdot (m - k)$ vertices in the vertex cover S' . This results in an additional **3m – 3k** vertices in Vertex
729 Cover S' of the instance of VC – CBG.

730 **Four Vertices are Selected from each of the k Blocks of type B_2 that Correspond to k
731 Vertices in Vertex Cover S of VC – CG:** For each vertex u in the vertex cover S of an instance
732 of VC – CG, we have four vertices in the vertex cover S' of an instance of VC – CBG. Specifically, for
733 each vertex $u \in S$, we have corresponding dummy vertices u'_1 , u'_2 , u''_1 , and u''_2 in the vertex cover
734 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 VC – 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 VC – CG implies a yes instance of the VC – CBG such that the vertex cover S' is of size at most $k + 3m + 6n$.

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

²²This 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 .

779 its neighbor without affecting the size of the minimum vertex cover. Similarly, in our case,
 780 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
 781 chain, which allows manipulation of vertices selected in the vertex cover, even when the linear
 782 chain for us is of odd size. We discuss two points in this case:

- 783 (a) **three vertices from the linear chain are in the vertex cover S' :** In the case of a
 784 linear chain of size seven (odd), the minimum vertex cover is of size three, and neither
 785 of the vertices at the end of the linear chain can be in the minimum vertex cover. There
 786 are $m - k$ such linear chains where three vertices are in the minimum vertex cover S' .
 787 The $m - k$ vertices not in the minimum vertex cover S correspond to these linear chains.
- 788 (b) **four vertices from the linear chain are in the vertex cover S' :** When four vertices
 789 from a linear chain are in the minimum vertex cover S' , it implies that at least one of the
 790 vertices is included to cover an edge that is not in the linear chain. W.l.o.g., let us begin
 791 with an assumption that the vertices $\{u, u', u''_1, u''_2\}$ (one vertex from the set X and three
 792 vertices from the Block type B_2) are a subset of vertices that is in the vertex cover S' .
 793 Then, these vertices can be replaced by the vertices $\{u'_1, u'_2, u''_1, u''_2\}$ (four vertices from
 794 Block type B_2) in the vertex cover S' because the vertices ending with 0 and vertices
 795 ending with 1 from Block type B_1 is in the vertex cover S' and the corresponding edges
 796 need not be covered by vertices in the linear chain. In general, when four vertices from
 797 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*
 798 such combination of the four vertices can be replaced by vertices $\{u'_1, u'_2, u''_1, u''_2\}$ (four
 799 vertices from Block type B_2). The instance of **VC – CBG** remains a yes instance with
 800 this modification. There are k such linear chains having four vertices in the minimum
 801 vertex cover S' . The k vertices in the minimum vertex cover S correspond to these linear
 802 chains.

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

808 These cases complete the other direction of the proof of correctness. In turn, it completes the
 809 overall proof that shows **VC – CBG** is **NP**-complete. \square

II VC – CBG ∈ P

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

814 **(Relative) Lack of Research on the VC – CBG is Surprising:** The matching theory within
 815 graph theory has received much attention. In particular, properties of a perfect matching and a
 816 maximum matching in a given graph have been studied extensively. Furthermore, matching in
 817 cubic bridgeless is also well-studied (see section 1.3). However, no analogous work that extensively
 818 discusses properties of the vertex cover²³, and in particular, properties of the vertex cover on cubic
 819 bridgeless graphs is known. This is surprising because there is a known relationship between the
 820 sizes of a maximum matching and a minimum vertex cover for a given graph (Lemma 1). Hence, it
 821 is intuitive to explore the existence of a deeper relation between the two. Additionally, a minimum
 822 vertex cover always exists (by definition), just like a maximum matching. Hence, an analog to
 823 Berge’s Theorem [Ber57], which relates augmenting paths and maximum matching, should be
 824 explored.

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

831 **No Algorithm for VC – CBG is Unsurprising:** While the lack of focus on understanding the
 832 properties of vertex cover, analogous to, say, Berge’s Theorem for maximum matching, is surpris-
 833 ing, the lack of an algorithm for the VC and VC – CBG is unsurprising. Karp’s landmark paper
 834 on the twenty-one NP-complete problems brought the vertex cover problem (VC) to the atten-
 835 tion of TCS researchers [Kar72]. Consequently, given that VC was proven to be NP-complete,
 836 understanding its hardness-related computational aspects has been a focus of TCS researchers
 837 (see subsection 1.1)²⁴. Additionally, one of the natural approaches to discover an algorithm
 838 for an NP-complete problem (and prove P = NP) directly relies on finding a polynomial-size
 839 Linear Programming or Semidefinite Programming that projects onto the polytope whose ex-
 840 treme points are the valid solutions. This approach was ruled out through a series of results
 841 [Yan88, FMP⁺15, LRS15, CLRS16, Rot17, CŽ24]. Hence, no effort on this front to find an algo-
 842 rithm for an NP-complete problem is unsurprising.

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

²³For 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 “VC” (and its variants) refers to a discussion of the computational properties.

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

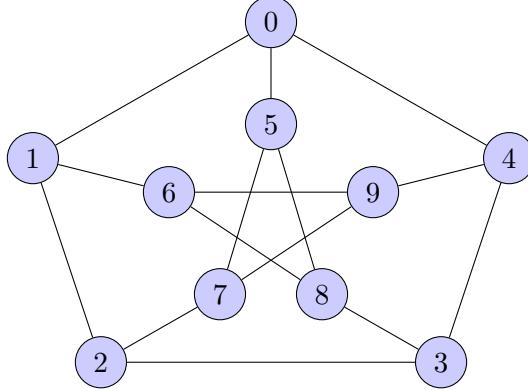


Figure 7: Petersen Graph used for the running example.

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 **VC** – **CBG** is unsurprising because: (i) **VC** was proven **NP**-complete early and subsequent research focused on its hardness. (ii) **VC** – **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 **VC** and an absence of an algorithm to solve the **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 **VC** – **CBG** (and **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 (**VC** – **CBG**) is in **P**.

Part II Contribution: We show that **VC** – **CBG** ∈ **P**.

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

²⁵Recall that the **VC** on 2-regular graphs is in **P** and the **VC** on 3-regular graphs is **NP**-complete. Hence, the **VC** on 3-regular graphs is the closest known problem to a variant of the **VC** in **P**. Next, if we consider this imaginative problem space between the **VC** on 2-regular graphs and the **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 **VC** that is closest to the variant in **P** in the most conceivable way possible.

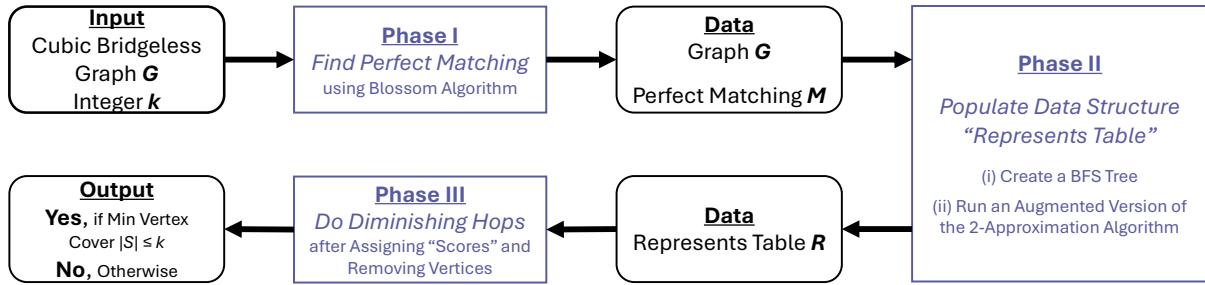


Figure 8: Figure 1 redrawn for ease of reference: A schematic representation of the three phases of the algorithm, with the corresponding input and output data for each phase clearly indicated.

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]).

²⁶The 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.

897 We now discuss each of these three phases in detail.

898 5.1 Find a Perfect Matching

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

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

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

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

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

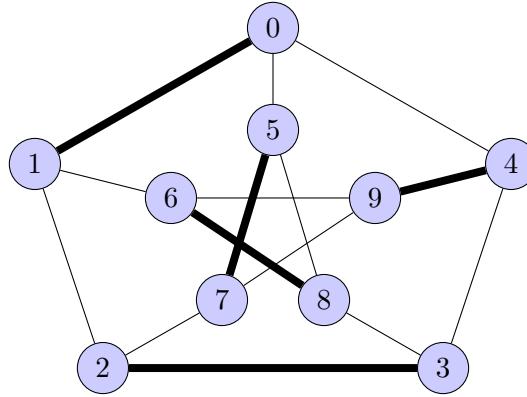


Figure 9: The bold edges of the Petersen Graph denote a perfect matching M .

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

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

914 **Theorem 3** (Petersen's Theorem [[Pet91](#)]). *Every cubic bridgeless graph contains a perfect match-
915 ing.*

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

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

923 Lemma 1 means that the largest number of edges in a matching does not exceed the smallest
 924 number of vertices in a cover. We use this fact to set a lower bound on the size of the minimum
 925 vertex cover. More specifically, we terminate the algorithm early if the given integer k is less
 926 than $|M|$. Moreover, given that the matching M is a perfect matching in our case, we know that
 927 $|M| = \frac{|V|}{2}$, which in turn implies that $|S| \geq \frac{|V|}{2}$.

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

934 5.2 Populate Represents Table

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

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

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

944 **Example 2.** *We are given the Petersen graph G (Figure 7) and a seed vertex $0 \in V$. Hence, the
 945 BFS algorithm seeded on vertex 0 will return the following table regarding the level at which each
 946 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} |

947 5.2.1 Augmented 2-Approximation Algorithm for the VC

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

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

953 In other words, a matching M is maximal if we cannot add any new edge $e \in E$ to the
 954 existing matching M . Next, recall that finding a maximal matching is equivalent to the vanilla
 955 2-approximation algorithm, which guarantees a vertex cover of size at most twice the size of the

956 minimum vertex cover: the algorithm picks an *arbitrary* edge $e = (u, v) \in E$, adds both the vertices
 957 u and v to the vertex cover S , removes all edges connected to either of the two vertices (u and
 958 v), and repeats until no edge remains. The vertex S is the resultant vertex cover. In this vanilla
 959 version, the method in which the edges are picked is *arbitrary* from two perspectives: (i) the *order*
 960 in which edges get picked is arbitrary, and (ii) consequently, *which* edge among the remaining edges
 961 gets picked is arbitrary. These two perspectives may seem similar but are different as outlined in
 962 the two steps discussed in the next paragraph.

963 We remove the above-mentioned arbitrariness in the edges that are picked. Specifically, during
 964 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 next²⁷. 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 .

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

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

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

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

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

²⁷The 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.

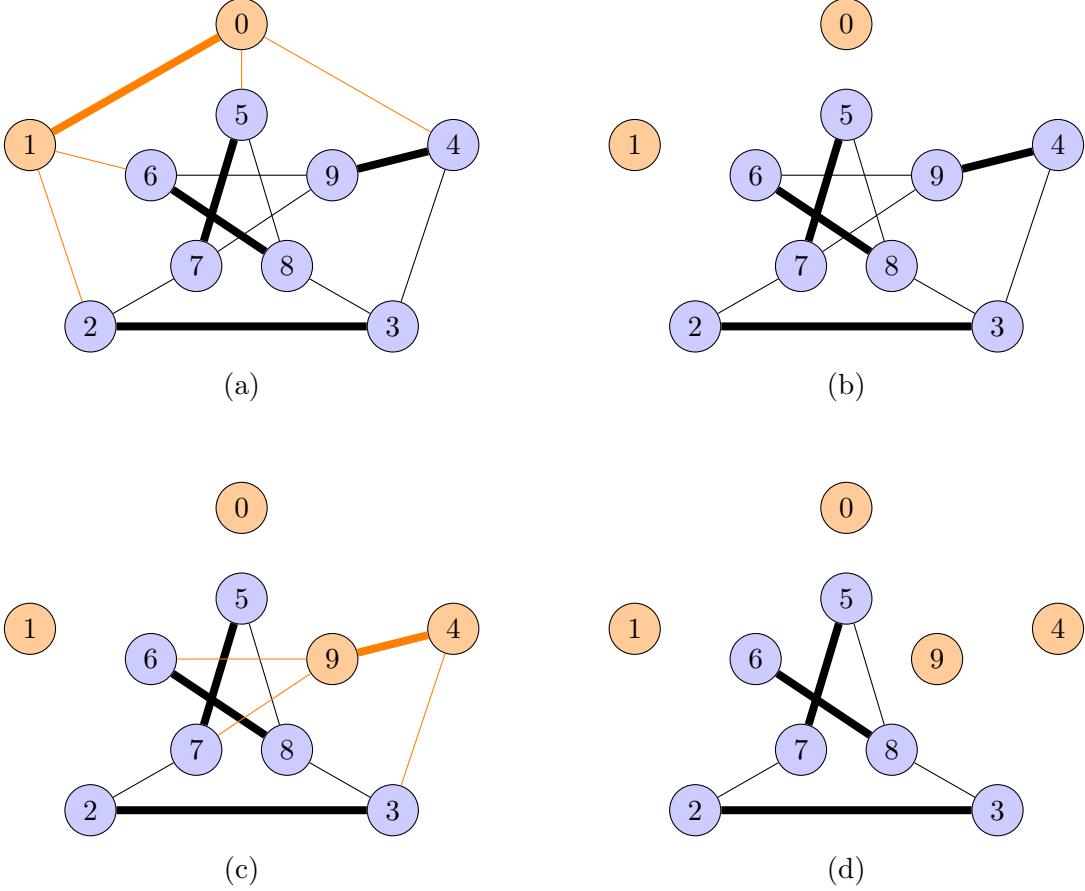


Figure 10: (a) Vertex 0 is on the Level 1 of the BFS tree. Hence, an edge in the perfect matching M that is connected to the vertex 0 is picked. Therefore, the edge connecting vertices 0 and 1 is picked. (b) All the edges connected to the two endpoints are removed. (c) Vertices 4 and 9 are the two endpoints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

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

1000 Note that due to Petersen's Theorem ([Theorem 3](#)), a perfect matching and a maximum matching
1001 mean the same thing in the case of cubic bridgeless graphs.

1002 **Lemma 3.** *The endpoints of a maximal matching form a vertex cover.*

1003 Overall, given that the augmented 2-approximation algorithm picks edges that are in a perfect
1004 matching, we know that the edges form a maximal matching too. Hence, its endpoints, which
1005 consist of all the vertices in the graph, form a vertex cover (trivially). Importantly, the augmented
1006 2-approximation algorithm picks the edges in a particular order. This does not alter the above
1007 discussion but is crucial in how the represents table gets populated and affects its properties.

1008 **5.2.2 Represents Table**

1009 We discuss the novel data structure called the “Represents Table”, which is a concept borrowed
 1010 from its brief mention in [Rel24]. It is an augmented version of a table data structure and consists
 1011 of unique properties and operations. Let us first define the property that led to the name *represents*
 1012 table. It is based on a concept where a vertex u that is connected to a vertex v via an edge is said
 1013 to represent²⁸ the other vertex.

1014 **Definition 12** (Represents). *Given a graph G , a vertex $u \in V$ is said to **represent** a vertex
 1015 $v \in V$ when the vertex v is connected to the vertex u by an edge $e \in E$. Conversely, the vertex v is
 1016 **represented by** the vertex u .*

1017 Observe that when a vertex u represents a vertex v , it is an alternative way of saying that an
 1018 edge connects the vertices u and v . Additionally, given that we use a cubic (bridgeless) graph, each
 1019 vertex represents three vertices and each vertex is represented by three vertices. The list of vertices
 1020 that a vertex u represents is stored in a list called a Represents List.

1021 **Definition 13** (Represents List). *Given a graph G , a vertex $u \in V$ is said to represent a set of
 1022 vertices $V' \subseteq V \setminus \{u\}$ if there exists an edge between the vertex u and every vertex in V' . These
 1023 vertices that the vertex u represents are in the represents list L_u such that for all vertices $u \in V$,
 1024 $L_u = \bigcup_{e \in E} e \setminus \{u\} \mid u \in e$*

1025 In the context of this paper, we can restate the above definition as follows: Given a cubic graph
 1026 G , a vertex $u \in V$ that is connected to three vertices x, y , and z by an edge each is said to represent
 1027 the vertices x, y , and z . These vertices that vertex u represents are in the represents list L_u such
 1028 that $L_u = \{x, y, z\}$. The size of each represents list in this paper is at most three (because we use
 1029 cubic graphs). We are now ready to define the represents table:

1030 **Definition 14** (Represents Table). *A represents table R is a 4-column table where a row stores
 1031 the two endpoints of an edge picked during an iteration of the execution of the augmented 2-
 1032 approximation algorithm, and for each endpoint u , also stores the corresponding represents list
 1033 L_u , which consists of the vertices the endpoint u represents.*

1034 **Example 4.** We are given the Petersen graph G (Figure 9), a perfect matching M and the levels
 1035 of a BFS tree seeded on vertex 0 (Table 1). As discussed in Example 3, the first iteration of the
 1036 augmented 2-approximation algorithm picks the edge connecting the vertices 0 and 1 and removes
 1037 all the edges that are connected to the two endpoints of the picked edge (Figure 10a and Figure 10b).
 1038 Then, the corresponding entry in the represents table R is as shown in Table 2.

1039 The first endpoint, namely vertex 0 represents vertices 1, 4, and 5. The second endpoint, namely
 1040 vertex 1 represents vertices 0, 2, and 6.

1041 At this point, one may argue that the represents table is our fancy way of renaming an adjacency
 1042 list. However, given the differences between their properties, we avoid using the latter term to avoid
 1043 the confusion and to ensure that the represents table is visualized as a data structure that is different
 1044 from an adjacency list. More specifically, unlike the adjacency list where each row enlists all the
 1045 vertices connected to a vertex, the represents table stores the two endpoints of a picked edge in

²⁸Informally, 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 | $L_1 = \{0, 2, 6\}$ |

1046 the *same* row. The corresponding represents list enlists *only* the vertices that are connected to an
 1047 endpoint during a *given iteration* of the augmented 2-approximation algorithm. The stress on the
 1048 words *given iteration* signifies that for a vertex to be listed in the represents list, an edge connecting
 1049 the vertex and the endpoint should not have been removed during any of the previous iterations of
 1050 the 2-approximation algorithm.

1051 **Example 5.** We continue the discussion from [Example 4](#) where we had populated the first row of
 1052 the represents table ([Table 2](#)).

1053 The second iteration of the augmented 2-approximation algorithm picks the edge connecting the
 1054 vertices 4 and 9 ([Figure 10c](#)). Note that the represents list for vertex 4 enlists the vertices 9 and
 1055 3. Vertex 0 is not listed because the edge connecting vertices 0 and 4 was removed during the first
 1056 iteration. The represents list for the vertex 9 is $L_9 = \{4, 6, 7\}$. Next, the algorithm now removes
 1057 all the edges that are connected to the two endpoints of the picked edge ([Figure 10d](#)).

1058 Similarly, the represents table is populated until the augmented 2-approximation algorithm ter-
 1059 minates. 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 | $L_8 = \{6\}$ |

1060 The example shows that each vertex of Petersen graph is listed as an endpoint in the represents
 1061 table ([Table 3](#)). Similarly, given a cubic bridgeless graph, each vertex of a given cubic bridgeless
 1062 graph is listed as an endpoint in the represents table. This is because a cubic bridgeless graph has
 1063 a perfect matching ([Theorem 3](#)). Therefore, each vertex is an endpoint of an edge in the perfect
 1064 matching. Hence, the augmented 2-approximation algorithm will enlist each vertex as an endpoint
 1065 in the represents table.

1066 **Lemma 4.** Given a cubic bridgeless graph, each vertex v of the graph is listed as an endpoint in
 1067 the corresponding represents table.

1068 **Operations and Properties of the Represents Table:** We now discuss the operations that
 1069 are supported by the represents table and discuss the properties relevant to this paper.

1070 **Operations** The represents table supports four basic operations, namely, insert, access, freeze, and
 1071 remove. The represents table does *not* support deletion of any information, as will become evident
 1072 during the discussion of the diminishing hops phase of our algorithm.

1073 • **insert:** The most basic operation supported by the table is the insertion of a vertex and its
 1074 corresponding represents list. Additionally, the insertion operation does not need to access
 1075 previous data. Hence, the asymptotic running time for each insert operation is $\mathcal{O}(1)$. We
 1076 witnessed this operation while populating the represents table.

1077 • **access / search:** (i) To access an endpoint u in the represents table, we do a sequential search
 1078 for the given endpoint. This takes $\mathcal{O}(m)$. (ii) To access the represents list of an endpoint
 1079 u , we need to first access the endpoint u , which takes $\mathcal{O}(m)$. Subsequently, accessing the
 1080 represents list L_u takes $\mathcal{O}(1)$. (iii) To access each of the represents list that consists of a
 1081 vertex u , we need to traverse through each of the m represents lists, each of constant size
 1082 (three). This takes $\mathcal{O}(m)$. Hence, the asymptotic running time for each access operation is
 1083 $\mathcal{O}(m)$.

1084 • **freeze:** The freeze operation is used to freeze an endpoint. In the context of this paper, when
 1085 we freeze an endpoint, it implies that the frozen vertex is selected as one of the vertices in the
 1086 vertex cover. Next, whenever an endpoint u is frozen, it is simultaneously delisted from each
 1087 of the represents lists it is a part of. This is analogous to marking every edge that touches
 1088 the vertex u chosen for the vertex cover as being covered. Finally, the entire represents list
 1089 L_u of the vertex u is delisted.

1090 Freezing an endpoint u takes $\mathcal{O}(m)$ time as we need to do a sequential search for the endpoint
 1091 u . The delisting of the vertex u from each of the represents lists also takes $\mathcal{O}(m)$: for each
 1092 of the m endpoints, we need to traverse through its represents list of size at most three and
 1093 delist the vertex u from the represents list if present. The delisting of the represents list L_u
 1094 takes $\mathcal{O}(1)$. Hence, the asymptotic running time for each freeze operation is $\mathcal{O}(m)$.

1095 • **remove:** The remove operation is used to remove an endpoint. In the context of this paper,
 1096 when we remove an endpoint u , it implies that the removed endpoint is *not* selected as one
 1097 of the vertices in the vertex cover. Hence, each of the three vertices that are connected to
 1098 the removed endpoint needs to be in the vertex cover to cover the edges that are connected
 1099 to the removed vertex. Hence, the corresponding steps carried out in the represents table are
 1100 as follows: each vertex in the represents list of the removed endpoint u and each vertex that
 1101 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 |

1102 The removal of an endpoint u takes $\mathcal{O}(m)$ time as we need to search for the endpoint u using
1103 a sequential search. The freeze operation will be carried out three times and each one takes
1104 $\mathcal{O}(m)$. Hence, the asymptotic running time for each remove operation is $\mathcal{O}(m)$.

1105 These are the main operations that can be carried out on the represents table (Table 4). The
1106 space complexity of the table is $\mathcal{O}(m)$.

1107 Properties We discuss unique properties of the represents table, which are essential for our dis-
1108 cussion on the phase three (diminishing hops) of our algorithm.

1109 **Property 1.** *Given a represents table R , an endpoint u in the i^{th} row of the table R can only
1110 represent an endpoint v if the endpoint v is in row j of the table R , for all $1 \leq i \leq j \leq \frac{m}{2}$.
1111 Conversely, an endpoint u in the j^{th} row of the table R can be represented by an endpoint v only if
1112 the endpoint v is in the i^{th} row of the table R , for all $1 \leq i \leq j \leq \frac{m}{2}$.*

1113 **Property 1** discusses that an endpoint in an earlier row within the represents table can only
1114 represent endpoints in the same row or any later row. It cannot represent endpoints in rows that
1115 come before it. Conversely, an endpoint in a later row within the represents table can only be
1116 represented by endpoints in the same row or any earlier row. It cannot be represented by endpoints
1117 in rows that come after it. Overall, these describe a “directional” relationship within represents
1118 table. Endpoints in earlier rows can represent endpoints in the same or later rows, and conversely,
1119 endpoints in later rows can be represented by endpoints in the same or earlier rows.

1120 **Property 2.** *Given a represents table R , endpoints u and v in the i^{th} row of the table R always
1121 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}
1122 row for all $1 \leq i \leq \frac{m}{2}$.*

1123 **Property 2** discusses that endpoints in the same row of the represents table *always* represent
1124 each other. **Property 2** adheres to **Property 1** and **Property 2** can be considered a specific case of
1125 **Property 1**. However, neither of the properties implies the other. Moreover, when we combine these
1126 two properties, they imply three possibilities about the representations related to an endpoint u in
1127 the table R when a graph is cubic : (i) u is represented by two other endpoints in rows above itself
1128 and does not represent any endpoint in rows below itself, (ii) u is represented by one endpoint in
1129 rows above itself and it represents one endpoint in rows below itself, and (iii) u is represented by
1130 zero endpoints in rows above itself and represents two endpoints in rows below itself.

1131 **Property 3.** *Given a represents table R , an endpoint $u \in R$ corresponds to a vertex $u \in V$ of
1132 the corresponding graph G , and the set of endpoints in the represent table R forms a vertex cover
1133 $S \subseteq V$ of the corresponding graph G .*

1134 **Property 3** refers to the simple one-to-one mapping of an endpoint in the represents table R
1135 and a vertex in the corresponding graph G . Additionally, in the general case, the endpoints in
1136 the represents table R form a vertex cover. This is because the endpoints of edges picked during
1137 maximal matching form a vertex cover. In our case of cubic bridgeless graphs, we always have a
1138 perfect matching and hence, all vertices of G will be an endpoint in the represents table R and
1139 these trivially form a vertex cover (because all vertices of a graph form a vertex cover).

1140 **Property 4.** *Given a represents table R , if each endpoint $u \in R$ is either frozen or removed, then
1141 the frozen endpoints form a vertex cover $S \subseteq V$ of the corresponding graph G .*

1142 Property 4 discusses the specific case when all the endpoints of the represents table R are
1143 either frozen or removed, and specifically the frozen endpoints form a vertex cover. The frozen
1144 endpoints form a vertex cover, so we focus our discussion on the removed endpoints. By design,
1145 when an endpoint is removed, all endpoints that it represents or is represented by are automatically
1146 frozen. This implies that no edge in the corresponding graph remains uncovered. Hence, the frozen
1147 endpoints will form a vertex cover when every endpoint is either frozen or removed. The frozen
1148 vertices do not form a vertex cover when at least one endpoint is neither frozen nor removed. This
1149 is because we can freeze, say, $m - 2$ endpoints and not touch the remaining 2 endpoints. The frozen
1150 endpoints may not form a vertex cover because the remaining 2 endpoints may be connected via
1151 an edge. Hence, the condition that each endpoint in the represents table R be either frozen or
1152 removed is necessary for the frozen endpoints to form a vertex cover.

1153 Overall, the operations performed on the represents table result in the above-discussed unique
1154 properties of the represents table. These operations and properties form the foundation for the
1155 discussion of the next phase of the algorithm.

1156 *In summary*, in the Phase II of the algorithm, given a cubic bridgeless graph G , a list of
1157 lexicographically sorted vertices V_{sort} , and a perfect matching M as input, we create a BFS tree,
1158 run an augmented version of the 2-approximation algorithm for the VC, and populate a novel
1159 data structure called the represents table R . The output of this phase is the represents table
1160 R . Additionally, we discussed how the represents table was created and listed its operations and
1161 properties:

- 1162 1. We began with an underlying data structure, a table.
- 1163 2. We used the BFS tree and the augmented 2-approximation algorithm to collect specific in-
1164 formation needed to populate the represents table.
- 1165 3. We discussed the corresponding insert operation that is used to enter the information into
1166 the represents table. We also discussed the access, freeze, and remove operations.
- 1167 4. We analyzed the time complexity of each operation²⁹.
- 1168 5. We observed some unique properties of the represents table.

1169 5.3 Diminishing Hops

1170 The diminishing hops phase of the algorithm constitutes the core contribution of the paper. To
1171 understand the concept of diminishing hops and its relevance to the minimum vertex cover, we first
1172 introduce the concept of “representation scores” and use it to decide whether to freeze or remove
1173 an endpoint from the represents table. We then discuss augmenting paths and its relevance to the
1174 maximum matching (Berge’s theorem [Ber57]), which serves as a motivation to finally introduce
1175 diminishing hops for the minimum vertex cover (Figure 11).

1176 5.3.1 Representation Score

1177 The idea for using Representation Score is to associate a score with each endpoint in the represents
1178 table R such that the score of each endpoint is used to decide whether to freeze an endpoint or
1179 remove it. The score is a quantification of the information related to each edge that connects the

²⁹We 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



Diminishing Hops for Minimum Vertex Cover

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

1180 vertices of the graph. More specifically, the score of an endpoint is a weighted number that captures
1181 how well an endpoint is represented by other endpoints in the table.

1182 The assignment of the score begins from the top row of the represents table R . The score
1183 assigned to an endpoint u is the sum of the scores of the endpoint that is on the same row as
1184 each endpoint that represents u . For instance, consider that the endpoint u is in the i^{th} row of the
1185 represents table R , for some integer $i > 1$. Next, if some endpoint x in row j , for all $j \in [1, i - 1]$,
1186 represents the endpoint u , then the score of endpoint y that is in the same row j as endpoint x is
1187 added to the score of endpoint u plus one. The score of each endpoint is initialized to zero.

1188 **Definition 15** (Representation Score). *The representation score ζ of an endpoint u in row i of the
1189 represents table R is denoted by*

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

1190 where endpoint y is in the same row as endpoint x for all endpoints x such that $u \in L_x$ and $y \neq u$.
1191 By design, the endpoint y will be in row j for some integer j such that $1 \leq j < i$.

1192 The higher the score of endpoint y , the higher the chance of endpoint u being frozen so that
1193 endpoint x can, in turn, be removed. Recall that both the endpoints in the first row have a score
1194 of zero each and the computation moves downward.

1195 **Example 6.** We use the populated represents table constructed in [Example 5](#).

1196 The representation score of both endpoints in the first row is 0. Hence, $\zeta_0 = \zeta_1 = 0$.

1197 There are two endpoints in the second row, namely 4 and 9. The endpoint 9 is not represented
1198 by any endpoint, which implies its score $\zeta_9 = 0$. The endpoint 4 is represented by endpoint 0.
1199 Hence, its score will be equal to the score of endpoint 1 (plus 1) because endpoint 1 is in the same
1200 row as endpoint 0. This means $\zeta_4 = \zeta_1 + 1 = 0 + 1 = 1$. Similarly, the representation score ζ will
1201 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 | $L_8 = \{6\}$ | $\zeta_8 = 7$ |

1202 We jump to calculate the representation score of the last endpoint in the last row, namely,
 1203 endpoint 8. The endpoint 8 is represented by endpoints 3 and 5. Hence, its score will be equal
 1204 to the sum of scores of endpoints 2 and 7 (plus 1 for each endpoint). This is because endpoint
 1205 2 is in the same row as endpoint 3 and endpoint 7 is in the same row as endpoint 5. Hence,
 1206 $\zeta_8 = (\zeta_2 + 1) + (\zeta_7 + 1) = (3 + 1) + (2 + 1) = 4 + 3 = 7$.

1207 Finally, while computing the representation score, we traverse through the entire represents
 1208 table by exploiting the fact that an endpoint u can be represented by at most two endpoints in
 1209 rows above itself. This is because the vertex degree of each vertex is three and one endpoint is in the
 1210 same row as endpoint u . Hence, the time complexity of computing the score is linear. We discuss
 1211 the details about the algorithm to compute scores and its complexity in [Section 6](#) and [Section 8](#),
 1212 respectively.

1213 5.3.2 Freezing and Removing Endpoints using Representation Score

1214 The representation scores are used to determine which endpoints to freeze or remove. The process³⁰
 1215 of freezing or removing an endpoint begins from the bottom row of the represents table R . There
 1216 are two endpoints in the last row, u and v . We freeze the endpoint with a higher representation
 1217 score ζ . Ties are broken using lexicographic ordering of the vertices. Simultaneously, we remove
 1218 the other endpoint. Formally:

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

1219 Recall that when an endpoint u is frozen, it is delisted from each represents list it is in. Therefore,
 1220 for each endpoint $a \in R$, $L_a = L_a \setminus u$. Simultaneously, all entries in the represents list of u are
 1221 delisted. Hence, $L_u = \emptyset$. Next, when an endpoint v is removed, all entries in the represents list
 1222 of v is delisted ($L_v = \emptyset$) and each endpoint that represents the endpoint v is frozen. Finally, the
 1223 representation scores of each endpoint in the remaining rows is recalculated top-down. The process
 1224 now iteratively moves up row-wise of the represents table R . This process of freezing and removing
 1225 endpoints of each row continues until each endpoint in the represents table R is either frozen or
 1226 removed. However, during an iteration, when an endpoint in a given row is already frozen, the other
 1227 endpoint is automatically removed. On the other hand, when an endpoint in a given row is already
 1228 removed, the other endpoint, by design, would have been frozen. Finally, when both endpoints are
 1229 frozen, no action is performed. In either case, the algorithm moves to the next iteration without the
 1230 need to use the representation score. The case when both the endpoints are removed is impossible.

1231 **Example 7.** We use the represents table with representation scores constructed in [Example 6](#).

1232 The process of freezing or removing an endpoint begins from the bottom row. Here, there are
 1233 two endpoints, namely 6 and 8, having representation scores of $\zeta_6 = 3$ and $\zeta_8 = 7$.

1234 First, freeze endpoint 8 because it has a higher representation score ($\zeta_8 > \zeta_6$ ($7 > 3$)). Then,
 1235 delist the endpoint 8 from the represents lists L_3 and L_5 . Also delist the entire represents list L_8 .

1236 Next, remove endpoint 6. Then, freeze the endpoints 1 and 9 because the removed endpoint
 1237 6 is in the represents lists L_1 and L_9 . Delist the entire represents list L_6 . Additionally, because
 1238 endpoints 1 and 9 are frozen: (i) delist the endpoint 1 from the represents list L_0 and delist the
 1239 endpoint 9 from the represents list L_4 , and (ii) delist the entire represents lists L_1 and L_9 .

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

1240 Finally, recalculate the representation score of all the endpoints that are neither frozen nor
 1241 removed. The represents table at the end of the first iteration, carried on the bottom row, looks as
 1242 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 |
|------------------------------|------------|---------------------|------------|---------------------|------------------------------|
| | 1 | List 1 | 2 | List 2 | |
| $\zeta_0 = 0$ | 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 | $L_8 = \{6\}$ | $\zeta_8 = 7$ |

1243 At the end of all iterations, the endpoints 0, 3, 6, and 7 are removed from the represents table
 1244 R . The endpoints 1, 2, 4, 5, 8, and 9 are frozen in the represents table R .

1245 At the end of the process of freezing or removing endpoints using the represents table R , the
 1246 frozen endpoints correspond to the vertices in a vertex cover S' of the graph G . We stress that at
 1247 this moment, we do *not* make any claim about the size of the vertex cover.

1248 **Theorem 4.** *Given a graph G consisting of a set V of m vertices and the corresponding represents
 1249 table R populated by a set W of m endpoints, a set S'' of l endpoints in the represents table R is
 1250 frozen, for some integer $1 \leq l \leq m$, and a disjoint set $W \setminus S''$ of $m - l$ endpoints in the represents
 1251 table R is removed if and only if there is a vertex cover S' of l vertices in the graph G that correspond
 1252 to the set S'' of frozen endpoints.*

1253 *Proof.* (\Rightarrow) If a set S'' of l endpoints in the represents table R is frozen, for some integer $1 \leq l \leq m$,
 1254 and a disjoint set $W \setminus S''$ of $m - l$ endpoints in the represents table R is removed, then there is a
 1255 vertex cover S' of l vertices in the graph G that correspond to the set S'' of frozen endpoints.

1256 When the represents table is populated, all the vertices in G are listed as endpoints. Hence,
 1257 the endpoints trivially form a vertex cover ([Property 3](#)). Next, the computation of representation
 1258 scores and the consequent process of freezing and removal of the endpoints results in each endpoint
 1259 either being frozen or removed. Hence, the frozen endpoints form a vertex cover ([Property 4](#)).

1260 (\Leftarrow) If there is a vertex cover S' of l vertices in the graph G , for some integer $1 \leq l \leq m$, then
 1261 a set S'' of l endpoints in the represents table R is frozen that correspond to the vertex cover S'
 1262 and a disjoint set $W \setminus S''$ of $m - l$ endpoints in the represents table R is removed. This is the
 1263 straightforward case, as we can simply freeze the endpoints that correspond to the vertices in the
 1264 vertex cover S' and remove the rest of the endpoints. \square

1265 Again, we stress that the above-stated theorem guarantees that the frozen endpoints form a
 1266 vertex cover and does not give any guarantee regarding the size of the vertex cover. We leave the
 1267 analysis on the size of the vertex cover derived using representation scores for future work. Overall,
 1268 here, we discussed the use of representation scores to freeze or remove each endpoint in the given
 1269 represents table, and the resultant frozen endpoints form a vertex cover. We finally discuss how
 1270 the represents table, representation score, and the vertex cover are used in the diminishing hops.

1271 **5.3.3 Diminishing Hops**

1272 The concept of diminishing hops is inspired by the use of augmenting paths for maximum matching
 1273 ([Figure 11](#)). More specifically, we prove a theorem on the use of diminishing hops for minimum
 1274 vertex cover, just like Berge's theorem is proven on the use of augmenting paths for maximum
 1275 matching [Ber57]. To do so, we first discuss an augmenting path and its relation to a maximum
 1276 matching proven through Berge's theorem. Then, we introduce a diminishing hop and show its
 1277 relation to the minimum vertex cover through a theorem (analogous to Berge's theorem).

1278 [Berge's Theorem, Augmenting Paths, and Maximum Matching](#) The Blossom Algorithm [[Edm65](#)]
 1279 is a polynomial-time algorithm to find a maximum matching in a given graph. The key concept
 1280 that the Blossom algorithm relies upon is Berge's theorem:

1281 **Theorem 5** (Berge's Theorem [[Ber57](#)]). *Given a graph G and a matching M , M is a maximum
 1282 matching if and only if there is no M -augmenting path in the graph G .*

1283 To understand this, we first define an alternating path and an augmenting path.

1284 **Definition 16** (Alternating Path). *Given a graph G and a matching M , an alternating path P
 1285 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
 1286 the matching M and (ii) whose edges alternate between not being in M and being in M (or being
 1287 in M and not being in M if the path starts from a vertex v that is incident to any edge e in the
 1288 matching M).*

1289 **Definition 17** (Augmenting Path). *Given a graph G and a matching M , an augmenting path is
 1290 an alternating path w.r.t. matching M that starts from a vertex v and ends at a vertex u such that
 1291 $u \neq v$ and neither the vertex u or the vertex v is incident to any edge e in the matching M .*

1292 An augmenting path's characteristic is that it increases the size of an existing matching. Hence,
 1293 if there is an augmenting path P w.r.t. a matching M , then we can increase the size of the matching
 1294 by one. To do so, we create a new matching M' by flipping the edges along the path P such that
 1295 (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'
 1296 ([Figure 12](#)). Formally, the new matching M' can be denoted by $M' = (M \setminus P) \cup (P \setminus M)$ and hence,
 1297 $|M'| = |M| + 1$. Berge's theorem uses this characteristic to prove that a matching M is maximum
 1298 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.

1299 We practically described an algorithm to find a maximum matching: start with an initial
 1300 matching (even a blank matching), augment the current matching with augmenting paths, and
 1301 terminate when no augmenting path remains. The eventual augmented matching is a maximum
 1302 matching. Subsequently, the key contribution of the Blossom algorithm is to find augmenting paths
 1303 efficiently.

1304 Similarly, this section first introduces the concept of a diminishing hop and then shows its
 1305 relation to a minimum vertex cover. Finally, we discuss how to compute a diminishing hop efficiently
 1306 in [Section 6](#) (Algorithm) and analyze its time complexity in [Section 8](#).

1307 Diminishing Hop A represents table R consists of vertices V in graph G that are endpoints of
 1308 edges in a maximum matching M found using the Blossom algorithm. In our case (of using cubic
 1309 bridgeless graphs), all vertices of a given graph G will be endpoints in the represents table because
 1310 a perfect matching always exists ([Lemma 4](#)). Next, if each endpoint in the represents table is either
 1311 frozen or removed, then the frozen endpoints form a vertex cover ([Property 4](#)). Conversely, if a
 1312 vertex cover S of a graph is given, then the corresponding endpoints in the represents table can be
 1313 frozen and the rest removed (i.e., endpoints in S can be frozen and endpoints in $V \setminus S$ removed).
 1314 Finally, in our case, the represents table consists of $\frac{m}{2}$ rows, which is also the lower bound on the
 1315 size of the MVC.

1316 Given that $\frac{m}{2}$ corresponds to the lower bound of an MVC and to the number of rows in the
 1317 represents table, we ideally want exactly one endpoint from each row frozen and the other removed.
 1318 Removing both endpoints of a given row is not possible by design. Hence, if there is a row in the
 1319 represents table that consists of two frozen endpoints, called a duad, then it should be assessed
 1320 if removing one of them can lead to a smaller number of frozen endpoints in the table. This
 1321 corresponds to a smaller-sized vertex cover. Importantly, recall that $\frac{m}{2}$ is a lower bound and not
 1322 the size of the MVC. Hence, it is perfectly possible for more than one row of the represents table to
 1323 have both its endpoints frozen. The aim here is to assess whether decreasing the number of such
 1324 rows is possible or not.

1325 **Definition 18** (Daud). *Given a represents table R where each endpoint u in the represents table
 1326 R is either frozen or removed, a duad³¹ refers to a pair of endpoints that (i) is in the same row of
 1327 the represents table R and (ii) are both frozen.*

1328 Consider a represents table R such that each endpoint is either frozen or removed and *exactly*
 1329 one of its rows has both its endpoints frozen (a duad). The set of frozen endpoints corresponds to
 1330 a vertex cover S . We shall generalize this discussion to when *at least* one duad exists later on by
 1331 successively doing diminishing hops³². Hence, for now, given a represents table R where (i) each
 1332 endpoint is either frozen or removed, (ii) each endpoint is marked “unvisited”, and (iii) there is one
 1333 duad, we carry out the following sequence of operations:

- 1334 1. Endpoints u and v in row i are both frozen, for some integer $i \in [1, \frac{m}{2}]$. Both are marked
 1335 “unvisited”. Hence, we start the hopping phase by choosing an endpoint to remove, say u .
- 1336 2. Remove endpoint u . Consequently, by design, each endpoint x is frozen such that either (i) x
 1337 is represented by endpoint u or (ii) x represents endpoint u . Mark endpoints u and v of row
 1338 i and each endpoint x as “visited”. Enqueue endpoint u in a queue Q . Subsequently,
 - 1339 (a) For all integers $j \in [i+1, \frac{m}{2}]$, consider an endpoint x in row j such that x is represented
 1340 by endpoint u . If the j^{th} row has two frozen endpoints x and y , repeat Step 2 by choosing
 1341 to remove endpoint y if endpoint y is marked “unvisited”. Move to Step 2(b) when either
 1342 an endpoint marked “visited” is encountered or endpoint u represents no endpoint or
 1343 only one endpoint in row j is frozen and “visited”.

³¹The 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.

³²Indeed, 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.

- 1344 (b) Dequeue an endpoint u from queue Q . For all integers $j \in [1, i - 1]$, consider an endpoint
 1345 x in row j such that x represents endpoint u . If the j^{th} row has two frozen endpoints x
 1346 and y , repeat Step 2 by choosing to remove endpoint y if endpoint y is marked “unvis-
 1347 ited”. If no endpoint represents endpoint u or when an endpoint marked as “visited” is
 1348 encountered or only one endpoint of row j is frozen and “visited”, then either (i) repeat
 1349 Step 2(b) if the queue Q is non-empty or (ii) move to Step 3 if the queue Q is empty.
- 1350 3. Count the number of marked frozen vertices S_u resulting from removing endpoint u . Mark
 1351 all the endpoints as “unvisited” and restore the represents table to the starting state (as was
 1352 given before Step 1). Repeat Step 2 by removing endpoint v . Subsequently, count the number
 1353 of marked frozen vertices S_v resulting from removing endpoint v .
- 1354 4. The vertex cover S is the initial set of frozen endpoints in the given represents table R . Hence,
 1355 if $|S| \leq |S_u|$ and $|S| \leq |S_v|$, then do nothing. Else if $|S_u| \leq |S_v|$, then the diminished vertex
 1356 cover is S_u ; else the diminished vertex cover is S_v .

1357 We informally stated the algorithm for diminishing hops for the restricted case when we are
 1358 given a represents table where each endpoint is either frozen or removed, and when there is exactly
 1359 one duad. We formalize it in [Section 6](#). Here, we discuss how a diminished vertex cover implies
 1360 a minimum vertex cover, and when a given set of frozen endpoints is not diminishable, the given
 1361 vertex cover is indeed the minimum. We first give an example:

1362 **Example 8.** We use the represents table that results after freezing or removing each endpoint
 1363 discussed in [Example 7](#). Recall that the endpoints 0, 3, 6, and 7 are removed from the represents
 1364 table R and the endpoints 1, 2, 4, 5, 8, and 9 are frozen (gray endpoints and red endpoints,
 1365 respectively, [Table 7](#)). The latter corresponds to a vertex cover. Also, observe that endpoints 4
 1366 and 9 in row 2 are both frozen. In other words, endpoints 4 and 9 in row 2 form a duad (yellow
 1367 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 | $L_8 = \{6\}$ | $\zeta_8 = 7$ |

1368 Next, as per Step 2, remove endpoint 4. Hence, row 2 now has only one frozen endpoint. Mark
 1369 both the endpoints as “visited” (green highlight, [Table 8](#)). Enqueue endpoint 4 to the queue $Q = \{4\}$.
 1370 By design, endpoints 0 and 3 are frozen. This is because endpoint 3 is represented by endpoint 4
 1371 (see list L_4) and endpoint 0 represents endpoint 4 (see list L_0). Consequently, endpoints 0 and 3
 1372 are marked as “visited” (green highlight).

1373 Next, the hopping across the table begins. As depicted in [Table 8](#), first hop to row 4 that contains
 1374 endpoint 3 (as per Step 2(a)). Now, as a consequence of freezing endpoint 3, row 4 consists of two
 1375 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 |
|------------------------------|------------|---------------------|------------|---------------------|------------------------------|
| | 1 | | 2 | | |
| $\zeta_0 = 0$ | 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 | $L_8 = \{6\}$ | $\zeta_8 = 7$ |

1376 one of the unvisited endpoints from row 4. Therefore, hop to endpoint 2 to remove it (i.e., repeat
1377 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 |
|------------------------------|------------|---------------------|------------|---------------------|------------------------------|
| | 1 | | 2 | | |
| $\zeta_0 = 0$ | 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 | $L_8 = \{6\}$ | $\zeta_8 = 7$ |

1378 We removed endpoint 2 (as per Step 2). Mark endpoint 2 as “visited” (green highlight, Table 10).
1379 Enqueue endpoint 2 to the queue $Q = \{4, 2\}$. Next, because endpoint 2 was removed, freeze and
1380 mark endpoints 7 and 1 as “visited” (green highlight, Table 10). Consequently, note that row 3 now
1381 has a duad because endpoint 7 was frozen. Next, we hop to rows where endpoint 2 is represented by
1382 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 |
|------------------------------|------------|---------------------|------------|---------------------|------------------------------|
| | 1 | | 2 | | |
| $\zeta_0 = 0$ | 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 | $L_8 = \{6\}$ | $\zeta_8 = 7$ |

1383 At this point, no “unvisited” endpoint represents endpoint 2 or is represented by endpoint 2
1384 (Table 11). Hence, as per Step 2(a), we move to Step 2(b). De-queuing queue $Q = \{4, 2\}$ gives us
1385 endpoint 4 and therefore $Q = \{2\}$. Endpoint 4 is represented by one endpoint, namely endpoint 0

1386 in row 1. We hop to row 1. Both the endpoints of row 1 are marked “visited” (and are frozen).
1387 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 | $L_8 = \{6\}$ | $\zeta_8 = 7$ |

1388 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.
1393

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 | $L_8 = \{6\}$ | $\zeta_8 = 7$ |

1394 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).
1396

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 | $L_8 = \{6\}$ | $\zeta_8 = 7$ |

1397 We removed endpoint 5 (as per Step 2). Mark it “visited” ([Table 14](#)) . Enqueue endpoint 5 to
 1398 the queue $Q = \{5\}$. Consequently, freeze and mark endpoints 0 and 8 as “visited”. Next, we first
 1399 hop to the row where endpoint 5 represents an endpoint, namely, endpoint 8 in row 5. We then hop
 1400 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 | $L_8 = \{6\}$ | $\zeta_8 = 7$ |

1401 More specifically, first hop to row 5 that contains endpoint 8 (as per Step 2(a)). Row 5 consists
 1402 of one frozen endpoint, which is marked “visited”. Note that endpoint 6 was already removed and
 1403 hence, we need not visit it as only one endpoint from its row is frozen. Additionally, by design,
 1404 each endpoint represented by and representing endpoint 6 is frozen. Hence, we move to execute
 1405 Step 2(b). Next, de-queuing queue $Q = \{5\}$ gives us endpoint 5 and therefore $Q = \{\}$. Endpoint 5
 1406 is represented by one endpoint, namely endpoint 0 in row 1. We hop to row 1. Both the endpoints
 1407 of row 1 are marked “visited” (and are frozen). Hence, we move to Step 3 as queue Q is empty.
 1408 This completes one iteration of the hopping phase, which we call as a duadic hop.

1409 Overall, the number of frozen endpoints is six. Hence, the corresponding vertex cover $S_4 =$
 1410 $\{0, 1, 3, 7, 8, 9\}$ is of size six and is not smaller than S , the original vertex cover given to us. Therefore,
 1411 we discard all changes made to the represents table and start with the represents table depicted
 1412 in [Table 7](#). Because the duadic hop we performed does not decrease the size of the vertex cover, it is
 1413 not a diminishing hop³³. Next, we repeat the entire exercise we discussed by removing the endpoint
 1414 9 of the duad in [Table 7](#). Finally, at the end of the execution, we get the corresponding vertex cover
 1415 S_9 of size six as well. Hence, as per Step 4, the vertex cover S is not diminishable. This is because
 1416 neither of the two duadic hops is a diminishing hop. More specifically, each duadic hop resulted in
 1417 a vertex cover of size that is same as the given vertex cover. Hence, the given vertex cover is indeed
 1418 the minimum vertex cover, which we formally prove in the succeeding discussion.

1419 We are now ready to define a diminishing hop and prove the corresponding theorem that relates
 1420 the above-discussed restricted diminishing hops to the minimum vertex cover.

1421 **Definition 19** (Dadic Hop). Given a represents table R where each endpoint u in table R is
 1422 either frozen or removed, a duadic hop H w.r.t. to the frozen endpoints in table R (equivalently
 1423 w.r.t. to a given vertex cover S) is a sequence of operations that (i) starts at a row where both
 1424 (duad of) the endpoints are frozen, (ii) removes one of the endpoints, (iii) freezes each endpoint
 1425 that represents or is represented by the removed endpoint, and (iv) repeats each operation until no
 1426 “unvisited” endpoint remains or “unvisited” but removed endpoints remain or an already frozen
 1427 endpoint is encountered.

³³This 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) |

1428 A duadic hop is analogous to an alternating path ([Definition 16](#)), but unlike an alternating
1429 path, where the path alternates between edges in matching and not in matching, a duadic hop is
1430 a sequence of operations that alternately freezes and removes endpoints in the table R . Hence,
1431 while an alternating path exists in a graph, a duadic hop does not exist in a table R but is created.
1432 The nomenclature of a duadic hop is based on the fact that the hop starts with a duad (or pair)
1433 of frozen endpoints, removes one of them, and hops to another row, (possibly) creating a duad of
1434 frozen endpoints at the row it hopped to³⁴. We now define a diminishing hop:

1435 **Definition 20** (Diminishing Hop). *Given a represents table R where each endpoint u in table R is
1436 either frozen or removed, a diminishing hop H w.r.t. to the frozen endpoints in table R (equivalently
1437 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
1438 one more endpoint than it freezes while satisfying all the properties of the table R .*

1439 A diminishing hop is analogous to an augmenting path ([Definition 17](#)). The former diminishes
1440 the size of a given vertex cover, and the latter augments the size of a given matching. Let us
1441 elaborate on the similarity between the two concepts. To find a maximum matching, Berge [[Ber57](#),
1442] suggested searching for augmenting paths. Specifically, he suggested starting from an
1443 exposed vertex (i.e., a vertex which is connected to edges not in a matching). Then, walk along
1444 an alternating path by iteratively finding the path until it exists. Consequently, if this alternating
1445 path stops at an exposed vertex, then it is an augmenting path, and the size of the matching can
1446 be increased by one. On the other hand, if the path is not augmenting, then backtrack (a little),
1447 choose another edge, and continue to form an alternating path until no augmenting path exists.

1448 Similarly, to find a minimum vertex cover, we propose searching for diminishing hops. Specif-
1449 cally, we are given a vertex cover S . The vertices in the vertex cover correspond to the frozen

³⁴The 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.

1450 endpoints in the represents table (and vertices in $V \setminus S$ correspond to removed endpoints). If
 1451 exactly one endpoint of each row is frozen, it is the minimum vertex cover, and we need not do
 1452 anything. However, if we have two frozen endpoints in a row, then we start from a row where
 1453 both of its endpoints are frozen. Then, move along a duadic hop by iteratively removing and
 1454 freezing endpoints until possible. Consequently, when duadic hops stop, if the number of endpoints
 1455 removed is greater than the number of endpoints frozen, then the duadic hop is a diminishing hop,
 1456 and the size of the vertex cover is decreased (by one). On the other hand, if the duadic hop is not
 1457 diminishing, then backtrack (a little), choose another endpoint to remove (or row with two frozen
 1458 endpoints), and continue duadic hops until no diminishing hop exists. At this point, we stress that
 1459 we do not discuss time complexity and focus on formalizing the relation between diminishing hops
 1460 and minimum vertex cover.

1461 Note that the definitions of a duadic hop and a diminishing hop are general in that they hold for
 1462 diminishing hops for restricted and for the general case³⁵. We now prove a theorem that associates
 1463 diminishing hop and minimum vertex cover for the restricted case of represents table R where
 1464 exactly one row contains two frozen endpoints. Subsequently, we prove a theorem that associates
 1465 diminishing hop and minimum vertex cover for the general case of represents table R where at least
 1466 one row contains two frozen endpoints. Recall that when one endpoint of each row is frozen, the
 1467 frozen endpoints form a minimum vertex cover, and hence, we do not need to prove anything.

1468 **Theorem 6** (Restricted Diminishing Hop and Vertex Cover). *Given a cubic bridgeless graph G ,*
 1469 *a corresponding represents table R and a vertex cover S of size $\frac{|V|}{2} + 1$, S is the minimum-size*
 1470 *vertex cover derivable from the represents table R if and only if there is no S -diminishing hop in*
 1471 *the represents table R .*

1472 **Proof Outline:** In this restricted case, the size of the minimum vertex cover can either be $\frac{|V|}{2} + 1$
 1473 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.
 1474 the given vertex cover does not exist, then it implies that the given vertex cover is the minimum
 1475 vertex cover. Additionally, if there is a diminishing hop, then the resultant vertex cover is of size
 1476 $\frac{|V|}{2}$, which is the minimum-size vertex cover possible in a graph having a perfect matching. On the
 1477 other hand, if the given vertex cover is not a minimum vertex cover, then there exists a diminishing
 1478 hop that leads to a minimum vertex cover that consists of any one of the two frozen endpoints in
 1479 it.

1480 Finally, note that the proof is designed to hold independent of the execution of the first two
 1481 phases of the algorithm. More specifically, the proof holds for *any* given represents table that
 1482 satisfies all its properties. Moreover, Petersen's Theorem guarantees that a locally optimal vertex
 1483 cover w.r.t. the represents table is also globally optimal ([Lemma 4](#)).

1484 *Proof.* We prove the contrapositive of the theorem. We start with the reverse direction, which is
 1485 relatively simpler than the forward direction.

1486 (\Leftarrow) If there exists an S -diminishing hop, then S is not a minimum vertex cover.

1487 Let S be a vertex cover of size $\frac{|V|}{2} + 1$. Then the corresponding represents table R consists
 1488 of $\frac{|V|}{2} + 1$ frozen endpoints (and $\frac{|V|}{2} - 1$ removed endpoints). Given that we use cubic bridgeless
 1489 graphs, there is always a perfect matching, which means that there are $\frac{|V|}{2}$ rows in the represents
 1490 table. This, by pigeonhole principle and by design requiring each row to have at least one frozen

³⁵Given 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.

1491 endpoint, implies that exactly one row in the represents table consists of a duad (i.e., a row with
 1492 both its endpoints as frozen). Let u and v be the endpoints that form a duad.

1493 Given that there exists an S -diminishing hop, we know that, by definition, the hop begins at the
 1494 row with the duad u and v . In turn, the diminishing hop decreases the number of frozen endpoints
 1495 in the table by one. The resultant frozen endpoints, one in each row, correspond to a vertex cover
 1496 S' . Hence, we know that either one of u or v will be in the vertex cover S' . Formally, given that
 1497 $S \cap \{u\} \neq \emptyset$ and $S \cap \{v\} \neq \emptyset$, it means that either $S' \cap \{u\} = \emptyset$ or $S' \cap \{v\} = \emptyset$. For the remaining
 1498 $\frac{|V|}{2} - 1$ rows where one of its two endpoints (y and z) is frozen, we know that either $S \cap \{y\} = \emptyset$
 1499 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
 1500 analogously, if $z \in S$, then it does not imply $z \in S'$). We only know for a fact that either one
 1501 of the two endpoints in a row will be in the vertex covers S, S' . We do not need to show which
 1502 specific endpoint will be in the vertex covers. Overall, we are sure that $|S'| = |S| - 1$. Therefore,
 1503 because we were able to find a vertex cover S' of a size smaller than the size of the vertex cover
 1504 S ($|S'| < |S|$; here, specifically $|S'| = |S| - 1$), S cannot be a minimum vertex cover. In fact, for
 1505 this restricted setting, we provide a stronger argument: vertex cover S' of size $\frac{|V|}{2}$ is indeed the
 1506 minimum vertex cover (Lemma 1). Hence, S cannot be a minimum vertex cover. We now prove
 1507 the other direction.

1508 (\Rightarrow) If S is not a minimum vertex cover, then there exists an S -diminishing hop.

1509 Let S be a vertex cover that is not a minimum vertex cover. Since S is not a minimum vertex
 1510 cover, there must exist at least one vertex cover S' that is smaller than S , i.e., $|S'| < |S|$. In this
 1511 restricted case, we know that the vertex cover S is of size $\frac{|V|}{2} + 1$, and the vertex cover S' is of size
 1512 $\frac{|V|}{2}$. We now find the association between S and S' .

1513 **Take Symmetric Difference of vertices in S and S' :** Let G' be a subgraph of G such that
 1514 the vertices in G' is a symmetric difference between the vertex covers S and S' . Formally,

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

1515 This means that the vertices in the subgraph G' are the vertices that are in the vertex cover S but
 1516 not in the vertex cover S' , or in S' but not in S . Vertices that are in both S and S' or that are
 1517 neither in S nor in S' are omitted. The edges in subgraph G' correspond to the edges in the graph
 1518 G that connect the vertices in G that correspond to the vertices in G' .

1519 **Properties of the subgraph G' :**

- 1520 1. The edges not in G' but in G are covered by the vertices that are in both the vertex covers S
 1521 and S' . Hence, the edges in G' are the edges that remain uncovered when the vertices only
 1522 in S' or only in S are removed.
- 1523 2. Each edge in the subgraph G' is covered by vertices that correspond to the vertices only in
 1524 the vertex cover S . Simultaneously, each edge in the subgraph G' is covered by vertices that
 1525 correspond to the vertices only in the vertex cover S' .
- 1526 3. The number of vertices in the subgraph G' is odd. Specifically, suppose there are m' vertices
 1527 in the subgraph G' that correspond to the vertices in the vertex cover S . In that case, there
 1528 are $m' - 1$ vertices in the subgraph G' that correspond to the vertices in the vertex cover S' .
- 1529 4. The subgraph G' can be a single vertex (singleton), a linear chain, a cycle, or a tree (or
 1530 multiple connected components of one or more of the four).

1531 **Relating S and S' to subgraph G' :** The vertices in G' alternate between a vertex in S and
 1532 a vertex in S' . This is because for every edge $e = (u, v)$ in subgraph G' , if the edge is covered by
 1533 vertex u in S , then it is covered by vertex v in S' , or vice versa. Vertices u and v cannot be in the
 1534 same vertex cover together. Otherwise, both the vertices would not be in the symmetric difference
 1535 of S and S' , and consequently, $e = (u, v)$ would not be an edge in the subgraph G' . Additionally, it
 1536 is not possible that a vertex cover S (or S') does not contain either of the vertices u and v because
 1537 if that is the case then S (or S') will not be a vertex cover as edge $e = (u, v)$ will not be covered.

1538 **Identifying a Diminishing Hop from the subgraph G' :** This is a critical part. Until now,
 1539 the proof in the forward direction has only discussed graphs and not mentioned the represents
 1540 table, which is needed for diminishing hops. Hence, we first associate the vertex covers S and S' for
 1541 the given graph G with the represents tables R and R' , respectively, that correspond to the given
 1542 graph G . More specifically, given a graph G and a vertex cover S , there is a represents table R
 1543 such that endpoints that correspond to vertices in the vertex cover S are frozen and the remainder
 1544 are removed. Similarly, given a graph G and a vertex cover S' , there is a represents table R' such
 1545 that endpoints that correspond to vertices in the vertex cover S' are frozen and the remainder are
 1546 removed. Next, in this restricted case, we know that one row of the represents table R consists
 1547 of a duad of frozen endpoints. Hence, remove the endpoint from the row with a duad such that
 1548 the removed endpoint is in the subgraph G' . The other endpoint in the duad will be frozen and
 1549 must be in both the vertex covers S and S' , and hence, not in the subgraph G' . Then, follow the
 1550 sequence of remove and freeze operations, i.e., the duadic hop w.r.t. S . Consequently, the table
 1551 R will become the same as table R' such that the endpoints removed from R during the duadic
 1552 hop correspond to the vertices in G' that are from the vertex cover S , and the endpoints frozen
 1553 in R during the duadic hop correspond to the vertices in G' that are from the vertex cover S' .
 1554 Hence, after the duadic hop w.r.t. S , table R becomes equivalent to table R' . Finally, given that
 1555 duadic hop w.r.t. S in the represents table R leads to a smaller number of frozen endpoints, and in
 1556 turn, smaller vertex cover, it implies that the S -duadic hop is, by definition, an S -diminishing hop.
 1557 Overall, because S is not a minimum vertex cover, the given table R has an S -diminishing hop.

1558 In summary, the alternating vertices in the subgraph G' (please refer to “Relating S and S'
 1559 to subgraph G' ”) from the vertex cover S and S' correspond to the removed and frozen endpoints
 1560 in the table R , respectively. Hence, the vertices in subgraph G' correspond to the sequence of
 1561 endpoints that are removed and frozen, i.e., an S -duadic hop. The duadic hop is an S -diminishing
 1562 hop. Therefore, if S is not a minimum vertex cover, then there exists an S -diminishing hop.

1563 This completes the other direction of the proof of correctness. In turn, it completes the over-
 1564 all contrapositive proof that shows that S is the minimum-size vertex cover derivable from the
 1565 represents table R if and only if there is no S -diminishing hop in the represents table R . \square

1566 We established the relation between a vertex cover S and an S -diminishing hop in the represents
 1567 table R for the restricted case where the represents table R has *exactly* one duad, i.e., *exactly* one
 1568 row with two frozen endpoints. We now generalize this result to the case where the represents table
 1569 R has *at least* one duad, i.e., *at least* one row with two frozen endpoints. As for the implementation,
 1570 the steps we mentioned (starting at the end of page 39) remain the same. However, when more
 1571 than one row has a duad, Step 1 is repeated for each of the existing duads unless a duadic hop
 1572 beginning at each duad is guaranteed to not be a diminishing hop. The formalization on how
 1573 to ensure that no diminishing hop remains and its corresponding time complexity is discussed in
 1574 Section 6 (Algorithm) and Section 8 (Time Complexity), respectively. Here, we continue to focus
 1575 on establishing the relation between a minimum vertex cover S and S -diminishing hop.

1576 **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 .*

1579 *Proof.* Before we begin the discussion of the proof, we share two observations:

1580 **Observation 1.** *The size of the minimum vertex cover S will be between $\frac{|V|}{2}$ and $|V| - 1$, i.e.,*
 1581 $|S| \in [\frac{|V|}{2}, |V| - 1]$ ³⁶³⁷.

1582 More specifically, when $|S| = \frac{|V|}{2}$, there are two facts: (i) S is the minimum vertex cover because
 1583 the size of perfect matching is $\frac{|V|}{2}$ (Lemma 1), and (ii) there is no duad, and in turn, there will
 1584 be no S -diminishing hop. Conversely, when there is no duad in a represents table R , and in turn,
 1585 there is no S -diminishing hop, it means that exactly $\frac{|V|}{2}$ endpoints are frozen (one endpoint frozen
 1586 for each row), which implies that S is the minimum vertex cover.

1587 **Observation 2.** *When the given graph is a cubic bridgeless graph G , the minimum-size (smallest)
 1588 vertex cover derivable from the represents table R implies a minimum vertex cover.*

1589 We now discuss the main proof. We again prove the contrapositive of the theorem. This proof
 1590 has subtle variations from the proof for Theorem 6, which necessitates a detailed discussion. We
 1591 start with the reverse direction, which is relatively simpler than the forward direction. Also, recall
 1592 that m denotes the number of vertices in the given graph ($m = |V|$).

1593 (\Leftarrow) If there exists an S -diminishing hop, then S is not a minimum vertex cover.

1594 Let S be a vertex cover of size $\frac{m}{2} + 1 \leq |S| \leq m - 1$. Then the corresponding represents table
 1595 R consists of x frozen endpoints (and remainder $m - x$ removed endpoints) where x is an integer
 1596 such that $x \in [\frac{m}{2} + 1, m - 1]$. Given that we use cubic bridgeless graphs, there is always a perfect
 1597 matching, which means that there are $\frac{m}{2}$ rows in the represents table. This, by pigeonhole principle
 1598 and by design requiring each row to have at least one frozen endpoint, implies that at least one
 1599 row in the represents table R consists of a duad (i.e., a row with both its endpoints as frozen).

1600 Let u and v be the endpoints that form a duad. Given that there exists an S -diminishing hop,
 1601 we know that, by definition, the hop begins at the row with a duad, say, row i containing endpoints
 1602 u and v . In turn, the diminishing hop decreases the number of frozen endpoints in the table by at
 1603 least one because one of the endpoints from u or v will be removed. The resultant frozen endpoints
 1604 in table R' correspond to a vertex cover S' . Hence, we know that either one of u or v will be in the
 1605 vertex cover S' . Formally, given that $S \cap \{u\} \neq \emptyset$ and $S \cap \{v\} \neq \emptyset$, it means that either $S' \cap \{u\} = \emptyset$
 1606 or $S' \cap \{v\} = \emptyset$. For the remaining $\frac{m}{2} - 1$ rows, we do not need to show which specific endpoints will
 1607 be in each of the vertex covers S and S' . It suffices to show that, by definition, the total number of
 1608 frozen endpoints in the remaining $\frac{m}{2} - 1$ rows of table R will be at least equal to the total number
 1609 of frozen endpoints in the remaining $\frac{m}{2} - 1$ rows of table R' . Hence, $|S \setminus \{u, v\}| \geq |S' \setminus \{u\}|$ or
 1610 $|S \setminus \{u, v\}| \geq |S' \setminus \{v\}|$. Consequently, $|S| > |S'|$. Therefore, because we were able to find a vertex
 1611 cover S' of a size smaller than the size of the vertex cover S ($|S'| < |S|$), S cannot be a minimum
 1612 vertex cover. We now prove the other direction.

1613 (\Rightarrow) If S is not a minimum vertex cover, then there exists an S -diminishing hop.

³⁶The 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.

³⁷A 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 $a \leq x \leq b$.

1614 Let S be a vertex cover that is not a minimum vertex cover. Since S is not a minimum vertex
1615 cover, there must exist at least one vertex cover S' that is smaller than S , i.e., $|S'| < |S|$. We now
1616 find the association between S and S' .

1617 **Take Symmetric Difference of vertices in S and S' :** Let G' be a subgraph of G such that
1618 the vertices in G' is a symmetric difference between the vertex covers S and S' . Formally,

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

1619 The edges in subgraph G' correspond to the edges in the graph G that connect the vertices in G
1620 that correspond to the vertices in G' .

1621 **Properties of the subgraph G' :**

- 1622 1. The edges not in G' but in G are covered by the vertices that are in both the vertex covers S
1623 and S' . Hence, the edges in G' are the edges that remain uncovered when the vertices only
1624 in S' or only in S are removed.
- 1625 2. Each edge in the subgraph G' is covered by vertices that correspond to the vertices only in
1626 the vertex cover S . Simultaneously, each edge in the subgraph G' is covered by vertices that
1627 correspond to the vertices only in the vertex cover S' .
- 1628 3. The subgraph G' can be a single vertex (singleton), a linear chain, a cycle, or a tree (or
1629 multiple connected components of one or more of the four, or a bipartite graph).

1630 **Relating S and S' to G' :** If the graph G' is a singleton or consists of a singleton component,
1631 then that single vertex must come from the vertex cover S . For the remaining cases, the vertices
1632 in (each connected component of) G' alternate between a vertex in S and a vertex in S' . In other
1633 words, G' is a bipartite graph. This is because for every edge $e = (u, v)$ in subgraph G' , if the edge
1634 is covered by vertex u in S , then it is covered by vertex v in S' , or vice versa. Vertices u and v
1635 cannot be in the same vertex cover together. Additionally, a vertex cover S (or S') must contain
1636 either of the vertices u and v .

1637 **Identifying a Diminishing Hop from the subgraph G' :** Until now, this proof in the forward
1638 direction discussed graphs and did not mention the represents table, which is needed for diminishing
1639 hops. Hence, we first associate the vertex covers S and S' for the given graph G with the represents
1640 tables R and R' , respectively, that correspond to the given graph G . More specifically, given a graph
1641 G and a vertex cover S , there is a represents table R such that endpoints that correspond to vertices
1642 in the vertex cover S are frozen and the remainder are removed. Similarly, there is a represents
1643 table R' for a given graph G and a vertex cover S' . This mapping from a graph to a represents
1644 table is general because:

- 1645 • how a represents table is populated is not used in the proof. Hence, the logic holds for any
1646 represents table that can be derived in the preceding two phases. Hence, the one represents
1647 table that will be derived by the algorithm is inherently covered by our proof.
- 1648 • additionally, just like an adjacency list is a form of a direct and complete representation of the
1649 underlying graph's structure, a represents table is also a direct and complete representation
1650 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 here³⁸.

- 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)

³⁸Just 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.

1692 cannot be a vertex cover. This contradicts our assumption about the presence of an odd cycle
1693 in G' .

- 1694 • **Even Linear Chain:** A linear chain of even length cannot exist by itself without another
1695 component in G' for reasons that are the same as even cycles. Hence, we focus our discussion
1696 on linear chains of odd length.

- 1697 • **Odd Linear Chain:** When the graph G' is (has) a linear chain of odd length, the linear
1698 chain consists of at least one vertex more from the vertex cover S than from the vertex cover
1699 S' . Hence, an S -diminishing hop can begin at any row with a duad in the represents table R
1700 such that one of the endpoints in the duad corresponds to a vertex from S in the linear chain
1701 of G' . Subsequently, the table R will become the same as table R' such that the endpoints
1702 removed from R during the S -diminishing hop correspond to the vertices in G' that are from
1703 the vertex cover S , and the endpoints frozen in R correspond to the vertices from the vertex
1704 cover S' . Finally, the S -diminishing hop in the represents table R leads to a smaller number of
1705 frozen endpoints, and in turn, a smaller vertex cover. Therefore, because S is not a minimum
1706 vertex cover, the given table R has an S -diminishing hop.

1707 Here, we note that such a mapping between the graph structure (G') and the table operations
1708 on R (hopping) exist because, by design and due to table properties, each vertex removed
1709 from G' correspond to a removal of an endpoint in the table, which in turn implies that
1710 each endpoint that represents and is represented by the removed endpoint will be frozen. In
1711 turn, this implies that at least one of the vertices in G' will be frozen too that correspond
1712 to the vertex from vertex cover S' . Overall, the aim is not to find an odd linear chain in
1713 the represents table. Rather we simply needed to prove that when such an odd linear chain
1714 exists in graph G' , there exists at least one duadic hop in represents table R , independent of
1715 how the table is derived, such that it mimics the odd linear chain in graph G' through its
1716 sequence of remove and freeze operations, which correspond to removing the vertices from
1717 vertex cover S and freezing the vertices from vertex cover S' in graph G' , respectively. This
1718 discussion of mapping between the graph structure (G') and the table operations (hopping)
1719 is generalizable and holds for each graph structure as discussed.

- 1720 • **Even Tree:** When G' is (has) a tree, it must be a binary tree because it is derived from a
1721 cubic graph G . When the length of the longest path between each pair of leaf vertices is even,
1722 it results in the same case as an even linear chain, and there is no change in the number of
1723 vertices in S and S' .

- 1724 • **Odd Tree:** When there is an odd path, an S -diminishing hop begins at a row with a duad
1725 in the table R such that one endpoint in the duad corresponds to a vertex from S in G' .
1726 Subsequently, the table R becomes equivalent to R' such that the endpoints removed from R
1727 correspond to the vertices in G' from the vertex cover S , and the endpoints frozen correspond
1728 to the vertices in G' from the vertex cover S' . More generally, because G' *must* be bipartite,
1729 this discussion (regarding an S -diminishing hop beginning at a row with a duad in the R)
1730 holds for the general case when G' is a bipartite graph.

1731 In summary, for each graph type, the alternating vertices in the subgraph G' from the vertex
1732 cover S and S' correspond to the removed and frozen endpoints in the table R , respectively.
1733 Hence, the vertices in subgraph G' correspond to the sequence of endpoints that are removed and
1734 frozen, i.e., an S -duadic hop. The duadic hop is an S -diminishing hop because the number of
1735 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 . \square

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**. \square

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

³⁹Berge’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.

1774 to populate a novel data structure called the “represents table”. We also discussed operations
1775 and properties of the represents table.

1776 **III Diminishing Hops:** The input to the third phase is the data structure represents table.
1777 Foremost, assign a weighted number to each vertex (also known as an endpoint) in the table,
1778 called the representation score ζ , which captures how well an endpoint is represented in the
1779 table. Next, use the representation score ζ to freeze or remove each endpoint in the represents
1780 table. The frozen endpoints correspond to a vertex cover. Finally, motivated by the use of
1781 augmenting paths (a specific case of an alternating path) w.r.t. a given matching to find a
1782 maximum matching, the third phase introduces diminishing hops (a specific case of a duadic
1783 hop) w.r.t. a given vertex cover to find a minimum vertex cover. A diminishing hop differs from
1784 the Vertex Cover Reconfiguration problem because we do not (i) stipulate that each remove /
1785 freeze operation of a hop should result in a vertex cover or (ii) bound the number of operations
1786 needed to complete the hopping.

1787 Overall, the combination of all these phases implies that we get an unconditional deterministic
1788 polynomial-time algorithm for the vertex cover problem on cubic bridgeless graphs.

1789 6 Algorithm

1790 We now present the core contribution of this paper, an algorithm to solve the VC – CBG problem. In
1791 the algorithm, all ties are broken and all ordering (sorting) of vertices is done based on lexicographic
1792 ordering unless noted otherwise. The ordering does not impact the correctness but ensures that for
1793 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

```
1:  $V_s$  = lexicographically sorted set of vertices
2: // PHASE I
3:  $M$  = a set of edges in a perfect matching found using the Blossom Algorithm [Edm65]
4: if  $k < |M|$  then
5:   return No
6: end
7: // PHASE II
8:  $R$  = POPULATE_REPRESENTS_TABLE( $G, M, V_s$ ) // Algorithm 2
9: // PHASE III
10:  $S$  = DIMINISHING_HOP_PHASE( $R$ ) // Algorithm 3
11: if  $|S| \leq k$  then
12:   return Yes
13: end
14: 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

1: 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

2: 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

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

4: **for each** level in T **do**

5: **for each** unvisited vertex u in level **do**

6: **if** there exists an edge that connects vertex u with another vertex on the same level
and the edge is in M **then**

7: | select the edge

8: **else if** there exists an edge that connects vertex u with another vertex on the next
level and the edge is in M **then**

9: | select the edge

10: **end**

11: Mark the two endpoints of the selected edge as visited in T

12: 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

13: Remove from graph G the selected edge and all the edges that are connected to the
two endpoints

14: If any vertex becomes edgeless in G , mark the vertex as visited in T

15: **end**

16: **end**

17: **return** R

Algorithm 3: DIMINISHING_HOP_PHASE(R)

Data: Represents Table R

Result: returns a Minimum Vertex Cover S

1: $S = \emptyset$

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

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

4: $R = \text{COMPUTE_REPRESENTATION_SCORE}(R)$ // Algorithm 4

5: $R, S = \text{VERTEX_ELIMINATION}(R, S)$ // Algorithm 5

6: **for each** integer a in $[1, \frac{m}{2}]$ **do**

7: | $R, S = \text{DIMINISHING_HOPS}(R, S)$ // Algorithm 7

8: **end**

9: **return** S

Algorithm 4: COMPUTE REPRESENTATION SCORE(R)

Data: Represents Table R
Result: returns Represents Table R with updated representation scores ζ

```
1: // The following loop traverses through the table  $R$  top-down
2: for each  $row$  in  $R$  do
3:   // The following loop executes exactly twice
4:   for each endpoint  $u$  in  $row$  do
5:     if  $u$  is frozen then
6:        $\zeta_u = -1$ 
7:       continue
8:     else if  $u$  is removed then
9:        $\zeta_u = -1$ 
10:      continue
11:    else
12:       $\zeta_u = 0$ 
13:      for each  $row_j$  in  $R$  that is above  $row$  do
14:         $x, y$  = two endpoints in  $row_j$ 
15:        //  $L_x$  denotes the list of endpoints that the endpoint  $x$  in  $row_j$  represents
16:        if vertex  $u \in L_x$  then
17:           $\zeta_u = \zeta_u + \max(0, \zeta_y) + 1$ 
18:        else if vertex  $u \in L_y$  then
19:           $\zeta_u = \zeta_u + \max(0, \zeta_x) + 1$ 
20:        else
21:          do nothing
22:        end
23:      end
24:    end
25:  end
26: end
27: return  $R$ 
```

Algorithm 5: VERTEX_ELIMINATION(R, S)

Data: Represents Table R
Vertex Cover S

Result: returns updated Represents Table R , Vertex Cover S

```
1: // The following loop traverses through the table  $R$  bottom-up
2: for each row in  $R$  do
3:    $R = \text{COMPUTE\_REPRESENTATION\_SCORE}(R)$  // Algorithm 4
4:   if both endpoints in row are either frozen or removed then
5:     continue
6:   else if endpoint  $u$  in row remains and endpoint  $v$  in row is frozen then
7:      $R, S = \text{FREEZE\_AND\_REMOVE}(R, S, \emptyset, u)$  // Algorithm 6.  $\emptyset$  denotes a null value
8:   else
9:     // at this point, both endpoints  $u$  and  $v$  in row are neither frozen nor removed, and
       represent exactly one endpoint, namely each other
10:    if  $\zeta_u \geq \zeta_v$  then
11:       $R, S = \text{FREEZE\_AND\_REMOVE}(R, S, u, v)$  // Algorithm 6
12:    else
13:       $R, S = \text{FREEZE\_AND\_REMOVE}(R, S, v, u)$  // Algorithm 6
14:    end
15:  end
16: end
17: 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

```
1: // Freeze Operation of Represents Table
2: Freeze endpoint  $\psi$  in  $R$ 
3: Append endpoint  $\psi$  to  $S$ 
4: Set the represents list  $L_\psi$  of endpoint  $\psi$  in  $R$  to null
5: Delist endpoint  $\psi$  from every represents list in  $R$ 
6: // Remove Operation of Represents Table
7: Remove endpoint  $\omega$  from  $R$ 
8: Remove endpoint  $\omega$  from  $S$  (if present)
9: for each non-frozen and unremoved endpoint  $u$  in  $R$  such that  $\omega \in L_u$  do
10:   |  $R, S = \text{FREEZE\_AND\_REMOVE}(R, S, u, \emptyset)$ 
11: end
12: for each non-frozen and unremoved endpoint  $u$  in  $L_\omega$  do
13:   |  $R, S = \text{FREEZE\_AND\_REMOVE}(R, S, u, \emptyset)$ 
14: end
15: Set the represents list  $L_\omega$  of endpoint  $\omega$  in  $R$  to null
16: 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

```
1:  $\lambda = \emptyset$  // an array of endpoints (vertices) visited during diminishing hops
2:  $R_{diminished} = R$ 
3:  $S_{diminished} = S$ 
4: // The loop traverses through the table  $R$  top-down
5: for each row in  $R$  do
6:    $R_{original} = R$ 
7:    $S_{original} = S$ 
8:    $\lambda_{original} = \lambda$ 
9:   // a duadic hop exists only if both endpoints  $u$  and  $v$  in row are frozen, and form a
     duad
10:  if both endpoints in row are frozen then
11:    for each endpoint  $u$  in row do
12:       $R, S, \lambda = \text{DUADIC\_HOP}(R, S, \emptyset, u, \lambda)$  // Algorithm 8
        // this holds if the number of vertices removed > number of vertices frozen
13:      if  $|S| < |S_{diminished}|$  then
14:         $R_{diminished} = R$ 
15:         $S_{diminished} = S$ 
16:         $\lambda_{diminished} = \lambda$ 
17:      end
18:    end
19:     $R = R_{original}$ 
20:     $S = S_{original}$ 
21:     $\lambda = \lambda_{original}$ 
22:  end
23:   $R = R_{diminished}$ 
24:   $S = S_{diminished}$ 
25:   $\lambda = \lambda_{diminished}$ 
26: end
27: end
28: return  $R, S$ 
```

Algorithm 8: DUADIC_HOP($R, S, \psi, \omega, \lambda$)

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

```
1: // During each execution of this algorithm, either  $\psi = \emptyset$  or  $\omega = \emptyset$ 
2: if  $\omega \neq \emptyset$  then
3:   if  $\omega \in \lambda$  then
4:     | return  $R, S, \lambda$ 
5:   end
6:   Add  $\omega$  to  $\lambda$ 
7:   Remove endpoint  $\omega$  from  $R$ 
8:   Remove endpoint  $\omega$  from  $S$ 
9:    $Q = \emptyset$  // a queue storing the endpoints to be frozen
10:  for each endpoint  $u$  in  $L_\omega$  do
11:    if  $u \notin \lambda$  then
12:      Add  $u$  to  $\lambda$ 
13:      if  $u \notin S$  then
14:        |  $Q = Q \cup \{u\}$  // enqueue endpoint  $u$ 
15:      end
16:    end
17:  end
18:  for each endpoint  $u$  in  $R$  such that  $\omega \in L_u$  do
19:    if  $u \notin \lambda$  then
20:      Add  $u$  to  $\lambda$ 
21:      if  $u \notin S$  then
22:        |  $Q = Q \cup \{u\}$ 
23:      end
24:    end
25:  end
26:  for each endpoint  $u$  in  $Q$  do
27:     $Q = Q \setminus \{u\}$  // dequeue endpoint  $u$ 
28:     $R, S, \lambda = \text{DUADIC\_HOP}(R, S, u, \emptyset, \lambda)$ 
29:  end
30: end
31: // the following condition will be true only when an endpoint has been removed
32: if  $\psi \neq \emptyset$  then
33:   Freeze endpoint  $\psi$  in  $R$ 
34:   Append endpoint  $\psi$  to  $S$ 
35:   // the following condition is equivalent to checking for the existence of a duad
36:    $u =$  the other endpoint that is in the same row of  $R$  as  $\psi$ 
37:   if  $u \in S$  then
38:     |  $R, S, \lambda = \text{DUADIC\_HOP}(R, S, \emptyset, u, \lambda)$ 
39:   end
40: end
41: return  $R, S, \lambda$ 
```

1794 **7 Proof of Correctness**

1795 We proved the relation between diminishing hops and minimum vertex cover in [Section 5](#). Hence,
 1796 showing that the algorithm successfully searches for diminishing hops, which, combined with already
 1797 proven results, implies the correctness of the algorithm. Overall, we prove the following theorem:

1798 **Theorem 8.** *[Algorithm 1](#) returns **Yes** if and only if the given instance of $\text{VC} - \text{CBG}$ is a **Yes**
 1799 instance.*

1800 More specifically, we prove the theorem through a sequence of lemmas. Foremost, in the reverse
 1801 direction, we have the following lemma:

1802 **Lemma 5.** *If the given instance of $\text{VC} - \text{CBG}$ is a **Yes** instance, then the [Algorithm 1](#) returns **Yes**.*

1803 *Proof.* When the given instance of $\text{VC} - \text{CBG}$ is a **Yes** instance, it implies that there is a vertex
 1804 cover S of size at most k ($|S| \leq k$). Additionally, it implies that $k \geq \frac{m}{2}$. This is because a
 1805 cubic bridgeless graph always consists of a perfect matching ([Theorem 3](#)), which means the size
 1806 of a maximum matching M (equivalently, a perfect matching for our paper) is $\frac{m}{2}$. Therefore, by
 1807 [Lemma 1](#), we know that the size of a minimum vertex cover S' is $|S'| \geq |M|$, which means $|S'| \geq \frac{m}{2}$.
 1808 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,
 1809 because

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

1810 we know that

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

1811 Therefore, Line 5 of [Algorithm 1](#) cannot return **No**.

1812 Next, by design, we know that Line 5 of [Algorithm 3](#) consists of a vertex cover S . This is
 1813 because the operations of the represents table R are designed to ensure that the frozen endpoints
 1814 in table R correspond to a vertex cover ([Theorem 4](#)). Finally, at the end of the execution of the loop
 1815 in Line 6 of [Algorithm 3](#), the final vertex cover S in Line 7 of [Algorithm 3](#) consists of a minimum
 1816 vertex cover because there is no S -diminishing hop in the given represents table ([Theorem 7](#)). This
 1817 will be returned by Line 9 of [Algorithm 3](#). This implies that no vertex smaller than S can exist.
 1818 Therefore, if the given instance of $\text{VC} - \text{CBG}$ is a **Yes** instance, then each vertex cover $S' \in \mathcal{S}$ that
 1819 can be a **Yes** instance must be of size greater than or equal to the minimum vertex cover and
 1820 less than or equal to k . Formally, $|S| \leq |S'| \leq k$. Hence, the condition in Line 11 of [Algorithm 1](#)
 1821 must be true ($|S| \leq k$), which means Line 12 of [Algorithm 1](#) must return **Yes**. The execution of
 1822 [Algorithm 1](#) will never reach Line 14, and hence, it cannot return **No**. \square

1823 Next, in the forward direction, we have the following lemma:

1824 **Lemma 6.** *If the [Algorithm 1](#) returns **Yes**, then the given instance of $\text{VC} - \text{CBG}$ is a **Yes** instance.*

1825 *Proof.* If the [Algorithm 1](#) returns **Yes**, then it can do so only if Line 12 of [Algorithm 1](#) returns
 1826 **Yes**. This implies that Line 5 of [Algorithm 1](#) cannot return **No**. Hence, the value of non-negative
 1827 integer k must be greater than the size of the perfect matching M found in Line 3 of [Algorithm 1](#);
 1828 formally, $k \geq \frac{m}{2}$. This also implies that Lines 8 and 10 of [Algorithm 1](#) must be executed, which are
 1829 the two main phases of the algorithm. We first discuss the execution of Line 8.

1830 **Line 8 of Algorithm 1 (Populate Represents Table):** The Line 8 of [Algorithm 1](#) invokes
 1831 [Algorithm 2](#). The output of [Algorithm 2](#) is a data structure called the represents table R . Line 12 of
 1832 [Algorithm 2](#) inserts a new row to the table such that the endpoints of an edge selected in Lines 7 or
 1833 9 are listed. Because the algorithm only selects the edges that are in the perfect matching M (Lines
 1834 6 or 8), it implies that the algorithm enlists endpoints of edges in a matching, which means the
 1835 endpoints form a vertex cover ([Lemma 3](#)). More specifically, in our case, the matching is a perfect
 1836 matching, and each perfect matching is a maximum matching, which in turn is a maximal matching
 1837 ([Lemma 2](#)). Hence, each vertex of graph G is listed as an endpoint in table R , which trivially forms
 1838 a vertex cover ([Property 3](#)). The Lines 1, 13, and 14 of [Algorithm 2](#) are important for the next
 1839 phase as they ensure the table R has certain properties ([Property 1](#), [Property 2](#), [Property 4](#)). The
 1840 key output of [Algorithm 2](#) is that the endpoints of the resultant represents table R form a vertex
 1841 cover.
 1842

1842 **Line 10 of Algorithm 1 (Diminishing Hop Phase):** The Line 10 of [Algorithm 1](#) invokes
 1843 [Algorithm 3](#) with the represents table R as its input. The first three lines of [Algorithm 3](#) are
 1844 initialization steps that are consequential for the next lines. Hence, we do not discuss them. Line 4
 1845 of [Algorithm 3](#) invokes [Algorithm 4](#) (Compute Representation Score). [Algorithm 4](#) assigns a score
 1846 to each endpoint. It does not alter the structure of the table or its endpoints and hence, it does
 1847 not need further discussion. Next, Line 5 of [Algorithm 3](#) invokes [Algorithm 5](#).

- 1848 • **Algorithm 5 (Vertex Elimination):** The overarching goal of this algorithm is to freeze or
 1849 remove each endpoint in the represents table R using the representation score such that the
 1850 frozen endpoints correspond to a vertex cover S such that $S \subset V$ ([Property 4](#), [Theorem 4](#))⁴⁰.

1851 More specifically, [Algorithm 5](#) traverses through the represents table R bottom-up. For each
 1852 row, it recomputes the representation score by invoking [Algorithm 4](#). Then, it carries out a
 1853 sequence of freeze and remove operations while adhering to the properties of the represents
 1854 table R . If both endpoints in a row of the table are frozen, then the algorithm does nothing.
 1855 By design, both the endpoints in a row cannot be removed. If one of the endpoints in a row
 1856 is frozen (and the other is neither frozen nor removed), then the other is removed (Line 7 of
 1857 [Algorithm 5](#) invokes [Algorithm 6](#) (Freeze and Remove)). Next, if neither of the endpoints in
 1858 a row is frozen or removed, one endpoint is frozen and the other one is removed based on
 1859 the representation score (Line 11 or 13 of [Algorithm 5](#) invokes [Algorithm 6](#) as appropriate).
 1860 Finally, the case when one endpoint is removed and the other is neither frozen nor removed
 1861 cannot happen (by design). Hence, [Algorithm 5](#) does not need to cover that case.

1862 [Algorithm 6](#) is specifically designed to freeze or remove an endpoint. When an endpoint ψ
 1863 needs to be frozen, [Algorithm 6](#) freezes the endpoint in table R (Line 2), adds it to the vertex
 1864 cover S (Line 3), and updates the corresponding represents lists (Lines 4 and 5). Lines 4 and
 1865 5 set the represents list of endpoint ψ to null and remove the endpoint ψ from the represents
 1866 lists of other endpoints, respectively. This operation denotes that the vertex ψ in the vertex
 1867 cover S covers each edge that connects ψ to its neighbors. Next, when an endpoint ω needs
 1868 to be removed, [Algorithm 6](#) removes the endpoint from table R (Line 7), removes it from the
 1869 vertex cover S (if present; Line 8), and freezes each endpoint it represents or it is represented
 1870 by (Lines 9 to 14). The last set of operations denotes that the vertex ω is not in the vertex
 1871 cover S , and hence, each of its neighbors must be in S to cover each edge connected to ω .

⁴⁰We 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.

1872 In summary, [Algorithm 5](#), through [Algorithm 6](#), carries out a deterministic sequence of freeze
 1873 and remove operations such that each endpoint in the represents table R is either frozen or
 1874 removed. Consequently, the frozen endpoints correspond to the vertices in a vertex cover S .

1875 Finally, Line 7 of [Algorithm 3](#) invokes [Algorithm 7](#) for a total of $\frac{m}{2}$ times. Each iteration of an
 1876 S -diminishing hop corresponds to one row of the represents table R .

- 1877 • **Algorithm 7 (Diminishing Hops):** There are three aspects to be proven for [Algorithm 7](#):
 1878 (i) establish the relation between a vertex cover S and an S -diminishing hop, (ii) prove
 1879 that the algorithm performs an S -diminishing hop when one exists, and (iii) $\frac{m}{2}$ calls to the
 1880 algorithm ensures that there exists no S -diminishing hop when it terminates. [Theorem 7](#)
 1881 already established that a vertex cover S is minimum if and only if there is no S -diminishing
 1882 hop. Hence, it remains to be discussed that [Algorithm 7](#) is an algorithm that uses S -duadic
 1883 hops to search and perform an S -diminishing hop in the represents table R . Hence, when an
 1884 S -diminishing hop does not exist, S is a minimum vertex cover.

1885 **Lemma 7.** *Given a represents table R where each endpoint is either frozen or removed and*
 1886 *a vertex cover S that corresponds to the frozen endpoints in the table R , [Algorithm 7](#) is an*
 1887 *algorithm to perform an S -diminishing hop if it exists.*

1888 *Proof.* When Line 7 of [Algorithm 3](#) invokes [Algorithm 7](#), input to [Algorithm 7](#) is a represents
 1889 table R ⁴¹ where each endpoint is either frozen or removed and a vertex cover S ⁴². Note that
 1890 during each iteration, the updated values of represents table R and the vertex cover S are
 1891 passed. Line 1 of [Algorithm 7](#) initializes an empty list that will store each endpoint that is
 1892 visited during a hop. Lines 2 and 3 keep a record of the represents table R with the smallest
 1893 number of frozen endpoints and of the smallest vertex cover S , respectively, during a given
 1894 iteration. Line 5 ensures a top-down row-wise traversal of the represents table R . Lines 6
 1895 to 8 store the concerned data as it was when an iteration of the loop begins. This is needed
 1896 because an S -duadic hop removes each of the two endpoints turn-wise to assess which one
 1897 leads to an S -diminishing hop. Line 10 ensures the presence of a duad without which an
 1898 S -duadic hop is not needed. Consequently, Line 11 does an S -duadic hop for each of the
 1899 endpoints in a duad. Line 12 invokes [Algorithm 8](#) (Duadic Hop). Line 14 assesses whether
 1900 the vertex cover returned by [Algorithm 8](#) (Duadic Hop) is smaller than the smallest one, and
 1901 if so, updates the relevant variables (Lines 15-17). Lines 19 to 21 restore the relevant variables
 1902 for the other endpoint of the duad to undergo a duadic hop. Lines 23 to 25 ensure that after
 1903 each endpoint of the duad is traversed, the smallest vertex cover is used as input for the next
 1904 row.

1905 Next, we mentioned that Line 12 of [Algorithm 7](#) invokes [Algorithm 8](#) (Duadic Hop). Its
 1906 inputs are the represents table R , a vertex cover S , an endpoint to be frozen ψ , an endpoint
 1907 to be removed ω , and a list of visited endpoints. By design, when [Algorithm 8](#) is invoked,
 1908 either ψ or ω will be null. An S -duadic hop always begins with the removal of an endpoint,
 1909 as evident from Line 12 of [Algorithm 7](#). Foremost, an endpoint ω can be removed only if it
 1910 is not visited (Line 3). If the endpoint ω is marked as visited, it implies it was frozen during
 1911 the removal of another endpoint in another row. If not visited, ω is now marked as visited

⁴¹The 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.

⁴²During 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 hops⁴³. 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

⁴³The 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.

⁴⁴Alternative 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 .

1949 case, after at most $\frac{m}{2}$ calls to [Algorithm 7](#) and consequently, at most m^2 S -duadic hops, there
1950 cannot be an S -diminishing hop.

1951 Overall, we showed that after $\frac{m}{2}$ iterations of Lines 6 to 8 in [Algorithm 3](#), there cannot exist
1952 an S -diminishing hop in the represents table R . \square

1953 In summary, we proved that [Algorithm 7](#), along with [Algorithm 8](#), is an algorithm (i) to
1954 perform an S -diminishing hop ([Lemma 7](#)) when one exists, (ii) that guarantees that no S -
1955 diminishing hop exists when it terminates after performing at most m^2 S -duadic hops across
1956 $\frac{m}{2}$ calls to [Algorithm 7](#) by [Algorithm 3](#) ([Lemma 8](#)), and (iii) that is based on the fact that
1957 the non-existence of an S -diminishing hop implies S is a minimum vertex cover ([Theorem 7](#)).

1958 After the termination of the loop in Line 8 of [Algorithm 3](#), Line 9 of the algorithm returns a
1959 minimum vertex cover S to Line 10 of [Algorithm 1](#) because there will be no S -diminishing hop
1960 ([Lemma 7](#), [Lemma 8](#), [Theorem 7](#)). This discussion effectively completes the proof of this lemma.

1961 In summary, recall that we posited that for the [Algorithm 1](#) to return **Yes**, Lines 8 and 10 of
1962 [Algorithm 1](#) must be executed. We subsequently discussed the execution of these lines. We proved
1963 that for frozen endpoints in the resultant represents table R and the corresponding vertex cover S ,
1964 there exists no S -diminishing hop after the execution of Line 10 of [Algorithm 1](#). This implies that
1965 S is a minimum vertex cover ([Theorem 7](#)). Subsequently, when Line 12 of [Algorithm 1](#) returns
1966 **Yes**, the given instance of VC – CBG must be a **Yes** instance. Hence, if [Algorithm 1](#) returns **Yes**,
1967 then the given instance of VC – CBG is a **Yes** instance. This completes the proof in the forward
1968 direction. \square

1969 The [Lemma 5](#) and [Lemma 6](#) complete both directions of the proof of correctness, which com-
1970 pletes the proof of [Theorem 8](#).

1971 8 Time Complexity Analysis

1972 In this section, we discuss the time complexity of the algorithm ([Table 16](#), [Table 17](#), [Table 18](#),
1973 [Table 19](#), [Table 20](#), [Table 21](#), [Table 22](#), [Table 23](#)). Each table corresponds to each algorithm
1974 (ranging from [Algorithm 1](#) to [Algorithm 8](#)). m denotes the number of vertices V and n denotes
1975 the number of edges E . However, for cubic graphs, we know that $n = \frac{3m}{2} = \mathcal{O}(m)$. Hence, for
1976 simplicity and in line with the literature, we compute time complexity with respect to m .

1977 In each table, we give the complexity of each line (each operation), the complexity of the loop
1978 (complexity of line multiplied by the number of loop iterations) and the dominant complexity. For
1979 convenience, the beginning of a loop, specifically the number of loop iterations, is highlighted (e.g.,
1980 [Line 4](#) in [Table 17](#)). Each statement within the loop is prefixed with a pointer (\blacktriangleright). In the case of
1981 nested loops, an additional pointer ($\triangleright, >$) is used. Whenever an algorithm calls another algorithm,
1982 the latter's worst-case time complexity becomes the former's line complexity, which is denoted by
1983 square brackets ([Table x]; e.g., Line 8 in [Table 16](#)).

1984 **Theorem 9.** *The asymptotic running time of [Algorithm 1](#) is $\mathcal{O}(m^5)$.*

1985 *Proof.* Line 10 in [Algorithm 1](#) dominates the complexity of all other lines as shown in [Table 16](#). This
1986 dominant complexity is $\mathcal{O}(m^5)$. Hence, the time complexity of the entire algorithm is $\mathcal{O}(m^5)$. \square

1987 **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.

- 1991 • [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 $\mathcal{O}(m^2)$.
- 1996 • [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 (\blacktriangleright) denotes that a line is within a loop. An additional pointer (\triangleright) 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 | $\mathcal{O}(1)$ | $\blacktriangleright \mathcal{O}(m^2)$ | $\mathcal{O}(m^2)$ |
| 6 | $\mathcal{O}(m)$ | $\blacktriangleright \triangleright \mathcal{O}(m^3)$ | $\mathcal{O}(m^3)$ |
| 7 | $\mathcal{O}(1)$ | $\blacktriangleright \triangleright \mathcal{O}(m^2)$ | $\mathcal{O}(m^3)$ |
| 8 | $\mathcal{O}(m)$ | $\blacktriangleright \triangleright \mathcal{O}(m^3)$ | $\mathcal{O}(m^3)$ |
| 9 | $\mathcal{O}(1)$ | $\blacktriangleright \triangleright \mathcal{O}(m^2)$ | $\mathcal{O}(m^3)$ |
| 10 | - | - | $\mathcal{O}(m^3)$ |
| 11 | $\mathcal{O}(m)$ | $\blacktriangleright \triangleright \mathcal{O}(m^3)$ | $\mathcal{O}(m^3)$ |
| 12 | $\mathcal{O}(1)$ | $\blacktriangleright \triangleright \mathcal{O}(m^3)$ | $\mathcal{O}(m^3)$ |
| 13 | $\mathcal{O}(m)$ | $\blacktriangleright \triangleright \mathcal{O}(m^3)$ | $\mathcal{O}(m^3)$ |
| 14 | $\mathcal{O}(m)$ | $\blacktriangleright \triangleright \mathcal{O}(m^3)$ | $\mathcal{O}(m^3)$ |
| 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 (\blacktriangleright) 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] | $\blacktriangleright \mathcal{O}(m^5)$ | $\mathcal{O}(m^5)$ |
| 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 (\blacktriangleright) denotes that a line is within a loop. Each additional pointer (\triangleright , $>$) 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 | $\mathcal{O}(1)$ | $\blacktriangleright \mathcal{O}(m)$ | $\mathcal{O}(m)$ |
| 5 | $\mathcal{O}(1)$ | $\blacktriangleright \triangleright \mathcal{O}(m)$ | $\mathcal{O}(m)$ |
| 6 | $\mathcal{O}(1)$ | $\blacktriangleright \triangleright \mathcal{O}(m)$ | $\mathcal{O}(m)$ |
| 7 | - | - | $\mathcal{O}(m)$ |
| 8 | $\mathcal{O}(1)$ | $\blacktriangleright \triangleright \mathcal{O}(m)$ | $\mathcal{O}(m)$ |
| 9 | $\mathcal{O}(1)$ | $\blacktriangleright \triangleright \mathcal{O}(m)$ | $\mathcal{O}(m)$ |
| 10 | - | - | $\mathcal{O}(m)$ |
| 11 | - | - | $\mathcal{O}(m)$ |
| 12 | $\mathcal{O}(1)$ | $\blacktriangleright \triangleright \mathcal{O}(m)$ | $\mathcal{O}(m)$ |
| 13 | $\mathcal{O}(1)$ | $\blacktriangleright \triangleright \mathcal{O}(m^2)$ | $\mathcal{O}(m^2)$ |
| 14 | $\mathcal{O}(1)$ | $\blacktriangleright \triangleright > \mathcal{O}(m^2)$ | $\mathcal{O}(m^2)$ |
| 15 | - | - | $\mathcal{O}(m^2)$ |
| 16 | $\mathcal{O}(1)$ | $\blacktriangleright \triangleright > \mathcal{O}(m^2)$ | $\mathcal{O}(m^2)$ |
| 17 | $\mathcal{O}(1)$ | $\blacktriangleright \triangleright > \mathcal{O}(m^2)$ | $\mathcal{O}(m^2)$ |
| 18 | $\mathcal{O}(1)$ | $\blacktriangleright \triangleright > \mathcal{O}(m^2)$ | $\mathcal{O}(m^2)$ |
| 19 | $\mathcal{O}(1)$ | $\blacktriangleright \triangleright > \mathcal{O}(m^2)$ | $\mathcal{O}(m^2)$ |
| 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 (\blacktriangleright) 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] | $\blacktriangleright \mathcal{O}(m^3)$ | $\mathcal{O}(m^3)$ |
| 4 | $\mathcal{O}(1)$ | $\blacktriangleright \mathcal{O}(m)$ | $\mathcal{O}(m^3)$ |
| 5 | - | - | $\mathcal{O}(m^3)$ |
| 6 | $\mathcal{O}(1)$ | $\blacktriangleright \mathcal{O}(m)$ | $\mathcal{O}(m^3)$ |
| 7 | $\mathcal{O}(m^2)$ [Table 21] | $\blacktriangleright \mathcal{O}(m^3)$ | $\mathcal{O}(m^3)$ |
| 8 | - | - | $\mathcal{O}(m^3)$ |
| 9 | - | - | $\mathcal{O}(m^3)$ |
| 10 | $\mathcal{O}(1)$ | $\blacktriangleright \mathcal{O}(m)$ | $\mathcal{O}(m^3)$ |
| 11 | $\mathcal{O}(m^2)$ [Table 21] | $\blacktriangleright \mathcal{O}(m^3)$ | $\mathcal{O}(m^3)$ |
| 12 | - | - | $\mathcal{O}(m^3)$ |
| 13 | $\mathcal{O}(m^2)$ [Table 21] | $\blacktriangleright \mathcal{O}(m^3)$ | $\mathcal{O}(m^3)$ |
| 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 (\blacktriangleright) 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 | $\mathcal{O}(m)$ | $\blacktriangleright \mathcal{O}(m^2)$ | $\mathcal{O}(m^2)$ |
| 11 | - | - | $\mathcal{O}(m^2)$ |
| 12 | $\mathcal{O}(m)$ | $\mathcal{O}(m)$ | $\mathcal{O}(m^2)$ |
| 13 | $\mathcal{O}(m)$ | $\blacktriangleright \mathcal{O}(m^2)$ | $\mathcal{O}(m^2)$ |
| 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 (\blacktriangleright) denotes that a line is within a loop. An additional pointer (\triangleright) 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 | $\mathcal{O}(m)$ | $\blacktriangleright \mathcal{O}(m^2)$ | $\mathcal{O}(m^2)$ |
| 7 | $\mathcal{O}(m)$ | $\blacktriangleright \mathcal{O}(m^2)$ | $\mathcal{O}(m^2)$ |
| 8 | $\mathcal{O}(m)$ | $\blacktriangleright \mathcal{O}(m^2)$ | $\mathcal{O}(m^2)$ |
| 9 | - | - | $\mathcal{O}(m^2)$ |
| 10 | $\mathcal{O}(1)$ | $\blacktriangleright \mathcal{O}(m)$ | $\mathcal{O}(m^2)$ |
| 11 | $\mathcal{O}(1)$ | $\blacktriangleright \mathcal{O}(m)$ | $\mathcal{O}(m^2)$ |
| 12 | $\mathcal{O}(m^3)$ [Table 23] | $\blacktriangleright \triangleright \mathcal{O}(m^4)$ | $\mathcal{O}(m^4)$ |
| 13 | - | - | $\mathcal{O}(m^4)$ |
| 14 | $\mathcal{O}(1)$ | $\blacktriangleright \triangleright \mathcal{O}(m)$ | $\mathcal{O}(m^4)$ |
| 15 | $\mathcal{O}(m)$ | $\blacktriangleright \triangleright \mathcal{O}(m^2)$ | $\mathcal{O}(m^4)$ |
| 16 | $\mathcal{O}(m)$ | $\blacktriangleright \triangleright \mathcal{O}(m^2)$ | $\mathcal{O}(m^4)$ |
| 17 | $\mathcal{O}(m)$ | $\blacktriangleright \triangleright \mathcal{O}(m^2)$ | $\mathcal{O}(m^4)$ |
| 18 | - | - | $\mathcal{O}(m^4)$ |
| 19 | $\mathcal{O}(m)$ | $\blacktriangleright \triangleright \mathcal{O}(m^2)$ | $\mathcal{O}(m^4)$ |
| 20 | $\mathcal{O}(m)$ | $\blacktriangleright \triangleright \mathcal{O}(m^2)$ | $\mathcal{O}(m^4)$ |
| 21 | $\mathcal{O}(m)$ | $\blacktriangleright \triangleright \mathcal{O}(m^2)$ | $\mathcal{O}(m^4)$ |
| 22 | - | - | $\mathcal{O}(m^4)$ |
| 23 | $\mathcal{O}(m)$ | $\blacktriangleright \mathcal{O}(m^2)$ | $\mathcal{O}(m^4)$ |
| 24 | $\mathcal{O}(m)$ | $\blacktriangleright \mathcal{O}(m^2)$ | $\mathcal{O}(m^4)$ |
| 25 | $\mathcal{O}(m)$ | $\blacktriangleright \mathcal{O}(m^2)$ | $\mathcal{O}(m^4)$ |
| 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 (\blacktriangleright) 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 | $\mathcal{O}(m)$ | $\blacktriangleright \mathcal{O}(m)$ | $\mathcal{O}(m)$ |
| 12 | $\mathcal{O}(1)$ | $\blacktriangleright \mathcal{O}(1)$ | $\mathcal{O}(m)$ |
| 13 | $\mathcal{O}(m)$ | $\blacktriangleright \mathcal{O}(m)$ | $\mathcal{O}(m)$ |
| 14 | $\mathcal{O}(m)$ | $\blacktriangleright \mathcal{O}(m)$ | $\mathcal{O}(m)$ |
| 15 | - | - | $\mathcal{O}(m)$ |
| 16 | - | - | $\mathcal{O}(m)$ |
| 17 | - | - | $\mathcal{O}(m)$ |
| 18 | $\mathcal{O}(m)$ | $\mathcal{O}(1)$ | $\mathcal{O}(m)$ |
| 19 | $\mathcal{O}(m)$ | $\blacktriangleright \mathcal{O}(m)$ | $\mathcal{O}(m)$ |
| 20 | $\mathcal{O}(1)$ | $\blacktriangleright \mathcal{O}(1)$ | $\mathcal{O}(m)$ |
| 21 | $\mathcal{O}(m)$ | $\blacktriangleright \mathcal{O}(m)$ | $\mathcal{O}(m)$ |
| 22 | $\mathcal{O}(m)$ | $\blacktriangleright \mathcal{O}(m)$ | $\mathcal{O}(m)$ |
| 23 | - | - | $\mathcal{O}(m)$ |
| 24 | - | - | $\mathcal{O}(m)$ |
| 25 | - | - | $\mathcal{O}(m)$ |
| 26 | $\mathcal{O}(1)$ | $\mathcal{O}(m)$ | $\mathcal{O}(m)$ |
| 27 | $\mathcal{O}(m)$ | $\blacktriangleright \mathcal{O}(m^2)$ | $\mathcal{O}(m^2)$ |
| 28 | $\mathcal{O}(m^2)$ | $\blacktriangleright \mathcal{O}(m^3)$ | $\mathcal{O}(m^3)$ |
| 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 | $\mathcal{O}(1)$ | - | $\mathcal{O}(m^3)$ |

2004 9 Concluding Remarks

2005 In this two-part study on the vertex cover problem on cubic bridgeless graphs ($\text{VC} - \text{CBG}$), we
2006 discovered that: (i) $\text{VC} - \text{CBG}$ is **NP**-complete ([Theorem 1](#)) and (ii) $\text{VC} - \text{CBG} \in \mathbf{P}$ ([Theorem 2](#))⁴⁵.
2007 As a consequence of these two theorems combined with Proposition 1(c) in [[Coo00](#)], which states
2008 that if some language L is **NP**-complete and $L \in \mathbf{P}$, then $\mathbf{P} = \mathbf{NP}$, we get the following corollary:

2009 **Corollary 2.** $\mathbf{P} = \mathbf{NP}$.

2010 9.1 Brief Additional Remarks

2011 **Practical Consequences + Ethical Implications:** We presented a polynomial-time algorithm
2012 for an **NP**-complete problem. However, the problem space for this paper has been narrowed down
2013 to cubic bridgeless graphs ($\text{VC} - \text{CBG}$). Hence, the algorithm's practical utility depends on how
2014 generalizable it is for various graph types. Moreover, even for the narrowed-down problem of
2015 $\text{VC} - \text{CBG}$, the time complexity is a higher-order polynomial. Therefore, the time complexity will
2016 only worsen for the general case (and in turn, for using the algorithm for other **NP**-complete
2017 problems; see [Footnote 11](#)). Hence, the practical impact of our work is (none to) limited until more
2018 efficient versions of this (galactic) algorithm are found, for $\text{VC} - \text{CBG}$, for $\text{VC} - \text{CG}$, for VC , or for
2019 other **NP**-complete problems.

2020 Given that we do not expect any immediate practical consequences of our work, we do not
2021 expect any ethical implications either. That said, our work may eventually lead to algorithms that,
2022 for example, may break certain variants of cryptography. Hence, we stress the need for a study
2023 to understand the immediate and long-term implications of the algorithm to various fields. This
2024 study should at least (i) provide appropriate quantification (e.g., how long is “long-term”, or what
2025 order of the polynomial is not “practical” for how big a size of data in which field) and (ii) suggest
2026 alternative solutions wherever applicable (e.g., use information-theoretic security).

2027 References

- 2028 [Aar16] Scott Aaronson. $P = \lim\limits_{n \rightarrow \infty} \text{NP}$? *Open problems in mathematics*, pages 1–122, 2016.
- 2029 [ABLT06] Sanjeev Arora, Béla Bollobás, László Lovász, and Iannis Tourlakis. Proving integrality
2030 gaps without knowing the linear program. *Theory OF Computing*, 2:19–51, 2006.
- 2031 [AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Annals of mathematics*, 160(2):781–793, 2004.
- 2033 [AKS11] Per Austrin, Subhash Khot, and Muli Safra. Inapproximability of vertex cover and
2034 independent set in bounded degree graphs. *Theory of Computing*, 7(1):27–43, 2011.
- 2035 [ALM⁺98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy.
2036 Proof verification and the hardness of approximation problems. *Journal of the ACM (JACM)*,
2037 45(3):501–555, 1998.
- 2038 [AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characteriza-
2039 tion of np. *Journal of the ACM (JACM)*, 45(1):70–122, 1998.

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

- 2040 [AW09] Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory.
 2041 *ACM Transactions on Computation Theory (TOCT)*, 1(1):1–54, 2009.
- 2042 [BBDL01] Therese C Biedl, Prosenjit Bose, Erik D Demaine, and Anna Lubiw. Efficient algorithms
 2043 for petersen’s matching theorem. *Journal of Algorithms*, 38(1):110–134, 2001.
- 2044 [Ber57] Claude Berge. Two theorems in graph theory. *Proceedings of the National Academy of
 2045 Sciences*, 43(9):842–844, 1957.
- 2046 [BGS75] Theodore Baker, John Gill, and Robert Solovay. Relativizations of the $P = ? NP$ question.
 2047 *SIAM Journal on computing*, 4(4):431–442, 1975.
- 2048 [BIP19] Peter Bürgisser, Christian Ikenmeyer, and Greta Panova. No occurrence obstructions in
 2049 geometric complexity theory. *Journal of the American Mathematical Society*, 32(1):163–
 2050 193, 2019.
- 2051 [BJS22] Marin Bougeret, Bart MP Jansen, and Ignasi Sau. Bridge-depth characterizes which
 2052 minor-closed structural parameterizations of vertex cover admit a polynomial kernel.
 2053 *SIAM Journal on Discrete Mathematics*, 36(4):2737–2773, 2022.
- 2054 [BLW86] Norman Biggs, E Keith Lloyd, and Robin J Wilson. *Graph Theory, 1736–1936*. Oxford
 2055 University Press, 1986.
- 2056 [BM08] John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph theory*. Springer
 2057 Publishing Company, Incorporated, 2008.
- 2058 [BW15] Samuel R Buss and Ryan Williams. Limits on alternation trading proofs for time–space
 2059 lower bounds. *computational complexity*, 24:533–600, 2015.
- 2060 [CLRS16] Siu On Chan, James R Lee, Prasad Raghavendra, and David Steurer. Approximate
 2061 constraint satisfaction requires large l_p relaxations. *Journal of the ACM (JACM)*,
 2062 63(4):1–22, 2016.
- 2063 [CLRS22] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Intro-
 2064 duction to algorithms*. MIT press, 2022.
- 2065 [Cob65] Alan Cobham. The intrinsic computational difficulty of functions. In Yehoshua Bar-
 2066 Hillel, editor, *Logic, methodology and philosophy of science*, pages 24–30. North-Holland
 2067 Pub. Co., 1965.
- 2068 [Coo71] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the
 2069 third annual ACM symposium on Theory of computing*, pages 151–158, 1971.
- 2070 [Coo00] Stephen Cook. The P versus NP problem. *Clay Mathematics Institute*, 2(6):3, 2000.
- 2071 [Coo03] Stephen Cook. The importance of the P versus NP question. *Journal of the ACM (JACM)*,
 2072 50(1):27–29, 2003.
- 2073 [CS12] Maria Chudnovsky and Paul Seymour. Perfect matchings in planar cubic graphs. *Com-
 2074 binatorica*, 32(4):403–424, 2012.
- 2075 [CŽ24] Lorenzo Ciardo and Stanislav Živný. Semidefinite programming and linear equations
 2076 vs. homomorphism problems. In *Proceedings of the 56th Annual ACM Symposium on
 2077 Theory of Computing*, pages 1935–1943, 2024.

- 2078 [DF95] Rod G Downey and Michael R Fellows. Fixed-parameter tractability and completeness
2079 i: Basic results. *SIAM Journal on computing*, 24(4):873–921, 1995.
- 2080 [DF12] Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer
2081 Science & Business Media, 2012.
- 2082 [DGKR03] Irit Dinur, Venkatesan Guruswami, Subhash Khot, and Oded Regev. A new multilay-
2083 ered pcp and the hardness of hypergraph vertex cover. In *Proceedings of the thirty-fifth*
2084 *annual ACM symposium on Theory of computing*, pages 595–601, 2003.
- 2085 [Din07] Irit Dinur. The pcp theorem by gap amplification. *Journal of the ACM (JACM)*,
2086 54(3):12–es, 2007.
- 2087 [DIP20] Julian Dörfler, Christian Ikenmeyer, and Greta Panova. On geometric complexity the-
2088 ory: Multiplicity obstructions are stronger than occurrence obstructions. *SIAM Journal*
2089 *on Applied Algebra and Geometry*, 4(2):354–376, 2020.
- 2090 [DS05] Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover.
2091 *Annals of mathematics*, pages 439–485, 2005.
- 2092 [Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–
2093 467, 1965.
- 2094 [EKK⁺11] Louis Esperet, František Kardoš, Andrew King, Daniel Král’, and Sergey Norine.
2095 Exponentially many perfect matchings in cubic graphs. *Advances in Mathematics*,
2096 227(4):1646–1664, 2011.
- 2097 [EKK12] Louis Esperet, František Kardoš, and Daniel Král’. A superlinear bound on the number
2098 of perfect matchings in cubic bridgeless graphs. *European Journal of Combinatorics*,
2099 33(5):767–798, 2012. EuroComb ’09.
- 2100 [EKŠŠ10] Louis Esperet, Daniel Král’, Petr Škoda, and Riste Škrekovski. An improved linear
2101 bound on the number of perfect matchings in cubic graphs. *European Journal of Com-
2102 binatorics*, 31(5):1316–1334, 2010.
- 2103 [EPL82] Jack Edmonds, WR Pulleyblank, and L Lovász. Brick decompositions and the matching
2104 rank of graphs. *Combinatorica*, 2:247–274, 1982.
- 2105 [Eul36] Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii
2106 academiae scientiarum Petropolitanae*, pages 128–140, 1736.
- 2107 [Fei03] Uriel Feige. Vertex cover is hardest to approximate on regular graphs. *Technical Report
2108 MCS03–15*, 2003.
- 2109 [FFR97] Ralph Faudree, Evelyne Flandrin, and Zdeněk Ryjáček. Claw-free graphs—a survey.
2110 *Discrete Mathematics*, 164(1-3):87–147, 1997.
- 2111 [FGL⁺96] Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Inter-
2112 active proofs and the hardness of approximating cliques. *Journal of the ACM (JACM)*,
2113 43(2):268–292, 1996.
- 2114 [FMP⁺15] Samuel Fiorini, Serge Massar, Sebastian Pokutta, Hans Raj Tiwary, and Ronald
2115 De Wolf. Exponential lower bounds for polytopes in combinatorial optimization. *Jour-
2116 nal of the ACM (JACM)*, 62(2):1–23, 2015.

- 2117 [For09] Lance Fortnow. The status of the p versus np problem. *Communications of the ACM*,
 2118 52(9):78–86, 2009.
- 2119 [For21] Lance Fortnow. Fifty years of p vs. np and the possibility of the impossible. *Communi-
 2120 cations of the ACM*, 65(1):76–85, 2021.
- 2121 [Gib85] Alan Gibbons. *Algorithmic graph theory*. Cambridge university press, 1985.
- 2122 [GJ02] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh
 2123 freeman New York, 2002.
- 2124 [GJS74] Michael R Garey, David S Johnson, and Larry Stockmeyer. Some simplified np-complete
 2125 problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing*,
 2126 pages 47–63, 1974.
- 2127 [GNW07] Jiong Guo, Rolf Niedermeier, and Sebastian Wernicke. Parameterized complexity of
 2128 vertex cover variants. *Theory of Computing Systems*, 41:501–520, 2007.
- 2129 [GW24] Paweł Gawrychowski and Mateusz Wasylkiewicz. Finding perfect matchings in bridge-
 2130 less cubic multigraphs without dynamic (2-) connectivity. In *32nd Annual European
 2131 Symposium on Algorithms (ESA 2024)*, pages 59–1. Schloss Dagstuhl–Leibniz-Zentrum
 2132 für Informatik, 2024.
- 2133 [Hal35] P Hall. On representatives of subsets. *Journal of the London Mathematical Society*,
 2134 1(1):26–30, 1935.
- 2135 [Hås01] Johan Håstad. Some optimal inapproximability results. *Journal of the ACM (JACM)*,
 2136 48(4):798–859, 2001.
- 2137 [IP17] Christian Ikenmeyer and Greta Panova. Rectangular kronecker coefficients and
 2138 plethysms in geometric complexity theory. *Advances in Mathematics*, 319:40–66, 2017.
- 2139 [Kar72] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of Com-
 2140 puter Computations*, pages 85–103. Springer, 1972.
- 2141 [Kho02] Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the
 2142 thiry-fourth annual ACM symposium on Theory of computing*, pages 767–775, 2002.
- 2143 [Kho19] Subhash Khot. On the proof of the 2-to-2 games conjecture. *Current Developments in
 2144 Mathematics*, 2019(1):43–94, 2019.
- 2145 [KMS23] Subhash Khot, Dor Minzer, and Muli Safra. Pseudorandom sets in grassmann graph
 2146 have near-perfect expansion. *Annals of Mathematics*, 198(1):1–92, 2023.
- 2147 [Kon31] Denés Konig. Gráfok és mátrixok. matematikai és fizikai lapok, 38: 116–119, 1931.
- 2148 [KR08] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within
 2149 $2 - \varepsilon$. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.
- 2150 [KSS09] Daniel Král’, Jean-Sébastien Sereni, and Michael Stiebitz. A new lower bound on the
 2151 number of perfect matchings in cubic graphs. *SIAM Journal on Discrete Mathematics*,
 2152 23(3):1465–1483, 2009.
- 2153 [KT06] Jon Kleinberg and Eva Tardos. *Algorithm design*. Pearson Education India, 2006.

- 2154 [Lev73] Leonid Anatolevich Levin. Universal sequential search problems. *Problemy peredachi informatsii*, 9(3):115–116, 1973.
- 2155
- 2156 [LP09] László Lovász and Michael D Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.
- 2157
- 2158 [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.
- 2159
- 2160
- 2161 [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.
- 2162
- 2163
- 2164 [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.
- 2165
- 2166 [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.
- 2167
- 2168 [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.
- 2169
- 2170
- 2171 [Mul99] Ketan Mulmuley. Lower bounds in a parallel model without bit operations. *SIAM Journal on Computing*, 28(4):1460–1509, 1999.
- 2172
- 2173 [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.
- 2174
- 2175 [Mul12] Ketan D Mulmuley. The gct program toward the p vs. np problem. *Communications of the ACM*, 55(6):98–107, 2012.
- 2176
- 2177 [MV80] Silvio Micali and Vijay V Vazirani. An $\mathcal{O}(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. In *21st Annual symposium on foundations of computer science (Sfcs 1980)*, pages 17–27. IEEE, 1980.
- 2178
- 2179
- 2180 [Oum11] Sang-il Oum. Perfect matchings in claw-free cubic graphs. *The Electronic Journal of Combinatorics*, 18-1:P62–1–P62–6, 2011.
- 2181
- 2182 [Pan23] Greta Panova. Computational complexity in algebraic combinatorics, appeared in current developments in mathematics, 2023, 2023.
- 2183
- 2184 [Pet91] Julius Petersen. Die theorie der regulären graphs. *Acta Mathematica*, 1891.
- 2185 [Raz85a] Alexander Razborov. Lower bounds on the monotone complexity of some boolean function. In *Soviet Math. Dokl.*, volume 31, pages 354–357, 1985.
- 2186
- 2187 [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.
- 2188
- 2189 [Rel24] Kunal Relia. On efficient computation of dire committees. *arXiv preprint arXiv:2402.19365*, 2024.
- 2190

- 2191 [Rot17] Thomas Rothvoß. The matching polytope has exponential extension complexity. *Journal
2192 of the ACM (JACM)*, 64(6):1–19, 2017.
- 2193 [RR94] Alexander A Razborov and Steven Rudich. Natural proofs. In *Proceedings of the
2194 twenty-sixth annual ACM symposium on Theory of computing*, pages 204–213, 1994.
- 2195 [Sbi80] Najiba Sbihi. Algorithme de recherche d'un stable de cardinalité maximum dans un
2196 graphe sans étoile. *Discrete Mathematics*, 29(1):53–76, 1980.
- 2197 [Tur36] Alan M Turing. On computable numbers, with an application to the entscheidungsproblem.
2198 *Proceedings of the London Mathematical Society*, 42(1):230–265, 1936.
- 2199 [Tur38] Alan M Turing. On computable numbers, with an application to the entscheidungsproblem.
2200 a correction. *Proceedings of the London Mathematical Society*, 2(1):544–546, 1938.
- 2201 [Tut47] William T Tutte. The factorization of linear graphs. *Journal of the London Mathemat-
2202 ical Society*, 1(2):107–111, 1947.
- 2203 [Val79a] Leslie G Valiant. Completeness classes in algebra. In *Proceedings of the eleventh annual
2204 ACM symposium on Theory of computing*, pages 249–261, 1979.
- 2205 [Val79b] Leslie G Valiant. The complexity of computing the permanent. *Theoretical computer
2206 science*, 8(2):189–201, 1979.
- 2207 [Var10] Moshe Y Vardi. on p, np, and computational complexity. *Communications of the ACM*,
2208 53(11):5–5, 2010.
- 2209 [VL91] Jan Van Leeuwen. *Handbook of theoretical computer science (vol. A) algorithms and
2210 complexity*. Mit Press, 1991.
- 2211 [Voo79] Marc Voorhoeve. A lower bound for the permanents of certain (0, 1)-matrices. In
2212 *Indagationes Mathematicae (Proceedings)*, volume 82-1, pages 83–86. Elsevier, 1979.
- 2213 [Wes01] Douglas Brent West. *Introduction to graph theory*, volume 2. Pearson Prentice hall
2214 Upper Saddle River, 2001.
- 2215 [Wig06] Avi Wigderson. P, np and mathematics—a computational complexity perspective. In
2216 *Proceedings of the ICM*, volume 6, pages 665–712, 2006.
- 2217 [Wig09] Avi Wigderson. Knowledge, creativity and p versus np. URL <http://www.math.ias.edu/~avi/PUBLICATIONS/MYPAPERS/AW09/AW09.pdf>. Circulated manuscript,
2218 2009.
- 2219 [Wil14] Ryan Williams. Nonuniform acc circuit lower bounds. *Journal of the ACM (JACM)*,
2220 61(1):1–32, 2014.
- 2221 [Wil19] R Ryan Williams. Some estimated likelihoods for computational complexity. *Computing
and Software Science: State of the Art and Perspectives*, pages 9–26, 2019.
- 2222 [Wil25] R Ryan Williams. Simulating time with square-root space. In *Proceedings of the 57th
2225 Annual ACM Symposium on Theory of Computing*, pages 13–23, 2025.
- 2226 [Yan88] Mihalis Yannakakis. Expressing combinatorial optimization problems by linear pro-
2227 grams. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*,
2228 pages 223–228, 1988.

2229 A Selection of Simple Connected Graphs

2230 We hand-waved the selection of simple connected graphs for our study (e.g., see [Footnote 2](#) and
2231 [Footnote 19](#)). Hence, we extensively discuss why this selection was worthy of hand-waving rather
2232 than a deeper discussion. Consequently, we reiterate that all our results are unconditional⁴⁶.

2233 Let us zoom out to discuss our choice of the graph used in this paper from the family of graphs
2234 (2-uniform hypergraphs). Foremost, it is clear by now that we use cubic bridgeless graphs. As is
2235 the norm in literature, all graphs are unweighted undirected finite graphs. Additionally, w.l.o.g.,
2236 we stated that the considered graphs are simple connected graphs. We clarify the last choice.

2237 **Connected Graphs:** We first discuss the requirement that graphs are connected.

- 2238 • **Graph Theory:** The assumption on the graph being connected is standard in graph theory
2239 literature about papers on matching theory. For instance, Line 35 on Page Number 843 (just
2240 below Theorem 1) in [\[Ber57\]](#) assumes the graph to be connected. This is because in the
2241 context of such work (and our paper), a technique that works for one connected component
2242 implicitly works for each connected component of a given graph, and in turn, for the entire
2243 graph, assuming all components share the common properties. Hence, when a theorem holds
2244 for a connected component of a given graph, it implies that it holds for the entire graph.
- 2245 • **Computational Complexity:** Our results in both parts of the paper hold when we have
2246 a connected graph. Hence, if an unconnected graph is given, our results will hold for each
2247 connected component of the graph. We simply need to take the union of the outcomes of
2248 each connected component to get the overall outcome. For instance, executing Lines 1 to 10
2249 of [Algorithm 1](#) for each connected component and eventually taking the union of the vertex
2250 covers S we get in Line 10 indeed results in a minimum vertex cover.

2251 **Simple Graphs:** We now discuss the requirement that graphs are simple. A simple graph, by
2252 definition, is undirected.

- 2253 • **Graph Theory:** If a matching theory (graph theory) result holds for general graphs, it
2254 implies it will hold for the special case of simple graphs. Hence, each known result we use
2255 in the paper holds for simple graphs, too. Our results are particularly designed to hold for
2256 simple graphs.
- 2257 • **Computational Complexity:** From the computational complexity perspective, we dis-
2258 cussed how $\text{VC} - \text{CBG}$ can alternatively be proven to be **NP**-complete by reducing from the VC
2259 on cubic simple graphs by using the same construction we discussed in Part I. Before that,
2260 the VC on cubic simple graphs can be proven to be **NP**-complete by reducing from $\text{VC} - \text{CG}$.
2261 More specifically, if the given graph is not simple, we first remove each multiple edge and
2262 each loop. Then, we add an edge to each vertex that has a degree less than three, such that
2263 the edge connects the vertex to a subgraph of five dummy vertices ([Figure 13](#)). This ensures
2264 the graph remains cubic while becoming simple⁴⁷. Each subgraph needs 3 vertices to form
2265 an MVC.

⁴⁶The 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.

⁴⁷Alternatively, for each edge, split it and insert Block Type B_1 (but not Type B_2) as discussed in Part I.

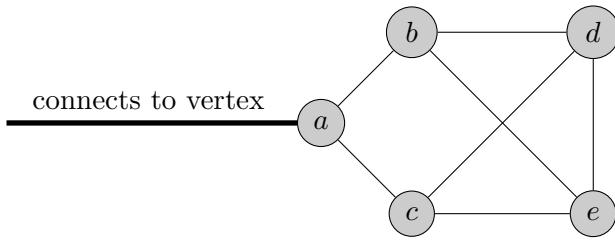


Figure 13: A subgraph of dummy vertices used to make a cubic graph simple.

2266 In summary, in the context of this paper, all our computational complexity and graph-theoretic
2267 results hold when we use simple connected graphs. Overall, for the sake of completeness, the
2268 unconditional results in the paper use finite unweighted simple connected cubic bridgeless graphs.