Tribhuvan University

Institute of Science and Technology

Comparative Evaluation of Minimum Degree Based Approximation Algorithms for Minimum Vertex Cover problem

Dissertation

Submitted To

Central Department of Computer Science and Information Technology

Kirtipur, Kathmandu, Nepal

In partial fulfillment of the requirements for the

Master's Degree in Computer Science and Information Technology

By

Santosh Kumar Mahato

Date: April 2, 2017

Tribhuvan University

Institute of Science and Technology

Comparative Evaluation of Minimum Degree Based Approximation Algorithms for Minimum Vertex Cover problem

Dissertation

Submitted to

Central Department of Computer Science and Information Technology

Kirtipur, Kathmandu, Nepal

In partial fulfillment of the requirements for the

Master's Degree in Computer Science and Information Technology

By

Santosh Kumar Mahato

Date: April 2, 2017

Supervisor

Mr. Arjun Singh Saud

Tribhuvan University

Institute of Science and Technology

Central Department of Computer Science and Information Technology

Student's Declaration

I hereby declare that I am the only author of this work and that no sources other than the listed here have been used in this work.

.....

Santosh Kumar Mahato

Date: April 2, 2017

Supervisor's Recommendation

I hereby recommend that this dissertation prepared under my supervision by **Mr.** Santosh Kumar Mahato entitled "**Comparative Evaluation of Minimum Degree Based Approximation Algorithms for Minimum Vertex Cover problem**" in partial fulfillment of the requirements for the degree of M. Sc. in Computer Science and Information Technology be processed for the evaluation.

… … … … … … … … … …

Mr. Arjun Singh Saud

CDCSIT, TU

Kirtipur, Kathmandu, Nepal

Date: April 2, 2017

Tribhuvan University

Institute of Science and Technology

Central Department of Computer Science and Information Technology

LETTER OF APPROVAL

We certify that we have read this dissertation and in our opinion it is satisfactory in the scope and quality as a dissertation in the partial fulfillment for the requirement of Masters Degree in Computer Science and Information Technology.

Evaluation Committee

... ...

Mr. Nawaraj Paudel
CDCSIT, Tribhuvan University,
Kathmandu, Nepal
(Head)

... ...

Mr. Arjun Singh Saud
CDCSIT, Tribhuvan University,
Kathmandu, Nepal
(Supervisor)

... ...

Asst. Prof. Arun K. Timalsina, PhD
Department of Electronics and Computer Engineering, Pulchowk Campus,
Institute Of Engineering,
Tribhuvan University,
Pulchowk , Lalitpur, Nepal
(External Examiner)

... ...

Mr. Sarbin Sayami
CDCSIT, Tribhuvan University,
Kathmandu, Nepal
(Internal Examiner)

Date: 13 April, 2017

ACKNOWLEDGEMENT

At first I thank God who gave ability, strength and confidence in me to complete this work. It is a great pleasure for me that I have completed this dissertation work and want to acknowledge the contributions of all the individuals to this work. I express my genuine thanks to my supervisor Mr. Arjun Singh Saud , for his valuable guidance in carrying out this work under his effective supervision and for providing support throughout this dissertation work. I want to express sincere thanks to Asst. Prof. Nawraj Paudel, Head of Department for his encouragement and support regarding the dissertation work. Next, Iam also very thankful to my colleague Mr. Ashok Pant, Mahesh Kumar Yadav and Mr. Bhim Rawat for their technical help and support.

I am also thankful to all the staff members of the Department of Computer Science and Information technology, TU (Kathmandu, Nepal) for their full cooperation and help. Thanks to all my friends for their supports and help in creation documentation.

Finally, I thank my family for their love, support and encouragement.

ABSTRACT

Minimum vertex cover(MVC) problem is a NP Complete optimization problem that attracts many researchers due to its wide range of application in real life problems. As MVC is NP-complete, there are no any algorithm that finds optimal solution to MVC problem in polynomial time. Numerous of approaches have been proposed among which approximation approach is much favored in the field of MVC as it guarantees to give a solution that is near to optimal or sub optimal solution. There is a number of MVC algorithm based on approximation approaches that constructs vertex cover .This Dissertation work is focused on a comparative study of three recent approximation approach based algorithms ,NOVCA,CSSA and NMVSA. The performance of each algorithm is measured in terms of approximation ratio and step count. Here step count is used as second performance metrics because the performance differences in approximation ratio of all the algorithms are relatively small. In the dissertation work, benchmark graph datasets are used for the comparison of algorithms and an extensive analysis have been provided to help the selection of efficient algorithm.

Keywords: Minimum vertex cover, Approximation algorithms, Approximation ratio, Near optimal vertex cover algorithm(NOVCA), New modified vertex support algorithm(NMVSA), Clever Steady Strategies Algorithm(CSSA)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ACO: Ant Colony Optimization

AVSA: Advance Vertex Support Algorithm

CNF: Conjuctive Normal Form

CSSA: Clever Steady Strategies Algorithm

DC: Degree Contribution

DCA: Degree Contribution Algorithm

DFS: Depth First Search

GIC: Greedy Independent Cover

GVCP: General Vertex Cover Problem

HVX: Heuristic Vertex Cover

IDE: Integrated Development Environment

MC: Maximum Clique

MDG: Maximum Degree Greedy

MIS: Maximum Independent Set

MVC: Minimum Vertex Cover

MVSA: Modified Vertex Support Algorithm

NMVSA: New Modified Vertex Support Algorithm

NOVCA: Near Optimal Vertex Cover Algorithm

NP: Non Deterministic Polynomial time

NPC: Non Deterministic Polynomial time Complete

P: Polynomial time

VCP: Vertex Cover Problem

VCUMI: Vertex Cover Using Maximum Independent Set

VLSI: Very Large Scale Integration

VSA: Vertex Support Algorithm

CHAPTER 1
INTRODUCTION AND BACKGROUND

1.1 Introduction

Vertex cover of a graph represents the sub set of vertices that are sufficient to cover all the edges of the undirected graph. Vertex cover problem is one of the graph related problem where the objective is to determine a set of vertices of a graph that covers allthe edges of the graph. It is an important NP Complete problem that has been extensively researched and reviewed by various researchers.

Minimum vertex cover problem is an important combinatorial optimization problem with a goal of finding a vertex cover of smallest possible size for the given graph.MVC has many real-world applications, such as network security, scheduling, VLSI design and industrial machine assignment. To find an optimal solution to minimum vertex cover is very difficult, but to get an alternative approximate or sub optimal solution is easier. So, many approximation algorithms have been proposed to construct vertex cover indifferent ways. Such algorithms have a polynomial time complexity and return a solution that is close to optimal solution. The quality of solution produced by approximation algorithm is traditionally measured in terms of approximation ratio of the solution and the optimal one. Approximation ratio is not necessarily a constant. If it is small, the quality of solution is good and the algorithm is considered to be efficient. But with ongoing research, there are several approximation algorithms for VCP that provide good quality solutions with almost same approximation ratios. So the efficiency of the algorithms cannot be measured by using only approximation ratios.

This Dissertation work explores the different approximation techniques that have been successfully applied to VCP, and compares three recent algorithms to determine the efficient one among them. In order to do so, an extensive literature review has been carried out and an experimental result, that compares these three algorithms in terms of different performance metrics, is provided in this dissertation work. The two comparison parameters: approximation ratio and step count, are used to compare the performance of algorithms. Here step count is used as second performance metrics because the differences in approximation ratio of these algorithms are relatively very small. Different standard benchmark graph datasets available

across the repository of internet are used for the comparing of algorithms performance and comparative analysis have been provided for the selection of efficient algorithm.

1.2 Problem Definition

The minimum vertex cover problem is the optimization problem of finding a smallest vertex cover in a given graph. Since it is a NP complete, there is no any algorithms that can find a optimal solution to this problem in polynomial time of the size of input. Among the different approaches to deal with NP-complete problems, approximation algorithms are most preferred technique as the size of the vertex-cover returned by these approximation algorithms is guaranteed to be no more than twice the size of an optimal vertex-cover. There are different Approximation algorithms for the vertex-cover problem that returns a solution that is near to optimal. Many of these algorithms claim that it is better or have best known approximation ratio for finding minimum vertex cover. Thus comparative evaluation of such algorithms is worth in computational complexity theory.

1.3 Objective

The main objective of this dissertation work is:

- To study the approximation algorithms for finding minimum vertex cover in a given undirected graph.
- To evaluate and to compare the performance of three recent approximation algorithms in terms of approximation ratio and step count.
- To suggest an approximation algorithm that has best approximation ratio and which gives best suboptimal/optimal solution to a wide range of input graphs.

1.4 Motivation

Vertex cover problem is one of the most explored NP complete problems due to its wide range of applicability to real world problem. Researchers are providing different innovative ideas to solve the minimum vertex cover problem efficiently and as fast as possible. A number of algorithms are presented one after another that are both efficient and fast in their own context but it is difficult to decide which algorithm is appropriate for the use with large variety of inputs. Therefore it is a worth work to compare existing algorithms that claim to be efficient and to provide an extensive analysis of these algorithms.

1.5 Background

1.5.1 The Vertex Cover Problem

In graph theory, a vertex cover of a graph is a set of vertices such that each edge of the graph is incident to at least one vertex of the set. It is a set of vertices in a graph such that every edge in the graph is covered by the vertices in the cover set. Formally, A vertex cover of an undirected graph G =(V,E) is a subset $V' \subseteq V$ such that if $(u,v) \in E$, then $u \in V'$ or $v \in V'$ (or both). That is, each vertex "covers" its incident edges, and a vertex cover for G is a set of vertices that covers all the edges in E. The size of a vertex cover is the number of vertices in it. The following Figure 1.1 shows two examples of vertex covers, where vertex cover V' is shaded [1].

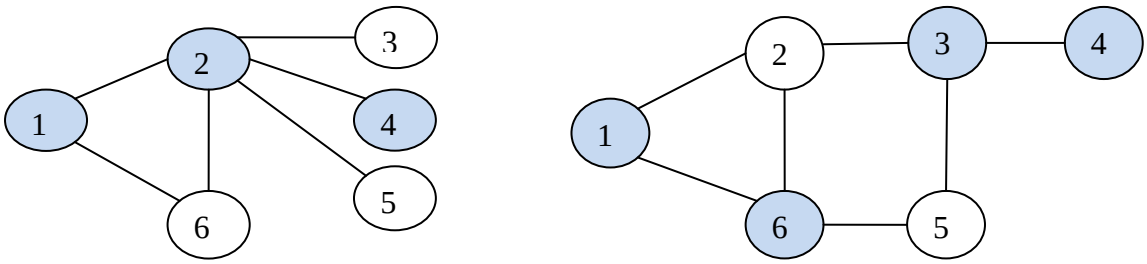


Figure 1.1: Vertex Cover examples

In above example, a possible vertex cover are {1,2,4} and {1,3,4,6}. The number of vertices in vertex cover gives its size. The vertex-cover problem is a problem that aims to find a vertex cover of minimum size in a given graph. Such a vertex cover is also called a minimum vertex cover or optimal vertex cover. It is a classical optimization problem in computer science and is a typical example of an NP hard optimization problem that has approximation algorithms. The following Figure 1.2 shows examples of minimum vertex cover for the graphs in Figure1.1, where minimum vertex cover is shaded.

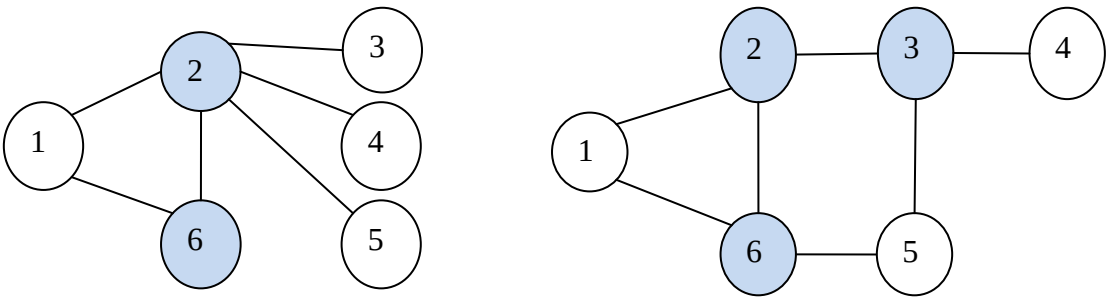


Figure1.2:Minimum vertex cover

In above example, minimum vertex cover for graphs in Figure 1.1 are {2,6} and {2,3,6}.

There are two forms of minimum vertex cover: Optimization and Decision.

The **minimum vertex cover problem** is the optimization problem of finding a minimum sized vertex cover in a given graph.

INSTANCE: Graph G

OUTPUT: Smallest number k such that G has a vertex cover of size k.

If the problem is stated as a decision problem, the problem is to determine whether a graph has a vertex cover of a given size. That is verification is done to analyze the vertex cover of a specified size.

INSTANCE: Graph G and positive integer k.

QUESTION: Does G have a vertex cover of size at most k ? [1]

This decision version was one of Karp's 21 NP-complete problems and is therefore a classical NP-complete problem in computational complexity theory. Furthermore, it (the vertex cover problem) is fixed-parameter tractable and a central problem in parameterized complexity theory. It is often used in computational complexity theory as a starting point for NP-hardness proofs [1]. Since It is a NP – complete problem, so there are no any algorithm that can solve it exactly in polynomial time. Because of the NP-Completeness and wide range of applications, vertex cover problem, especially minimum vertex cover problem has been an important research topic among researchers. Many real world problems can be formulated as an instance of minimum vertex cover problem. Example areas are communication network, civil and electrical engineering, bio-informatics etc. One typical application of minimum vertex cover problem is in installing cameras on traffic lights to cover each road on a map can be modeled as minimum vertex cover problem where lights representing vertex and roads as edges and job is to install the minimum number of cameras so every road has at least one camera on its either end [2].

Even though it is not possible to find an optimal vertex cover in a graph G in polynomial time, a vertex cover that is near to optimal can be found efficiently in reasonable time. Approximation algorithms play a critical role in obtaining near to optimal solution to all NP-Complete problems[3].

1.5.2 Application of Vertex Cover

The vertex cover problem is a NP complete class problem in terms of complexity. Due to NP class problem this problem mainly attracts researcher towards it. Other main reason is it's relevant to real world applications. The minimum vertex cover (MVC) problem is very popular due to its real life applications. MVC is used in civil and electrical engineering, VLSI design, MAP labeling, computer networking, sensor networks and Bioinformatics [4].

One of the real life application examples of the minimum vertex cover is the deployment of guards on the corridors in a museum. Each edge corresponds to corridor in a museum and each corridor intersection corresponds to a vertex in the graph. The problem here is to deploy guards in such a way to cover each corridor of a museum with minimum number of guards. This problem can be solved by using the minimum vertex cover algorithms. [4].

Application Example in sensor network: The paper [5] addresses a problem in wireless sensors network to increase the network life time and proposed a solution for it using vertex cover algorithm. The proposed algorithms adopt the vertex cover technique to enhance the live-time of the wireless sensor network by selecting minimum number of nodes that reduce power consumption. The minimum (optimal) number of vertices (sensors) problem is NP-Complete. Therefore, a near-optimal solution to the problem can be obtained by an approximation algorithm that solves the problem in polynomial time.

The vertex cover problem is also closely related to many other hard graph problems such as the problems of maximum clique and independent set problems and so it interests the researchers in the field of design of optimization and approximation algorithms. For instance, the independent set problem [3] is similar to the minimum vertex cover problem because a minimum vertex cover defines a maximum independent set and vice versa. Another interesting problem that is closely related to the minimum vertex cover is the edge cover which searches for the smallest set of edges such that each vertex is included in one of the edges[6].

Another example of a practical application is the hitting set problem arises in efficient dynamic detection of race conditions. In this case, each time global memory is written, the current thread and set of locks held by that thread are stored. Under lockset-based detection, if later another thread writes to that location and there is not a race, it must be because it holds at least one lock in common with each of the previous writes. Thus the size of the

hitting set represents the minimum lock set size to be race-free. This is useful in eliminating redundant write events, since large lock sets are considered unlikely in practice[1].

1.5.3 Problems Related to Vertex cover

Vertex cover problem is closely related to two other popular optimization problems: MC and MIS. The Figure 1.3 shows an example. These three problems actually can be considered as three different forms of the same problem[6].

1.5.3.1 Clique problem

A clique in an undirected graph G= (V, E) is a subset V' ⊆V of vertices, eachpair of which is connected by an edge in E. In other words, a clique is a completesub-graph of G. The size of a clique is the number of vertices it contains. The Cliqueproblem is the optimization problem of finding a clique of maximum size in a graph.As a decision problem, we ask simply whether a clique of a given size k exists in thegraph [3].

Instance: a graph G= (V, E) and a positive integer k ≤ |V|.

Question: is there a clique V' ⊆ V of size ≥ k?

1.5.3.2 Independent-set problem

An independent set of a graph G= (V, E) is a subset V' ⊆V of vertices such thateach edge in E is incident on at most one vertex in V'. The independent set problem isto find a maximum-size independent set in G [3].

Instance: a graph G= (V, E) and a positive integer k ≤ |V|.

Question: is there an independent set of size ≥ k?

The following are equivalent for G= (V, E) and a subset V' of V and G = (V,$\bar{E}$), where

$\bar{E}$ = {(u, v): u, v ∉V, u≠ v, and (u, v) ∉ E}

(a). V' is a clique of G.

(b). V' is an independent of G

(c). V-V' is a vertex-cover of G

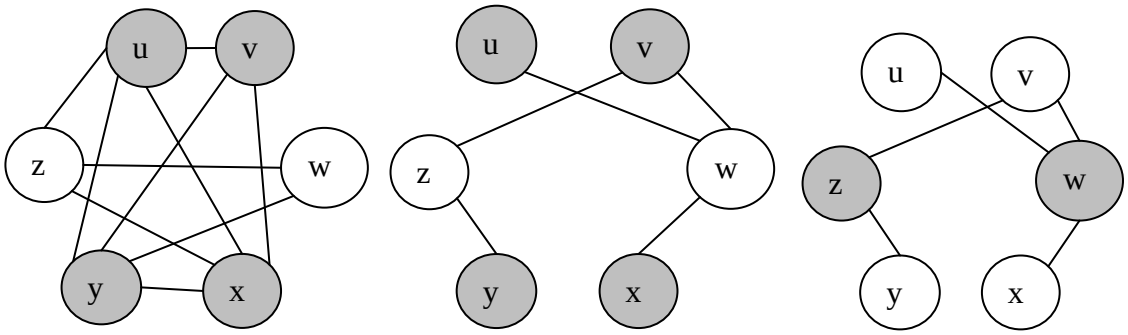


Figure1.3: Relationship of VC with other NP-problems.

(a) Graph G(V,E) with Clique V'={u,v,x,y} (b)Complement graph G' with Independent set V'={u,v,x,y} (c) Vertex cover for graph in Figure (b) is V-V'={w,z}

1.5.4 Complexity Classes of Problems: P, NP, and NPC

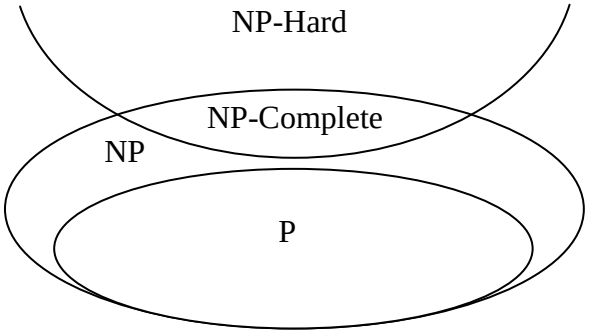


Figure 1.4 : Relationship among P,NP, NP-complete and NP Hard Problems

The class P consists of those problems that are solvable in polynomial time. More specifically, they are problems that can be solved in time $O(n^k)$ for some constant k, where n is the size of the input to the problem. The class NP consists of those problems that are “verifiable” in polynomial time. The abbreviation NP refers to "nondeterministic polynomial time". That is if a “certificate” of a solution to a problem is given, then it can be verified that the "certificate" is correct, in time polynomial of the size of the input to the problem. For example, minimum vertex cover problem, the Hamiltonian cycle problem, 3-CNF satisfiability etc. are NP complete problems. Any problem in P is also in NP, since if a problem is in P then it can be solved in polynomial time without even being supplied a certificate.

NP-hard problems are partly similar but more difficult problems than NP complete problems. They don't themselves belong to class NP, but all problems in class NP can be reduced to them. Very often, the NP-hard problems really require exponential time or even worse.

NP-complete problems are a subset of NP-hard problems, and that's why NP-complete problems are sometimes called NP-hard. In computational complexity theory, a decision problem is NP-complete when it is both in NP and NP-hard. The set of NP-complete problems is often denoted by NP-C or NPC[3,7]. Figure 1.4 shows the Relationship among P, NP, NP-complete and NP Hard Problem.

1.5.5 Coping with NP-completeness/ Solving NP-complete problems

There are many important optimization problems with NP-completeness that may be quite hard to solve exactly. Due to the theoretical and practical importance NP complete problems must be solved, as solutions to these problems are useful in various fields. Various algorithmic approaches have been used to tackle NP-complete problems. At present, all known algorithms for NP-complete problems require time that is super polynomial in the input size, and it is unknown whether there are any faster algorithms. The following techniques [7] can be applied to solve computational problems in general, and they often give rise to substantially faster algorithms:

Approximation: The idea is that instead of searching for an optimal solution, a solution that is near to optimal or "almost" optimal can be determined. This is an algorithm that runs in polynomial time (ideally), and produces a solution that is within a guaranteed factor of the optimum solution.

Randomization: Randomness can be used to get a faster average running time, and allow the algorithm to fail with some small probability. An algorithm that uses random numbers to decide what to do next anywhere in its logic is called Randomized Algorithm. The algorithm typically uses uniformly random bits as an auxiliary input to guide its behavior, in the hope of achieving good performance in the "average case" over all possible choices of random bits. Formally, the algorithm's performance will be a random variable determined by the random bits; thus either the running time, or the output (or both) are random variables.

General Search Methods: There are a number of very powerful techniques for solving general combinatorial optimization problems that have been developed in the areas of AI and operations research such as branch-and-bound, A*-search, simulated annealing, and genetic algorithms. The performance of these approaches varies considerably from one problem to problem and instance to instance. But in some cases they can perform quite well.

Restriction: By restricting the structure of the input (e.g., to planar graphs), faster algorithms are usually possible. Narrowing the problem space helps to solve the problem. For instance, if we can't solve TSP on general graphs, we can try to just solve it for graphs obeying a Euclidean distance metric.

Parameterization: If certain parameters of the input are fixed, faster algorithms can be designed. These algorithms depends on a parameter say 'k' of the given problem . Generally the algorithm running time is exponential on k but not on the input size. Hence for smaller value of k, a solution can be found.

Heuristic: A heuristic is a strategy for producing a valid solution, but there are no guarantees how close it is to optimal. These algorithms works “reasonably well” in many cases, but the quality of result and efficiency of the algorithm may not be always so good. This is worthwhile if all else fails, or if lack of optimality is not really an issue. A heuristic is a function that ranks alternatives in search algorithms at each branching step based on available information to decide which branch to follow. Metaheuristic approaches are often used. A metaheuristic is a heuristic designed to select a heuristic (partial search algorithm) that may offer a satisfactorily good solution to an optimization problem, especially with incomplete information or limited computation capacity. Metaheuristics sample a set of solutions which is too large to be completely sampled. Metaheuristics may make few assumptions about the optimization problem being solved, and so they may be usable for a variety of problems.

1.6 Contribution Of This Dissertation

The main contribution of this dissertation to the field of vertex cover problem is that this dissertation work compares three recent algorithms experimentally on more than 70 different instances of different benchmark graphs datasets. This experimental comparison

provides clear result that helps in the selection process of the minimum vertex cover algorithms for solving the real world problem where vertex cover algorithms can be applied.

1.7 Outline of the Dissertation

The remaining part of the document is organized as follows,

Chapter 2 describes the research methodology of the dissertation work on Vertex cover problems. It includes the methods and techniques used in the area of vertex cover problem till now and the method used for the comparative analysis in this dissertation work.

Chapter 3presents the overview of the algorithms used in this dissertation work.

Chapter 4 describes the details of implementation of the three algorithms used in this work.

Chapter 5contains the experimental data collected during this dissertation work and analyses the experimental results. The performances of the algorithms over the input datasets are analyzed in this section.

Chapter 6concludes the dissertation work by summarizing the analysis and further recommendation of the research work.

CHAPTER 2
RESEARCH METHODOLOGY

2.1 Literature Review

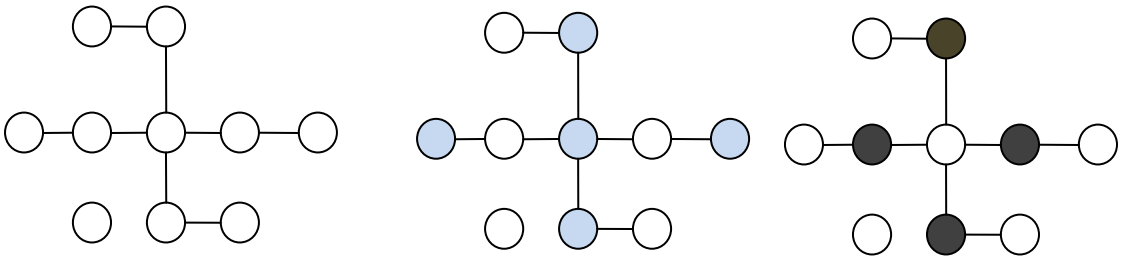
Vertex cover problem fascinates a lot of computer science researchers. A lot of work in the field of vertex cover problem had been done in the past and new research is going on. In 1972, Researcher Karp proved this problem to be NP complete[8]. Even if a problem is NP-complete, there are three ways to get around NP-completeness. First, if the actual inputs are small, an algorithm with exponential running time may be perfectly satisfactory. Second, it may be possible to isolate important special cases that can be solved in polynomial time. Third, we might come up with approaches to find near-optimal solutions in polynomial time (either in the worst case or the expected case). In practice, near optimality is often good enough. An algorithm that returns near-optimal solutions is called an approximation algorithm. Although the vertex cover problem and minimum vertex cover problem is NP complete problems, there had been a lot of research works for finding a optimal or near to optimal solutions using different techniques such as heuristics algorithms , approximation algorithms etc. Many researchers are in favor of approximation algorithms as the size of the vertex-cover returned by these approximation algorithms is guaranteed to be no more than twice the size of an optimal vertex-cover.

APPROX-VERTEX-COVER [3] is a polynomial time 2-approximation algorithm that picks any edge and adds the corresponding vertices to vertex cover set S. The idea of this heuristic is to simply put both vertices into the vertex cover and remove all the edges that are incident to either of the added vertex. This process is repeated for all remaining edges. This simple algorithm guarantees an approximation within a factor of 2 for the vertex cover problem i.e. the cover generated is at most twice the size of the optimum cover. It runs in O (V + E) time and loops until all edges have been removed returning a vertex cover that is twice the optimal cover.

The Maximum Degree Greedy (MDG)[9,10] improves the 2-for-1 heuristic of the approx vertex cover algorithm. MDG is a simple but clever algorithm. At each step, it selects and adds the vertex of highest degree to vertex cover, and then removes it and its all incident edges. This process is repeated until all edges in the graph are covered. Clearly a vertex of

higher degree should be more valuable in forming a small vertex cover since it covers a number of edges.

The Greedy Independent Cover (GIC)[10] is an adaptation of the greedy algorithm for the maximum independent set problem[11]. It is based on the concept that the vertices not in independent set must belongs to vertex cover. In this algorithm, vertex of the minimum degree is selected and all its neighbors are added to the vertex cover, the process continues until we cover all edges. The greedy heuristic cannot always find an optimal solution. The Figure 2.1 shows failure of greedy approach in a simple graph.



(a)A Graph instance

(b) A vertex cover of size 5 obtained by the greedy algorithm

(c) A minimum vertex cover

Figure 2.1: Vertex cover by greedy algorithm

Vertex cover algorithms based on greedy approach has worst performance on complete bipartite graphs and gives a solution of size two times the optimal size.

The Depth First Search (DFS) algorithm has worst-case approximation ratio of 2. This algorithm creates depth-first search spanning tree from the given graph and returns its non-leaf vertices. If G is connected the result is a connected vertex cover. Otherwise the algorithm is executed on each connected component of G (excepted isolated vertices) [10].

The LISTLEFT(LL) algorithm [10] proposed by Avis and Imamura to find vertex cover based on a list heuristic. In this model, an algorithm scans the vertices one by one in a fixed given order (called a list) and takes a decision for each currently scanned vertex (and each decision is definitive). In this algorithm the order of scanning the list (i.e. selection) of the vertices is known in advance and cannot be changed during the process. It works by scanning the vertex list from left to right and the vertex that has at least a right neighbor not in vertex cover is added to vertex cover set. The authors show that the ListLeft Algorithm has

an approximation ratio of$\frac{\sqrt{\Delta}}{2}+\frac{3}{2}$($\Delta$is the maximum degree of the graph) when lists are sorted by decreasing order of their degrees and that any list algorithm cannot have an approximation better than$\frac{\sqrt{\Delta}}{2}$in that case[10].

Another list heuristic algorithm ListRight [12] modifies the LISTLEFT algorithm by changing the order of processing the list, i.e. list are scanned from right to left. The ListRight (LR) algorithm is a better list heuristic than ListLeft. It was proved that for any list L ListRight returns a vertex cover whose size is smaller than or equal to the one constructed byListLeft applied on the same list L. ListRight has a worst-case approximation ratio of Δ, where Δ is the maximum degree in the graph. [10,12].

Aggression and Stein presented a divide and conquer approach. Their idea is to divide the graph into sub graphs and then solving each sub graph. The sub-solutions are then combined to get the solution of the original graph. The division of graph is done on the basis of specific graph structure and unique properties. The included methodologies are degree one and two vertices elimination, triangle elimination, almost bipartite etc. Their approach does not seem to be feasible for practical purpose as the criteria for graph division is vague. Divide and conquer approach is very efficient for providing solutions to other scenarios but it fails to provide a generic solution plus the graph divisions also add to the complexity[13].

Alom [13] proposed a solution to vertex cover problem by introducing an O(|E|) greedy algorithm. This algorithm selects the vertex which has maximum number of edges incident to it. All the edges, that are incident to this vertex, are discarded. If more than one vertex have the same maximum number of edges, this algorithm select the vertex which have at least one edge that is not covered by other vertices. This process is repeated until to cover all vertices[13].

Many of the previous mentioned methods to solve the problem of MVC depend on the degree of the vertex itself. Balaji et al. presented another technique that depends of a value of the support of the vertex. Support of a vertex is the sum of degrees of all vertices adjacent to that vertex. They proposed an algorithm called vertex support algorithm (VSA). They have tested their approach on large number of benchmarks and are optimal in most of the cases and its rum time complexity is O (EV^2) [13,14,15]. They provide empirical results obtained using extensive set benchmark graphs to show the effectiveness of algorithm. They have also compared their approach with other techniques. The algorithm design is little bit complex and

the calculation of support for each and every node in iterative manner adds some complexity. But the quality of results is considerably better than other approaches[13,15].

A modification of the VSA called Modified Vertex Support Algorithm(MVSA) is proposed by Imran and Hasham by modifying the decision of MVC node selection, MVSA select node on the basis of their surroundings [13,14]. Here in MVSA, the selection of the vertex does not depend only on the vertex that have the maximum the support value but it finds all the vertices with minimum support value and then it selects the vertex with minimum support from the list of all neighbors of the selected vertices. MVSA selects vertex after analysis of surrounding support value of all attached vertices to the candidate node. All vertices attached to minimum support value is analyzed first on the basis of their support value and then vertex with maximum support is selected. It is a small modification but experimental results provided in [14]show that MVSA can provide better results in comparison to the original VSA[13,14].

Some other solutions depend on genetic algorithms such as algorithm for heuristic vertex cover (HVX) [16]. Xu and Ma[13,17] presented a solution that uses annealing algorithms to find the minimum vertex cover. In their work they show almost 100% approximation ratio for some benchmarks but they need to apply it on more benchmarks. A new clever intelligent greedy approach is presented by Gajurel and Bielefeld named NOVAC-I [13,18,19]. This approach works on a clever concept raised from the keen observation and analysis of relationship among vertices. The vertices attached to minimum degree nodes are candidate of MVC with high probability. A well modified version of VSA named Advance Vertex Support Algorithm (AVSA) has been proposed in [20]and has showed that if the selection is modified then results vary a lot from the original values. It also concludes that the quality of solution also depends on the data structure used and its manipulation.

The Clever Steady Strategies Algorithm (CSSA) [4] presents a simple and fast polynomial time algorithm. The proposed algorithm consists of three stages which produce optimal or approximate vertex cover for any un weighted and undirected G = (V, E). The CSSA is tested on small as well as on large benchmark instances. The experimental results and comparative analysis show that the CSSA yields better and fast solution than those approximation algorithms found in literature for solving minimum vertex cover problem. A recent paper proposes a new algorithm, New modified vertex support algorithm (NMVSA) [21] which is a modification of already existed algorithm called MVSA that uses the same

principle of selecting candidate from the neighborhood of the vertex with a modification in the selection procedure. A comparative study is conducted between the NMVSA and MVSA which shows that the proposed algorithm NMVSA provides better or equal results in the most cases of the underlying data sets which leads to a better average approximation ratio of NMVSA. NMVSA inherits the simplicity of the original algorithm[21].

In a research paper[22],Chen, J., Kou, L., and Cui, X. presents a new approach, an approximation algorithm is obtained for the minimum vertex cover problem that is based on Dijkstra algorithm. In the process of getting a vertex cover, the maximum value of shortest paths is considered as a standard, and some other criteria are defined The time complexity of the Algorithm is $O(n^3)$,where n is the number of vertices in a graph.

In [23] authors have presented a new extra fast approximation algorithm for solving MVC generally in all graphs. The proposed algorithm is named as degree contribution algorithm (DCA), and it introduces a new data structure called 'degree contribution' (DC) for each node which is the sum of degree of that node and total number of nodes with that degree in graph.This data structure for graphs takes account of whole graph for each node contribution value. All decisions regarding vertices are made on the basis of the proposed data structure. Effectiveness of DCA is shown by applying it to best available benchmarks and after large number of experiments worst approximation ratio recorded was 1.041 and an average approximation ratio was 1.005. These results show that algorithm can perform well in solving graphs faster as compared to other algorithms

In [24] an evolutionary approach to solving the generalized vertex cover problem (GVCP) is presented. Binary representation and standard genetic operators are used along with the appropriate objective function. The experiments were carried out on randomly generated instances with up to 500 vertices and 100 000 edges.

`A hybrid approach to approximating the minimum vertex cover based on a combination of a steady state genetic algorithm with a greedy heuristic. First the genetic algorithm produces a set of nodes, which is then reduced by greedy heuristics. They tested their approach against ant colony optimization (ACO) and showed that their approach not only works faster than ACO, but is also efficient in producing final output[25].

The authors of [25] propose a new approximation algorithm for the minimum vertex cover problem called vertex cover using a maximum independent set (VCUMI). This algorithm works by removing the nodes of a maximum independent set until the graph is an approximate solution of MVC. Based on its empirical results, it states that VCUMI

outperforms all competing algorithms presented in the literature. Based on all the benchmarks[26] used, VCUMI achieved the worst case error ratio of 1.033, while VSA, MDG and NOVAC-1 gave the worst error ratios of 1.583, 1.107 and 1.04, respectively.

2.2 Concepts of Approximation algorithm

Many computational problems of practical significance are NP-complete. Some of themare so important that they cannot be left unsolved for the fact that their optimal solution is impossible to be determined in a reasonable time frame. If a problem is NP-complete, it is unlikely to find a polynomial-time algorithm for solving it exactly, but as previously mentioned there are different methods to tackle with them. In practice, near-optimality is often good enough. *An algorithm that returns near-optimal solutions is called an approximation algorithm*[3].

Depending on the problem, an optimal solution may be defined as one with maximum possible cost or one with minimum possible cost; that is the problem may be either maximization or a minimization problem. *An algorithm for a problem has an approximation ratio of ρ(n) if, for any input of size n, the cost c of the solution produced by the algorithm is within afactor of ρ(n) of the cost c* of an optimal solution[3]*:

$$\text{Max}(c/c^*, c^*/c) \leq \rho(n)$$

*An algorithm that achieves an approximation ratio of ρ(n) is called a ρ(n)-approximation algorithm.*The definitions of approximation ratio and of ρ (n)-approximation algorithm apply for both minimization and maximization problems[3].

2.3 Research Methodology

2.3.1 Input Selection

All the graph data that is used for the testing and analysis of the approximation algorithms are obtained from the different benchmark graph datasets available over the internet[26]. The benchmark graph datasets have predetermined optimal solution [26].

2.3.2 Input File Format

The inputs provided in the above files are in the formats as given below:

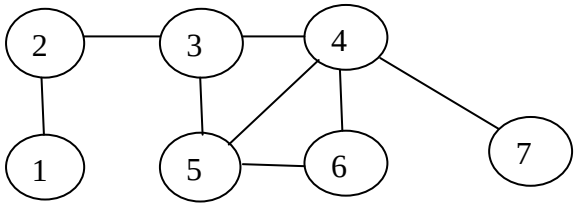


Figure 2.2: Input Graph

Sample graph data file format for above graph in Figure2.2 :

p edge 7 8

e 1 2

e 2 3

e 3 4

e 4 5

e 4 6

e 4 7

e 5 6

e 5 3

Here 'p' is the number of vertices in the graph, edge is the number of edges in the graph and 'e 1 2' represents an edge between vertex 1 and vertex 2 and so on.

2.3.3 Output Format

The output file containing vertex cover information is formatted as given below:

size 3 2 4 5

Here the 'size' and the first number following the word 'size' is the size of vertex cover found by the algorithm. The other numbers following the first number, represents the vertex included in the minimum vertex cover set.

There are other three output files for each algorithm generated as outputs that contains data and information about the input and output of the algorithms implemented, formatted as given below:

(**Algorithm name) results:**

input graph file |	**No. of vertices |**	**MVC Size |**	**step count |**
frb30-15-1.mis	**450**	**427**	**427**

2.3.4 Data Collection ,testing and comparison

The benchmark graph dataset is passed as the input in the different algorithms implemented and three separate output files are generated for each of the input graph data files and each algorithm. Each of these output files contains the size of vertex cover and the vertex cover set found by the algorithm. This output file containing vertex cover set is tested for the verification that the output file contains a valid vertex cover set for the input graph.

The aim of this Dissertation is to compare the performance of three algorithms CSSA, NOVCA-I, and NMVSA. The size of vertex cover and vertex cover set is determined by executing the implemented programs for the algorithms for each input graph and the required data is collected in separate text file as output of the programs. The approximation ratio of the algorithms is calculated as:

Approximation ratio= MVC size determined by algorithm / Optimal MVC size.

The results of the experimentation are tabulated in tables.

CHAPTER 3
THE ALGORITHMS

3.1 Nearly Optimal Vertex Cover (NOVAC-I)

Gajurel and Bielfeld presented an extremely fast polynomial time algorithm, the Near Optimal Vertex Cover Algorithm (NOVCA)-I[18,19] that produces an optimal or near optimal vertex cover for any known undirected graph G (V, E). NOVCA-I is motivated by the fact that a vertex cover candidates are those that are adjacent to minimum degree vertex. So that when the adjacent vertices of minimum degree are added to vertex cover list, the degree of the minimum degree vertex will be forcibly rendered to zero without choosing it. This fact has been reinforced during tie when the vertex with neighbors having maximum degrees is preferred over other minimum vertices. The complexity of NOVCA-I is O(E (V + $\log^2$V)); with V = n, the complexity becomes O(n^2 (n + $\log^2$n)) which is polynomial. NOVCA-I constructs the vertex cover by repeatedly adding, at each step, all vertices adjacent to the vertex of minimal degree; in the case of a tie, it selects the one having the maximum sum of degrees of its neighbors. The vertices are chosen in increasing order of their degrees i.e. the adjacent vertices of minimum degree vertex are included in vertex cover VC. The implementation forcibly renders the degree of low degree vertices to zero without choosing them. Run time complexity using their approach is in O (n^2 log n) where 'n' is total number of vertices [18].

3.2 Clever Steady Strategy Algorithm

The algorithm Clever steady strategy algorithm (CSSA) [4] selects a minimum degree vertex from a list of all vertices adjacent to minimum degree vertices. The algorithm primarily consists of three stages which produce optimal or approximate vertex cover for any unweighted and undirected G = (V, E). In the first step the degree of each node of the given graph is calculated. Then the minimum degree node(s) is searched and the adjacent nodes of minimum degree nodes are determined. In the third stage the minimum degree node in all adjacent nodes of minimum degree is searched out and is selected as a candidate for MVC and all its edges are deleted. These three steps are executed repeatedly until no edge remains in the graph.

The complexity of the CSSA is O(n^2 log n), where n is number of vertices.

3.3 New Modified Vertex Support Algorithm

The selection of vertices in New Modified Vertex Support Algorithm (NMVSA) [21] algorithm relies on the degree of the neighborhood vertices and support of a vertex. The definition of neighborhood, degree, and support of vertex is as:

- ***Neighborhood of a vertex****: Let G be an undirected graph G (V, E) where V is set of vertices and E is set of Edges.* $|E|=m$, $|V|=n$. *For each* $v \in V$ *the neighborhood of v,* $N(v) = \{u \in V \mid u \text{ is adjacent to } v.\}$
- ***Degree of a vertex****: The degree of the vertex d (v) is the number of adjacent neighbors for vertex* $v \in V$.
- ***The support of a vertex****: support of a vertex s (v) is the sum of degrees of all neighbors of v.*
- ***Vertex Cover****: Vertex cover* $c=\{x \in V \mid x=u \text{ or } v \text{ if } (v,u) \text{ is an edge } e \in E\}$

The idea of selection in the algorithm NMVSA depends on the fact that the candidates of vertex cover are adjacent to the vertices with minimum degrees. On each iteration, NMVSA adds a vertex from the support list with the maximum degree to the vertex cover and delete all edges connected to this vertex. The process continues until no more edges still in E [21].

The selections of the vertices that will be part of the vertex cover rely on the vertices that are adjacent to minimum degree vertices and their values of support. Value of support is a value represents the sum of the degrees of the neighbors of the vertices. The intuition behind the algorithm is to select the vertices that connect as much as possible from the vertices that are located on the edges of the graph. The complexity of the NMVSA is $O(mn^2)$, where m is number of edges and n is number of vertices.

CHAPTER 4

IMPLEMENTATION

4.1 Tools Used in program development

All the three algorithms studied in this dissertation work, are separately implemented using C++ programming language. The standard template library (STL) of C++ programming language are heavily used in the implementation of algorithms as these library provides number of built in methods that are very useful while developing the programs.

4.2 Data Structure Used

Different data structures such as array, vector, lists that are readily available in Standard Template Library of C++ are used while implementing the algorithms for this dissertation work. The graph data structure is implemented using adjacency list concept, A graph class is implemented using the combination of these data structures as shown below:

```
class graph
{
   int V;                    // No. of vertices
   list<int> *adj;           // Pointer to an array containing adjacency lists
   public:
        vector<int>deg;      //vector to store the degree of vertices
        vector<int>spt;      // vector to store the support of vertex
   graph(int V);                    // Constructor
 void addEdge(ifstream& fin);       // method to add an edge to graph reading from
                                    //input file
 int mindeg(vector<int>  );         //method to find minimum degree in the graph
 int minspt(vector<int>  );         //method to find minimum support in the graph
vector<int>printVertexCover(graph g);  //method to find vertex cover
};
```

4.3 Experimental Setup

The system used during implementation has Windows 8 operating system with 4 GB RAM and Intel Core I3 processor. The algorithms are implemented by creating a console based multifile project by using Dev-C++ Version 5.6.3 IDE. The multifile project contains 4 separate files; 3 files for the three algorithms and 1 file is for the main process.

4.4 Pseudocode of algorithms

4.4.1 NOVCA-I Algorithm

Algorithm 4.1

step 1. Initialize VC to '0'; //VC means vertex cover set.

step 2. Calculate degree for each vertex in the graph.

step 3. Calculate the sum of degrees of adjacent vertices for each vertex.

step 4. Get a vertex with minimum degree, and if there are more than one vertex with minimum degree the select the one having maximum value of sum of degrees of adjacent vertices.

step 5. Add all adjacent vertices of that vertex to vertex cover and delete all its edges.

step 6. Update value of degrees and sum of degrees of adjacent vertices for each vertex.

step 7. Repeat through steps 4 to 7 while all Edges are processed.

step 8. Return MVC

4.4.2 CSSA Algorithm

Algorithm 4.2

step 1. Calculate degree of all vertices in the graph G.

step 2. Find out the minimum degree vertex/vertices and create a list L of their adjacent
vertex/vertices.

step 3. Find minimum degree vertex/vertices (V_m) in the list L.

step 4. Add V_m to minimum vertex cover (MVC) set and delete all edges adjacent to it.

step 5. Repeat steps 1 to 4 until edge exists.

step 6. Return MVC.

4.4.3 NMVSA Algorithm

Algorithm 4.3

step 1. Calculate degree for each vertex that is not included in vertex cover set VC.

step 2. Calculate support for each vertex

step 3. Find minimum support value(minsup)

step 4. Create list L of vertices V_i with same minimium support value (minsup)

step 5. Find and create list(H) of neighbours of minimum support vertices in L

step 6. Select vertex(vertices) C with maximum support value from H and add them to vertex cover set

step 7. Delete all adjacent edges of C

step 8. Repeat step 1 to 7 until edges exists in graph.

step 9. Return MVC

4.4.4 Illustration of algorithms

- **Illustration of NOVCA**

Let us consider the given graph as in Figure 4.1.

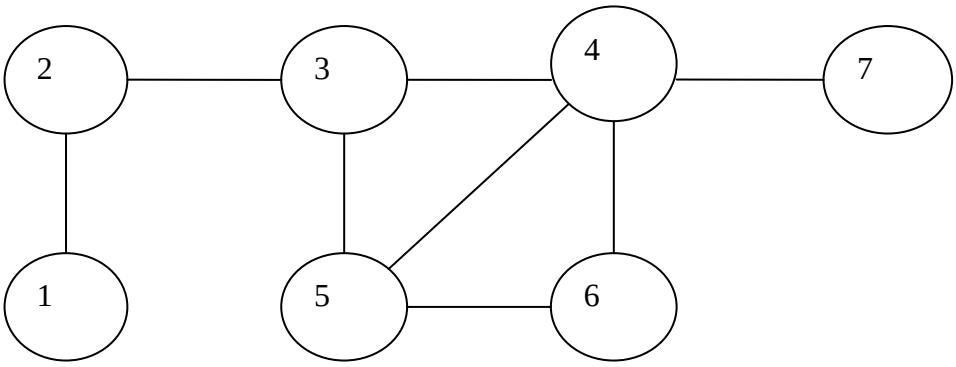


Figure 4.1: A graph with 7 vertices and 8 edges

- Edge ={(1,2),(2,3).(3,4),(3,5),(4,5),(4,6), (4,7), (5,6)}, vc=0,VC={ }
- Degree and sum of degrees of adjacent vertices for each vertex are calculated as in Table 4.1.

Table 4.1: Degree and sum of degree of adjacent for each vertex

Vertex(v)	1	2	3	4	5	6	7
Degree(v)	1	2	3	4	3	2	1
Sum_deg_adj(v)	2	4	9	9	9	7	4

- minimum degree vertices: 1,7
  since sum_adj_deg(7)>sum_adj_deg(1)
  selected vertex = 7
  adjacent (7)= 4
- Add vertex 4 to vertex cover and delete all its edges ; VC={4},vc=1;
- Value of degrees and sum of degrees of adjacent vertices for each vertex are updated as in Table 4.2.

Table 4.2: Updated Degree and sum of degree of adjacent for each vertex(I)

Vertex(v)	1	2	3	~~4~~	5	6	7
Degree(v)	1	2	2	0	2	1	0
Sum_deg_adj(v)	2	3	4	0	3	2	0

- Edge ={(1,2),(2,3),(3,5),(5,6)},
- minimum degree vertices: 1,6
  since sum_adj_deg(6)=sum_adj_deg(1)
  selected vertex = 6
  adjacent (6)=5
- Add vertex 5 to vertex cover and delete all its edges
- VC={4,5},vc=2;
- value of degrees and sum of degrees of adjacent vertices for each vertex are Updated as in Table 4.3.

Table 4.3: Updated Degree and sum of degree of adjacent for each vertex(II)

Vertex(v)	1	2	3	~~4~~	5	6	7
Degree(v)	1	2	1	0	0	0	0
Sum_deg_adj(v)	2	2	2	0	0	0	0

- Edge ={(1,2),(2,3)}
- minimum degree vertices: 1,3
- since sum_adj_deg(1)=sum_adj_deg(3)
- selected vertex = 1
- adjacent(1)=2
- Add vertex 2 to the VC and delete all its edges
- VC={4,5,2}, vc=3;
- Edge list={}
- Stop;

Final Minimum Vertex cover VC={4,5,2}

➢ **Illustration of NMVSA**

Let us consider the given graph in Figure 4.1

- Edge ={(1,2),(2,3),(3,4),(3,5),(4,5),(4,6), (4,7), (5,6)}, vc=0,VC={ }
- degree and support for each vertex in the graph are calculated as in Table 4.4.

Table 4.4: Degree and support for each vertex

Vertex(v)	1	2	3	4	5	6	7
Degree(v)	1	2	3	4	3	2	1
support(v)	2	4	9	9	9	7	4

- minimum support=2
- list of minimum support vertices, L:{ 1}
- list of neighbor of L; H ={2}
- max support of H =4
- max support vertex/vertices in H={2}
- Add vertex 2 to vertex cover and delete all its edges; VC={2},vc=1;
- value of degrees and support for each vertex are updated as in Table 4.5.

Table 4.5: Updated Degree and Support for each vertex(I)

Vertex(v)	1	2	3	4	5	6	7
Degree(v)	0	0	2	4	3	2	1
support(v)	0	0	7	9	9	7	4

- Edge ={((3,4),(3,5),(4,5),(4,6), (4,7), (5,6)},
  minimum support=4
- list of minimum support vertices, L:{ 7}
- list of neighbor of L; H ={4}
- max support of H =9
- max support vertex/vertices in H={4}
- Add vertex 4 to vertex cover and delete all its edges
- VC={2,4}; vc=2;
- value of degrees and support for each vertex are updated as in Table 4.6.

Table 4.6: Updated Degree and Support for each vertex(II)

Vertex(v)	1	2	3	4	5	6	7
Degree(v)	0	0	1	0	2	1	0
support(v)	0	0	2	0	2	2	0

- Edge={(3,5),(5,6)}
- minimum support=2
    - list of minimum support vertices, L:{ 3,5,6}
    - list of neighbor of L; H ={3,5,6,5}
    - max support of H =2
    - max support vertex/vertices in H={3,5,6,5}
- Add vertex 5 to the VC and delete all its edges
- VC={2,4,5 }, vc=3;
- Edge list={}
- Stop;
  Final Minimum Vertex cover VC={2,4,5}

➢ **Illustration of CSSA**

Let us consider the given graph

- Edge ={(1,2),(2,3),(3,4),(3,5),(4,5),(4,6), (4,7), (5,6)}, vc=0,VC={ }
- degree for each vertex in the graph are calculated as in Table 4.7.

Table 4.7: Degree of each vertex

Vertex(v)	1	2	3	4	5	6	7
Degree(v)	1	2	3	4	3	2	1

- minimum degree vertices: 1,7
- Adj(1,7)={2,4}
- minimum degree in adj(1,7) =2
- Add vertex 2 to vertex cover and delete all its edges

  VC={2},vc=1;
- value of degrees for each vertex are updated as in Table 4.8.

Table 4.8: Updated Degree of each vertex(I)

Vertex(v)	1	2	3	4	5	6	7
Degree(v)	0	0	2	4	3	2	1

- Edge ={((3,4),(3,5),(4,5),(4,6), (4,7), (5,6)},
- minimum degree vertices: 7

  adjacent (7)=4

  minimum degree of adjacent(7)=4
- Add vertex 4 to vertex cover and delete all its edges
- VC={2,4}; vc=2;
- value of degrees for each vertex are updated as in Table 4.9.

Table 4.9: Updated Degree of each vertex(II)

Vertex(v)	1	2	3	4	5	6	7
Degree(v)	0	0	1	0	2	1	0

- Edge={(3,5),(5,6)}
- Minimum degree vertex=3,6
- adjacent(3,6)=5
- Add vertex 5 to the VC and delete all its edges

- VC={2,4,5 }, vc=3;
- Edge list={}
- Stop;

Final Minimum Vertex cover VC={2,4,5 }

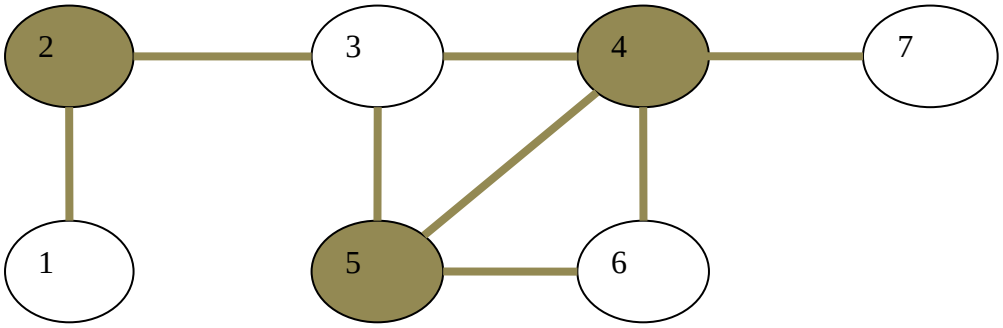


Figure 4.2: Final vertex cover for graph in Figure 4.1

CHAPTER 5

RESULT ANALYSIS

5.1 Approximation Ratio

The first Performance metrics used in this comparative study is Approximation ratio. The following Table 5.1(I) to Table 5.1(IV) lists the Approximation ratio of CSSA, NOVCA and NMVSA algorithms along with the number of vertices in the input graph, Size of Optimal MVC for the graph, the size of MVC for the input graph determined by the algorithms.

Table 5.1:(I) Approximation ratio of NOVCA, NMVSA and CSSA

Input graph file	No. of Vertices	Optimal MVC	MVC Size			Approximation ratio		
			CSSA	NOVCA	NMVSA	NMVSA	NOVCA	CSAA
frb30-15-1	450	420	427	424	425	1.012	1.010	1.017
frb30-15-2	450	420	426	426	426	1.014	1.014	1.014
frb30-15-3	450	420	425	425	425	1.012	1.012	1.012
frb30-15-4	450	420	426	426	424	1.010	1.014	1.014
frb30-15-5	450	420	425	424	425	1.012	1.010	1.012
frb35-17-1	595	560	566	566	567	1.013	1.011	1.011
frb35-17-2	595	560	565	565	567	1.013	1.009	1.009
frb35-17-3	595	560	565	565	565	1.009	1.009	1.009
frb45-21-1	945	900	908	906	907	1.008	1.007	1.009
frb45-21-2	945	900	908	907	909	1.010	1.008	1.009
frb45-21-3	945	900	909	909	910	1.011	1.010	1.010
frb45-21-4	945	900	906	907	908	1.009	1.008	1.007
frb45-21-5	945	900	907	908	908	1.009	1.009	1.008
frb50-23-2	1150	1100	1112	1108	1109	1.008	1.007	1.011
frb50-23-3	1150	1100	1107	1107	1107	1.006	1.006	1.006
frb50-23-4	1150	1100	1107	1107	1110	1.009	1.006	1.006
frb50-23-5	1150	1100	1111	1109	1110	1.009	1.008	1.010

Table 5.1: (II) Approximation ratio of NOVCA, NMVSA and CSSA

Input graph file	No. of Vertices	Optimal MVC	MVC Size			Approximation ratio		
			CSSA	NOVCA	NMVSA	NMVSA	NOVCA	CSAA
graph50-01	50	30	30	30	30	1.000	1.000	1.000
graph50-02	50	30	30	30	30	1.000	1.000	1.000
graph50-03	50	30	30	30	30	1.000	1.000	1.000
graph50-04	50	40	40	40	40	1.000	1.000	1.000
graph50-05	50	27	27	27	27	1.000	1.000	1.000
graph50-06	50	38	38	38	38	1.000	1.000	1.000
graph50-07	50	35	35	35	35	1.000	1.000	1.000
graph50-08	50	29	29	29	29	1.000	1.000	1.000
graph50-09	50	40	40	40	40	1.000	1.000	1.000
graph50-10	50	35	35	35	35	1.000	1.000	1.000
graph100-01	100	60	60	60	60	1.000	1.000	1.000
graph100-02	100	65	65	65	65	1.000	1.000	1.000
graph100-03	100	75	75	75	75	1.000	1.000	1.000
graph100-04	100	60	60	60	60	1.000	1.000	1.000
graph100-05	100	60	60	60	60	1.000	1.000	1.000
graph100-06	100	80	80	80	80	1.000	1.000	1.000
graph100-07	100	65	65	65	65	1.000	1.000	1.000
graph100-08	100	75	75	75	75	1.000	1.000	1.000
graph100-09	100	85	85	85	85	1.000	1.000	1.000
graph100-10	100	70	70	70	70	1.000	1.000	1.000
graph250-05	250	200	200	200	200	1.000	1.000	1.000
graph500-01	500	350	350	350	350	1.000	1.000	1.000
graph500-02	500	400	400	400	400	1.000	1.000	1.000
graph500-03	500	375	375	375	375	1.000	1.000	1.000
graph500-04	500	300	300	300	300	1.000	1.000	1.000
graph500-05	500	290	290	290	290	1.000	1.000	1.000

Table 5.1: (III) Approximation ratio of NOVCA, NMVSA and CSSA

Input graph file	No. of Vertices	Optimal MVC	MVC Size			Approximation ratio		
			CSSA	NOVCA	NMVSA	NMVSA	NOVCA	CSAA
hamming10-2	1024	512	512	512	512	1.000	1.000	1.000
hamming10-4	1024	984	992	988	988	1.004	1.004	1.008
hamming6-2	64	32	32	32	32	1.000	1.000	1.000
hamming8-2	256	128	128	128	128	1.000	1.000	1.000
hamming8-4	256	240	240	240	240	1.000	1.000	1.000
C125.9	125	91	94	92	94	1.033	1.011	1.033
C250.9	250	206	208	211	211	1.024	1.024	1.010
C500.9	500	443	453	451	452	1.020	1.018	1.023
DSJC500.5	500	<=487	489	489	489	1.004	1.004	1.004
c-fat500-10	500	374	374	374	374	1.000	1.000	1.000
c-fat500-5	500	436	436	436	436	1.000	1.000	1.000
gen400_p0_9_65	400	335	354	355	354	1.057	1.060	1.057
gen400_p0_9_75	400	325	353	353	356	1.095	1.086	1.086
johnson32-2-4	496	480	480	480	480	1.000	1.000	1.000
johnson8-2-4	28	24	24	24	24	1.000	1.000	1.000
keller4	171	160	160	164	163	1.019	1.025	1.000
keller5	776	749	754	761	754	1.007	1.016	1.007

Table 5.1: (IV) Approximation ratio of NOVCA, NMVSA and CSSA

Input graph file	No. of Vertices	Optimal MVC	MVC Size			Approximation ratio		
			CSSA	NOVCA	NMVSA	NMVSA	NOVCA	CSAA
MANN_a81	3321	2221	2225	2225	2409	1.085	1.002	1.002
MANN_a45	1035	690	693	693	739	1.071	1.004	1.004
p_hat300-2	300	275	275	275	277	1.007	1.000	1.000
p_hat300-3	300	264	266	266	268	1.015	1.008	1.008
p_hat500-1	500	491	492	492	492	1.002	1.002	1.002
p_hat500-2	500	464	467	466	473	1.019	1.004	1.006
p_hat500-3	500	450	455	454	457	1.016	1.009	1.011
p_hat700-3	700	638	642	641	643	1.008	1.005	1.006
sanr400_0_7	400	379	382	382	381	1.005	1.008	1.008
san400_0_5_1	400	387	392	391	391	1.010	1.010	1.013
san400_0_7_1	400	360	379	379	378	1.050	1.053	1.053
san400_0_7_2	400	370	382	383	382	1.032	1.035	1.032
san400_0_7_3	400	378	385	384	385	1.019	1.016	1.019
san400_0_9_1	400	300	326	318	317	1.057	1.060	1.087
sanr400_0_5	400	387	389	388	389	1.005	1.003	1.005
frb100-40	4000	3900	3919	3917	3920	1.005	1.004	1.005

Table 5.2: Minimum, Average and Maximum Approximation ratio of NOVCA, NMVSA and CSSA

Approximation ratio	NMVSA	NOVCA	CSSA
Average approximation ratio	1.012	1.008	1.009
Max approximation ratio	1.095	1.086	1.087
Min approximation ratio	1	1	1

5.2 Step Count

Step count is second performance metrics used in this dissertation work that counts the number of vertices that the particular algorithm requires to process a candidate vertex of the vertex cover set. The following Table 5.3(I) to Table 5.3(V) lists the step counts of the different algorithms.

Table 5.3: (I) Step count of CSSA, NOVCA and NMVSA

Input graph file	No. of Vertices	step count		
		CSSA	**NOVCA**	**NMVSA**
frb30-15-1	450	427	25	425
frb30-15-2	450	426	24	426
frb30-15-3	450	425	25	425
frb30-15-4	450	426	24	424
frb30-15-5	450	425	25	425
frb35-17-1	595	566	29	567
frb35-17-2	595	565	30	567
frb35-17-3	595	565	30	565
frb45-21-1	945	908	39	907
frb45-21-2	945	908	38	909
frb45-21-3	945	909	36	910
frb45-21-4	945	906	37	908
frb45-21-5	945	907	37	908
frb50-23-2	1150	1112	41	1109
frb50-23-3	1150	1107	43	1107
frb50-23-4	1150	1107	43	1110
frb50-23-5	1150	1111	41	1110

Table 5.3: (II) Step count of CSSA, NOVCA and NMVSA

Input graph file	No. of Vertices	Step count		
		CSSA	NOVCA	NMVSA
graph50-01	50	30	9	30
graph50-02	50	30	10	30
graph50-03	50	30	6	30
graph50-04	50	40	9	40
graph50-05	50	27	9	27
graph50-06	50	38	6	38
graph50-07	50	35	6	35
graph50-08	50	29	7	29
graph50-09	50	40	4	40
graph50-10	50	35	8	35
graph100-01	100	60	6	60
graph100-02	100	65	10	65
graph100-03	100	75	7	75
graph100-04	100	60	4	60
graph100-05	100	60	14	60
graph100-06	100	80	6	80
graph100-07	100	65	4	65
graph100-08	100	75	6	75
graph100-09	100	85	5	85
graph100-10	100	70	4	70
graph250-05	250	200	14	200
graph500-01	500	350	17	350
graph500-02	500	400	8	400
graph500-03	500	375	8	375
graph500-04	500	300	5	300
graph500-05	500	290	6	290

Table 5.3: (III) Step count of CSSA, NOVCA and NMVSA

Input graph file	No. of Vertices	Step count		
		CSSA	NOVCA	NMVSA
hamming10-2	1024	512	256	512
hamming10-4	1024	992	34	988
hamming6-2	64	32	16	32
hamming8-2	256	128	64	128
hamming8-4	256	240	12	240
C125.9	125	94	32	94
C250.9	250	208	39	211
C500.9	500	453	49	452
DSJC500.5	500	489	11	489
c-fat500-10	500	374	2	374
c-fat500-5	500	436	2	436
gen400_p0_9_65	400	354	45	354
gen400_p0_9_75	400	353	46	356
johnson32-2-4	496	480	15	480
johnson8-2-4	28	24	3	24
keller4	171	160	7	163
keller5	776	754	15	754

Table 5.3: (IV) Step count of CSSA, NOVCA and NMVSA

Input graph file	No. of Vertices	Step count		
		CSSA	NOVCA	NMVSA
MANN_a81	3321	2225	1096	2409
MANN_a45	1035	693	342	739
p_hat300-2	300	275	23	277
p_hat300-3	300	266	33	268
p_hat500-1	500	492	8	492
p_hat500-2	500	467	31	473
p_hat500-3	500	455	45	457
p_hat700-3	700	642	58	643
sanr400_0_7	400	382	17	381
san400_0_5_1	400	392	8	391
san400_0_7_1	400	379	21	378
san400_0_7_2	400	382	17	382
san400_0_7_3	400	385	15	385
san400_0_9_1	400	326	61	317
sanr400_0_5	400	389	11	389
frb100-40.mis	4000	3919	83	3920

5.3 Analysis and Result

The performance of the approximation algorithms for MVC is measured in terms of following two factors:

5.3.1 Approximation Ratio

The approximation ratio is the ratio of the size of MVC of given graph in solution and the optimal size of MVC of the given graph. An approximation algorithm with approximation ratio 1 produces an optimal solution, and an approximation algorithm with a large approximation ratio may return a solution that is much worse than optimal. In the above table, the approximation ratio for each input graph is calculated separately for each algorithm after all the data are collected.

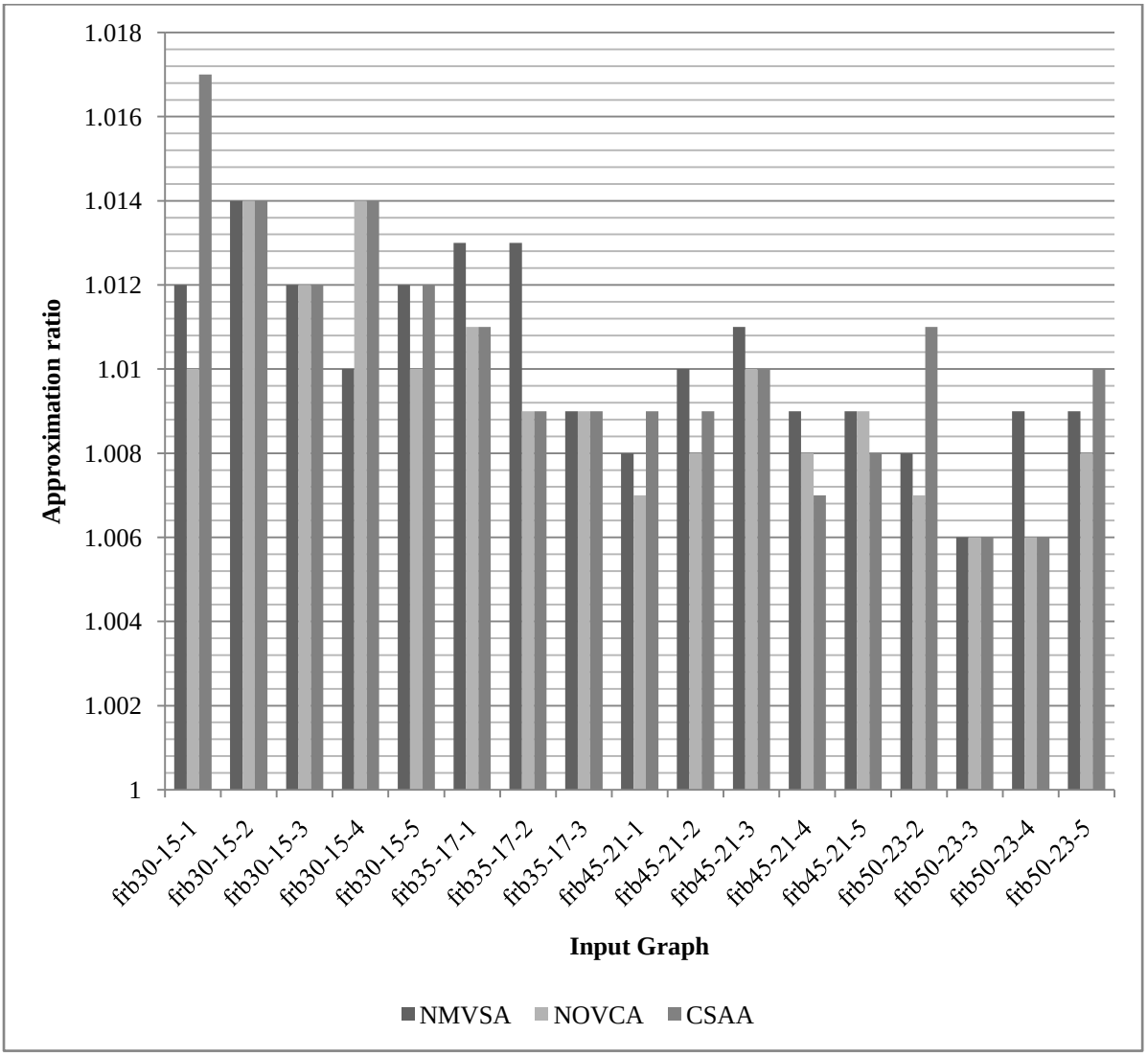


Figure 5.1: (I)Approximation ratio of NOVCA, NMVSA and CSSA

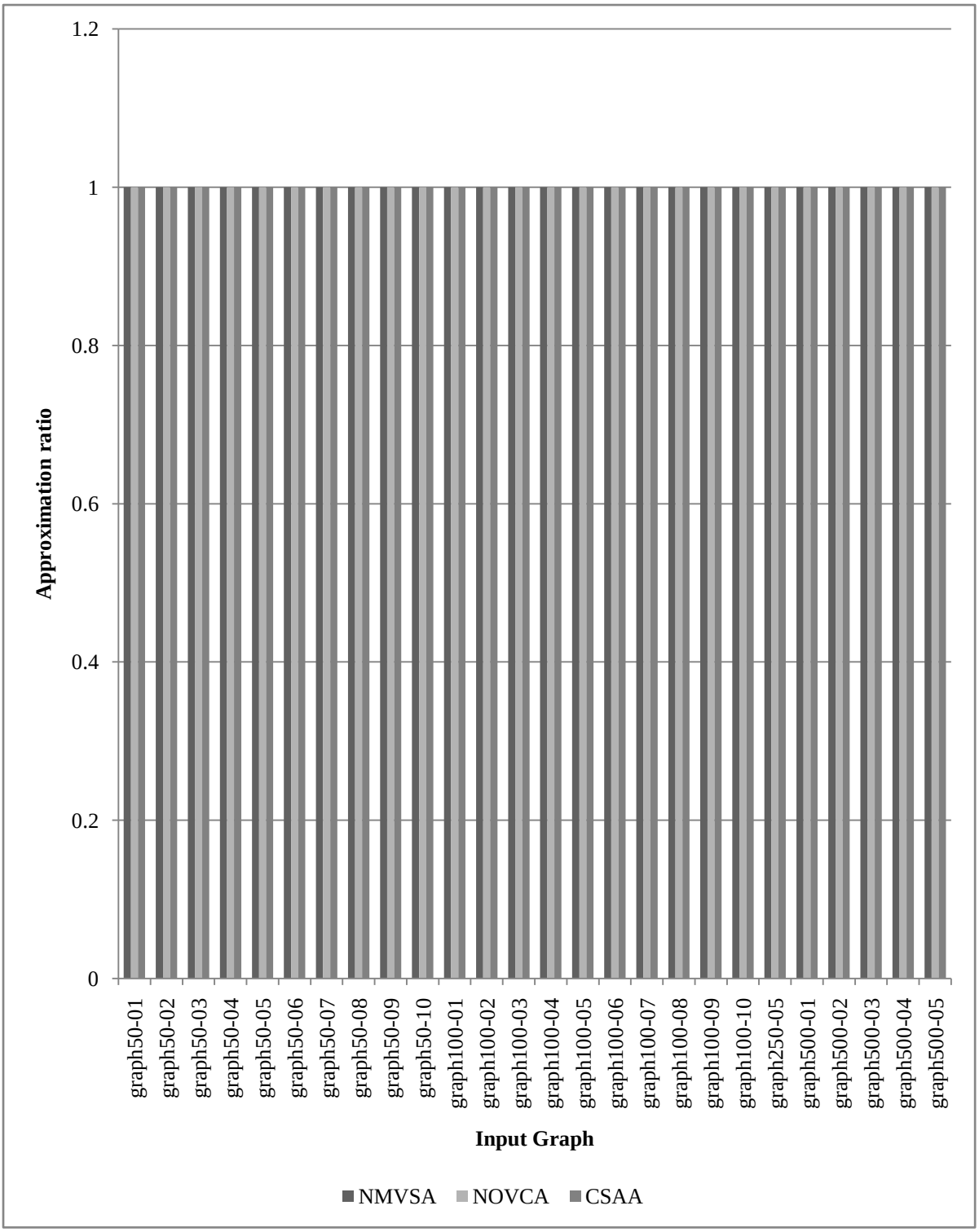


Figure 5.1(II) Approximation ratio of NOVCA, NMVSA and CSSA

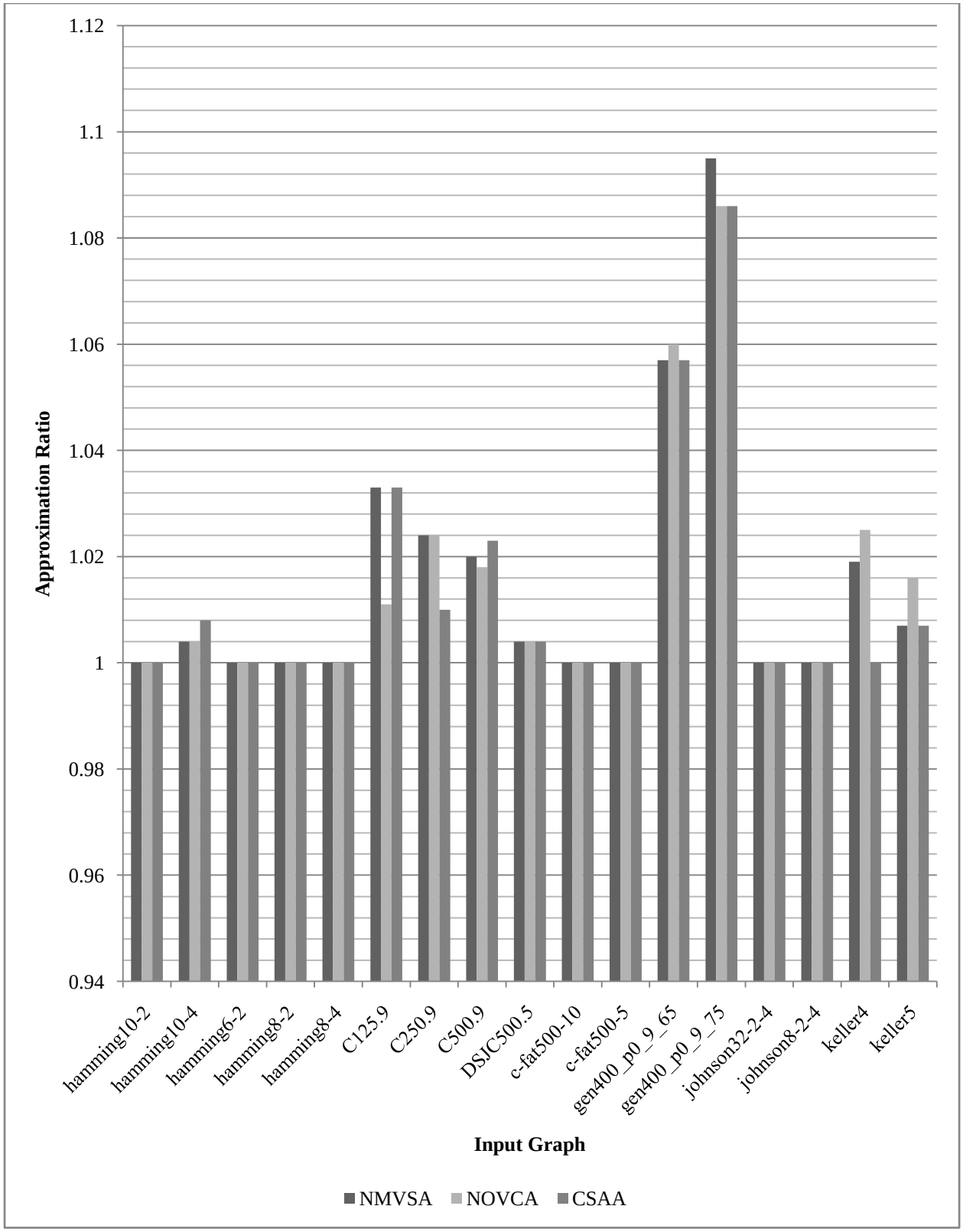


Figure 5.1 :(III) Approximation ratio of NOVCA, NMVSA and CSSA

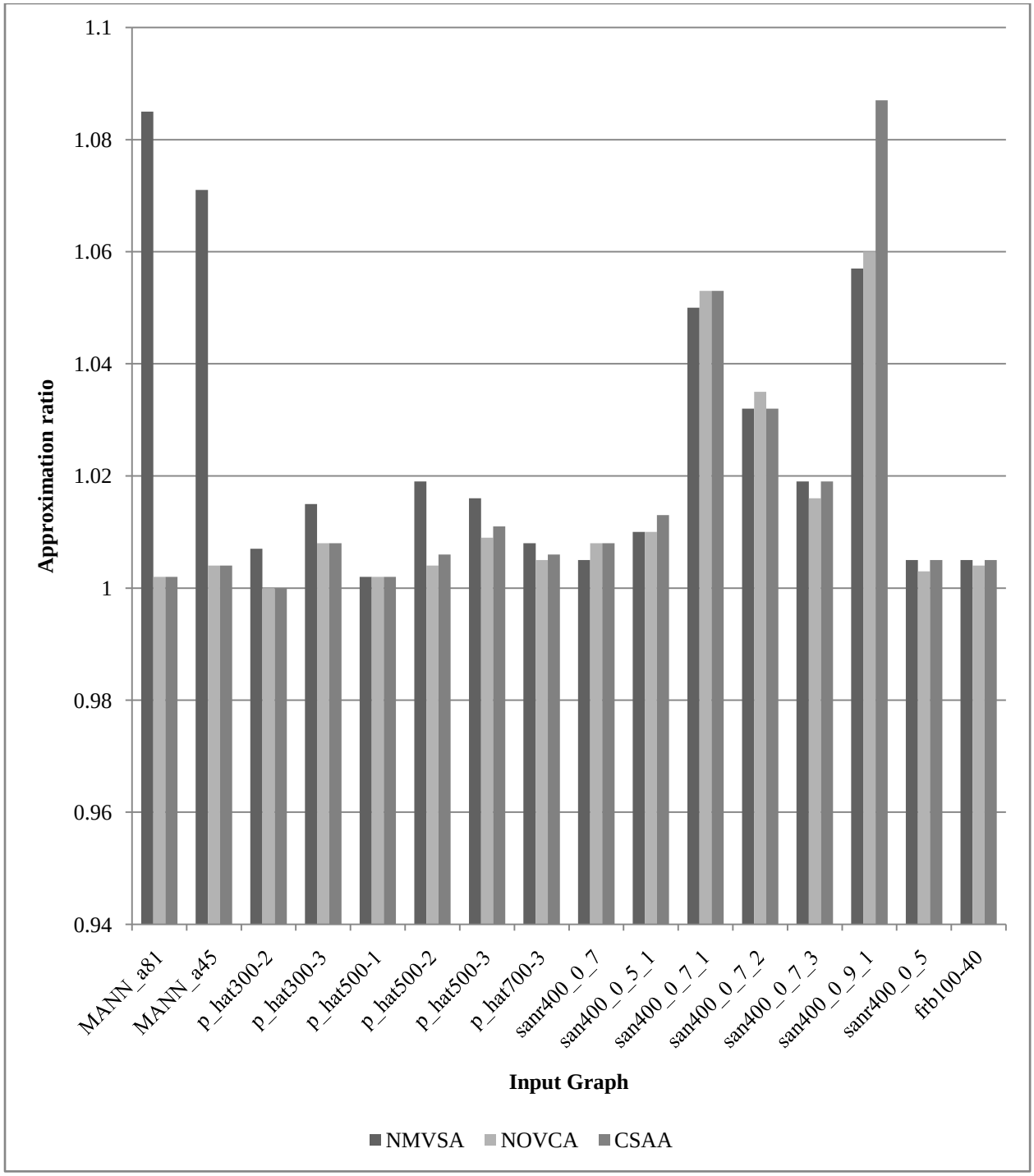


Figure 5.1: (IV)Approximation ratio of NOVCA, NMVSA and CSSA

From the above tabulated information in Table 5.1(I) to Table 5.1(IV) and Figure 5.1(I)to Figure 5.1(IV) ,it is observed that MVC size determined by all the algorithms NMVSA, NOVCA and CSSA has nominal differences (except in some cases). The approximation ratios of all the three algorithms do not differ significantly. The minimum approximation ratio of all the three algorithms is one, the maximum approximation ratios are

also very close to one and they do not deviate much. The average approximation ratio of NOVCA-I is 1.008 which less than that of CSSA and NMVSA, 1.009 and 1.012respectively. It is clearly seen that NOVCA-I has better or same performance as the other two NMVSA, and CSSA algorithm in terms of approximation ratio. As the phrase, the lesser the approximation ratio the better the algorithm is, suggests that the NOVCA-I algorithm is better than the other two algorithms. It is very close to or equal to 1 for the given benchmark graph datasets. A solution having 1 approximation ratio is an optimal solution and solution with approximation ratio very close to 1 can be considered as sub optimal solution and can be used as optimal solution.

5.3.2 Step Count

It represents a number that counts the number of vertices processed by the algorithm to find a valid vertex cover of the given graph. This measure serves in comparing the performance of algorithms in terms of the number of vertices to be processed for finding vertex cover. The lesser step count will indicate the better performance of algorithm. This feature is implemented using a integer variable 'step_count' in the program. This variable is initialized to zero when algorithm starts and gets incremented each time a vertex is processed for finding candidates of minimum vertex cover. When a valid and complete vertex cover set is found by the algorithm, the last value of this variable represents the total number of steps that the algorithms has taken to process the vertices of the minimum vertex cover.

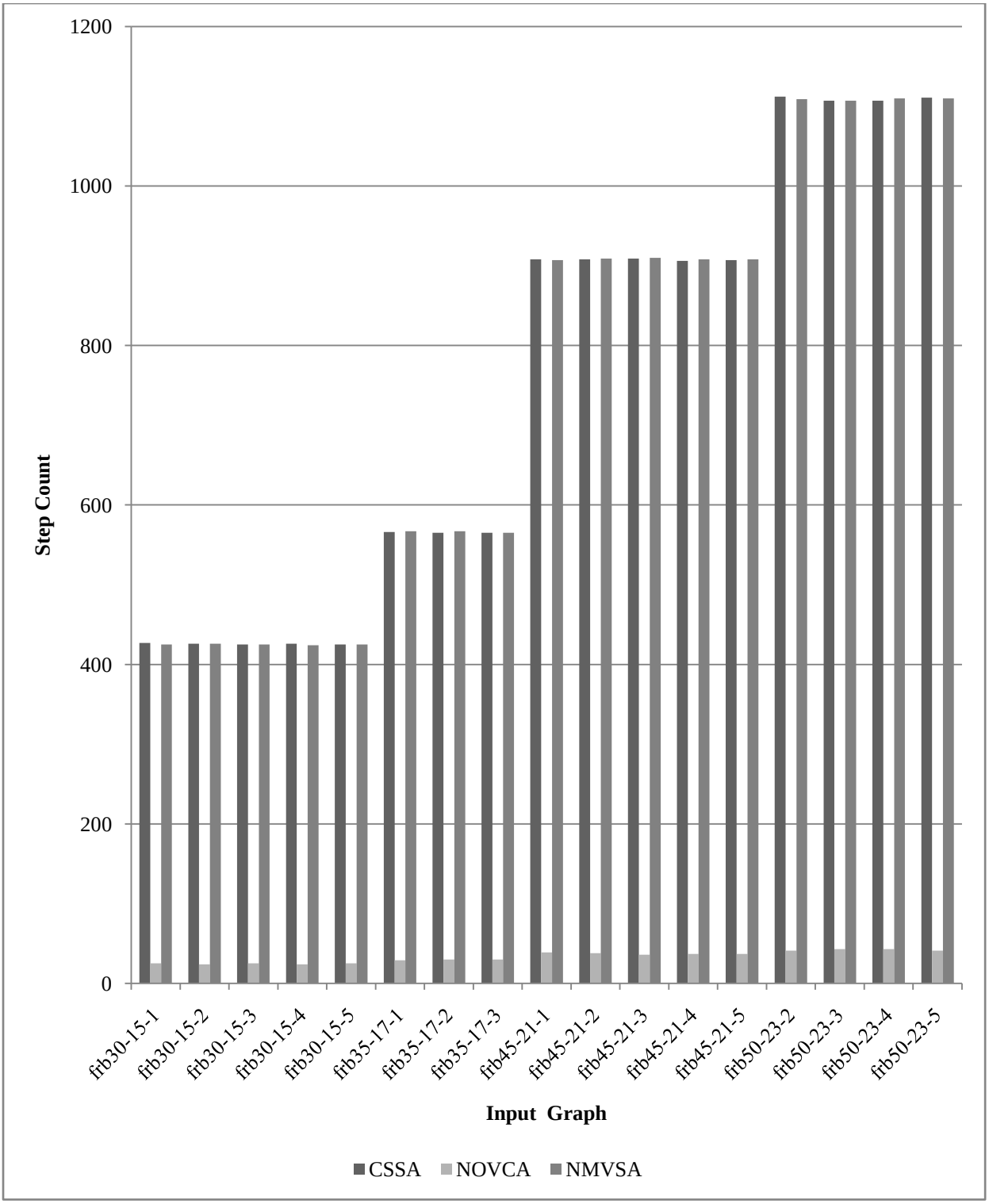


Figure 5.2: (I)Step Count of NOVCA, NMVSA and CSSA

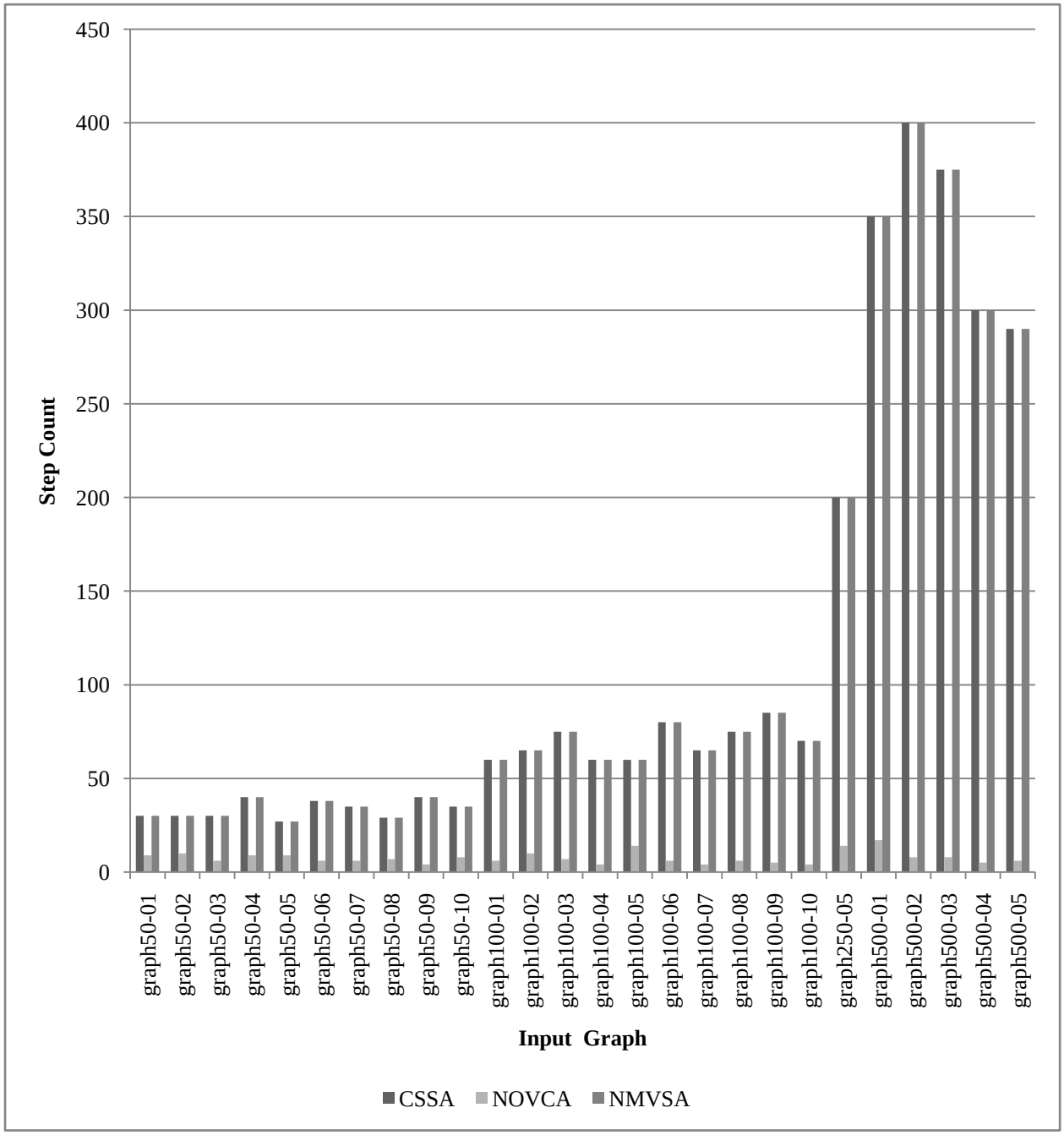


Figure 5.2: (II)Step Count of NOVCA, NMVSA and CSSA

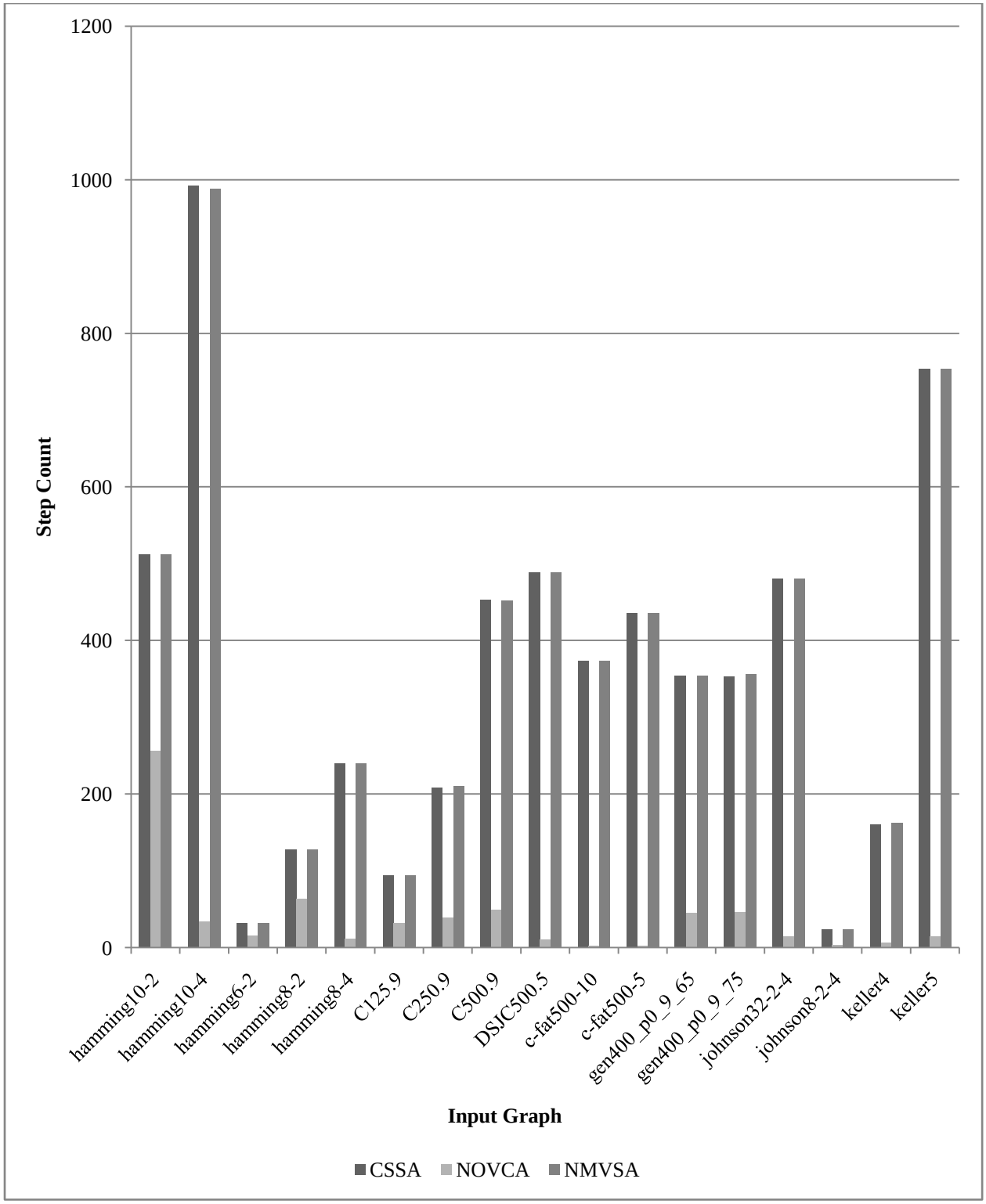


Figure 5.2:(III) Step Count of NOVCA, NMVSA and CSSA

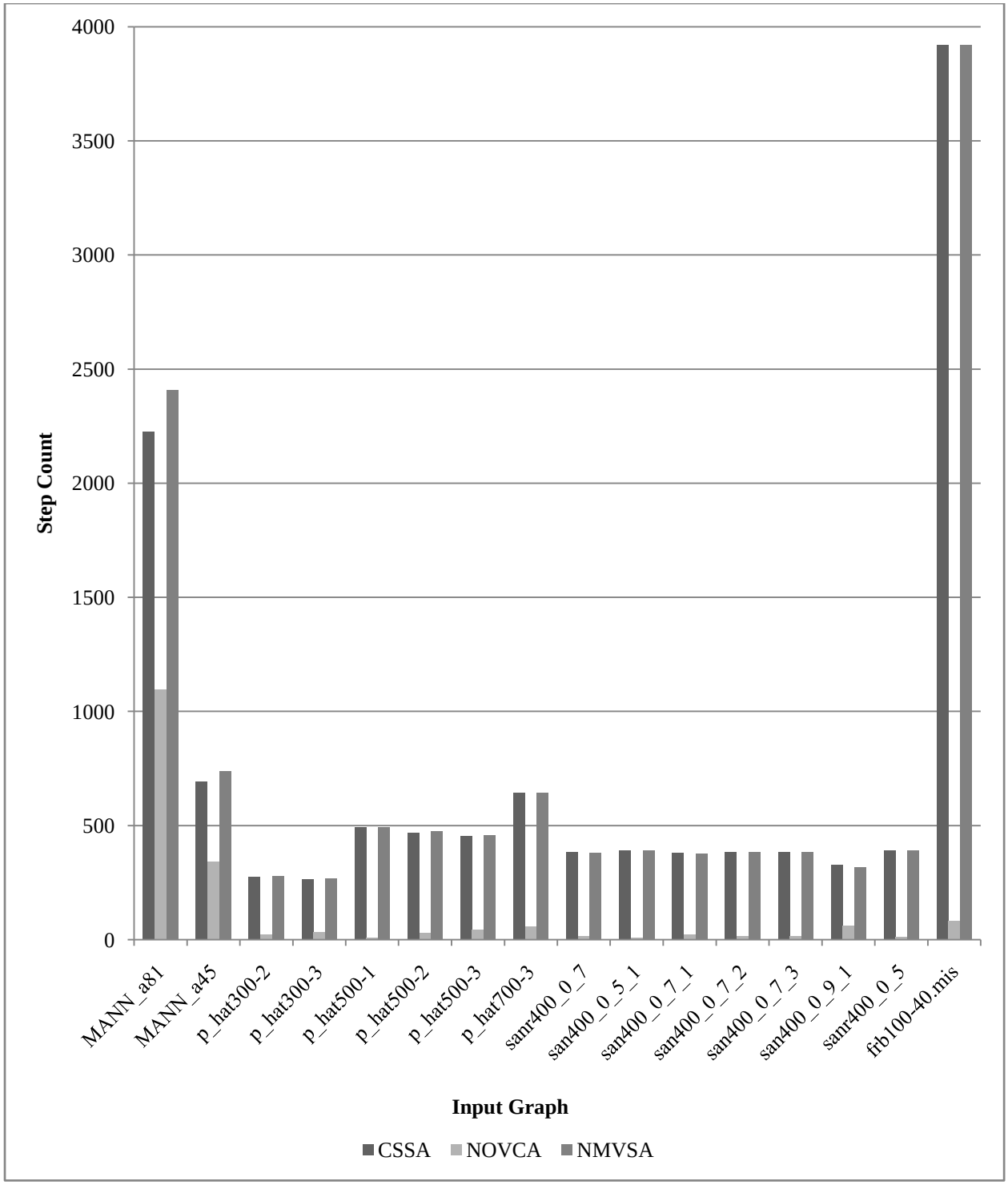


Figure 5.2: (IV)Step Count of NOVCA, NMVSA and CSSA

From the observation of Table 5.3(I) to Table 5.3(IV) and Figure 5.2(I) to Figure5.2(IV), it is obvious that the steps needed to process the possible vertex cover candidates by NOVCA-I is much more less than the step counts of NMVSA and CSSA algorithm. This is because NOVCA-I selects minimum degree vertex and adds its all adjacent

vertices to vertex cover. This strategy adds a number of vertices to vertex cover set in a single step (specially in dense graphs).So that when the adjacent vertices of minimum degree are added to vertex cover list, the degree of the minimum degree vertex adjacent to these vertices will be forcibly rendered to zero. Whereas the CSSA selects one vertex at a time to add it to vertex cover set which obviously shows that step count increases as the size of vertex cover increases. NMVSA tries to add a number of vertices to vertex cover list by selecting those vertices having same maximum support in the neighborhood list of minimum support vertices but it may leave the minimum support vertices without rendering its degree to zero. These vertices may again be selected for further processing but may not be fruitful to select the vertex cover candidates and may add extra steps and vertex to vertex cover. So NOVCA seems to be better than NMVSA and CSSA in term of step counts.

CHAPTER 6
CONCLUSION AND FURTHER RECOMMENDATIONS

6.1 Conclusion

Vertex cover Problem is a NP-complete problem; hence a polynomial time algorithm that finds an optimal solution efficiently is impossible in present context. Approximation algorithm is the best suitable approach to find a acceptable solution to minimum vertex cover problem. From the experimental results and analysis based on the two performance metrics (approximation ratio and step counts), it is observed that the performance of NOVCA-I algorithm outperforms the other two algorithms: NMVSA and CSSA. NOVCA shows better performance in dense graph.The average approximation ratio of NOVCA is 1.008 for the different benchmark graph instances used in this dissertation work. Although the other two algorithms have not so significant difference in approximation ratio with NOVCA but they have step counts much more than NOVCA. From experimental observation and analysis of the results, it can be concluded that the algorithm NOVCA-I is efficient and has better performance in terms of both approximation ratio and step count, than the other two algorithms NMVSA and CSSA.

6.2 Further Recommendation

In this Dissertation work the comparison of three approximation algorithms is done on the benchmark graph datasets. The future scope of the research may include comparison of these algorithms with other algorithms such as hybrid algorithms, genetic algorithms. Algorithm performance enhancement by changing the selection procedure, parameter of the vertex cover candidate, finding the redundant or extra vertex in the vertex cover set and removing these vertices may be some possible research directions.

REFERENCES:

[1] "Vertex cover - Wikipedia," Vertex cover. [Online]. Available: https://en.wikipedia.org/wiki/vertex_cover [Accessed: 15-Jan-2017]

[2] G. Singh, G. Sharma and P. Singh "A Novel Algorithm to solve vertex cover problem," International Journal of Advanced Computation Engineering and Networking, Vol-1, Issue-1, ISSN(p): 2320-2106, pp 60-66, Mar 2013.

[3] T. H. Cormen, C. E. Lieserson, R. L. Rivest and C. Stein, "Introduction to Algorithms," 3rd Ed, Cambridge: MIT Press, 2009.

[4] M. Fayaz, S. Arshad, "Clever Steady Strategy Algorithm: A Simple and efficient approximation algorithm for minimum vertex cover problem," 2015 13th International Conference on Frontiers of Information Technology (FIT), 2015

[5] Y. Khamayseh, W. Mardini, and A. Shatnawi, "An Approximation Algorithm for Vertex Cover Problem," 2012 International Conference on Computer Networks and Communication Systems (CNCS 2012) IPCSIT vol.35 (2012) © (2012) IACSIT Press, Singapore , 2012 .

[6] K. V. R. Kumar," Choosing the efficient algorithm for vertex cover problem," (Masters Thesis, THAPAR UNIVERSITY PATIALA, 2009).

[7] "NP-completeness. - Wikipedia," NP- completeness. [Online]. Available: https://en.wikipedia.org/wiki/NP-completeness [Accessed: 15-Jan-2017]

[8] R. Karp, "Reducibility among combinatorial problems," New York: Plenum Press, 1972.

[9] K. L. Clarkson, "A modification to the greedy algorithm for the vertex cover problem," Information Processing Letters, vol. 16, no.1, pp 23-25, 1983.

[10] F. Delbot and C. Laforest, "Analytical and experimental comparison of six algorithms for the vertex cover problem," Journal of Experimental Algorithmics, vol. 15, 2010.

[11] M. Halldorsson and J. Radhakrishnan, "Greed is good: Approximating independent sets in sparse and bounded-degree graphs," In Proceedings of 26th Annual ACM Symposium on Theory of Computing, New York: ACM, 1994.

[12] F. Delbot and C. Laforest, "A better list heuristic for vertex covers," Information Processing Letters, vol. 107, 2008.

[13] I. Khan, and S. Khan," Experimental Comparison of Five Approximation Algorithms for Minimum Vertex Cover," International Journal of u-and e-Service, Science and Technology, vol.7, no.6, pp 69-84, 2014.

[14] I. Khan. and H. Khan, "Modified Vertex Support Algorithm: A New approach for approximation of Minimum vertex cover," Research Journal of Computer and Information Technology Sciences, ISSN 2320-6527, vol. 1, no. 6, pp. 7-11, 2013.

[15] S. Balaji, V. Swaminathan, and K. Kannan, "Optimization of unweighted minimum vertex cover," World Academy of Science, Engineering and Technology, vol. 43, pp. 716-729, 2010.

[16] S. Bansal, and A. Rana," Analysis of Various Algorithms to Solve Vertex Cover Problem," International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075, vol. 3, issue-12, May 2014

[17] X. Xu and J. Ma, "An efficient simulated annealing algorithm for the minimum vertex cover problem," Neurocomputing, vol. 69, no. 7, pp. 913-916, 2006.

[18] S. Gajurel, and R. Bielefeld," A fast near optimal vertex cover algorithm (NOVCA)," International Journal of Experimental Algorithms(IJEA), vol. 3, issue 1, pp. 9-18, 2012.

[19] S. Gajurel and R. Bielefeld, "A Simple NOVCA: Near Optimal Vertex Cover Algorithm," Procedia Computer Science, vol. 9, 2012.

[20] I. Khan, I. Ahmad and M. Khan, "AVSA, Modified Vertex Support Algorithm for Approximation of MVC," International Journal of Advanced Science and Technology, vol. 67, pp. 71-78, 2014.

[21] M. Eshtay, A. Sliet, and A. Sharieh," NMVSA Greedy Solution for Vertex Cover Problem," International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 7, No. 3, pp.60-64, 2016

[22] J. Chen, L. Kou, and X. Cui, "An Approximation Algorithm for the Minimum Vertex Cover Problem," Procedia Engineering, vol. 137, pp. 180-185, 2016.

[23] I. Khan, and H. Khan," Degree Contribution Algorithm for Approximation of MVC. International Journal of Hybrid Information Technology," vol. 7, no.5, pp. 183-190, 2014.

[24] M. Milanovic," Solving the generalized vertex cover problem by genetic algorithm," Computing and Informatics," vol. 29, no. 6, 2010.

[25] I. Khan, and N. Riaz. " A new and fast approximation algorithm for vertex cover using a maximum independent set (VCUMI)," Operations Research and Decisions, vol.25, no. 4, pp. 5-18, 2015.

[26] "Vertex Cover Benchmark Instances", [Online].
Available: https://turing.cs.hbg.psu.edu/txn131/vertex_cover.html [Accessed: 5-Feb-2017]