T.R.

EGE UNIVERSITY

Graduate School of Applied and Natural Science

A STUDY ON VERTEX COVER PROBLEM AND ITS APPLICATIONS ON WIRELESS NETWORKS

PhD Thesis

Yasin YİĞİT

International Computer Department

İzmir
2020

T.R.
EGE UNIVERSITY
Graduate School of Applied and Natural Science

A STUDY ON VERTEX COVER PROBLEM AND ITS APPLICATIONS ON WIRELESS NETWORKS

Yasin YİĞİT

Supervisor : Assoc. Prof. Dr. Orhan DAĞDEVİREN

International Computer Department
Information Technology Third Cycle Programme

İzmir
2020

Yasin YİĞİT tarafından Doktora tezi olarak sunulan “A Study on Vertex Cover Problem and Its Applications on Wireless Networks” başlıklı bu çalışma EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliği ile EÜ Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi’nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 18.12.2020 tarihinde yapılan tez savunma sınavında aday oybirliği ile başarılı bulunmuştur.

Jüri Üyeleri :		**İmza**
Jüri Başkanı	**:** Doç. Dr. Orhan DAĞDEVİREN	....
Raportör Üye	**:** Prof. Dr. M. Serdar KORUKOĞLU	....
Üye	**:** Doç. Dr. Hasan BULUT	....
Üye	**:** Doç. Dr. M. Alper AKKAŞ	....
Üye	**:** Dr. Öğretim Üyesi Sercan DEMİRCİ	....

EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

ETİK KURALLARA UYGUNLUK BEYANI

EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliğinin ilgili hükümleri uyarınca Doktora Tezi olarak sunduğum “A Study on Vertex Cover Problem and Its Applications on Wireless Networks” başlıklı bu tezin kendi çalışmam olduğunu, sunduğum tüm sonuç, doküman, bilgi ve belgeleri bizzat ve bu tez çalışması kapsamında elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara atıf yaptığımı ve bunları kaynaklar listesinde usulüne uygun olarak verdiğimi, tez çalışması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını, bu tezin herhangi bir bölümünü bu üniversite veya diğer bir üniversitede başka bir tez çalışması içinde sunmadığımı, bu tezin planlanmasından yazımına kadar bütün safhalarda bilimsel etik kurallarına uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul edeceğimi beyan ederim.

18.12.2020

Yasin YİĞİT

ÖZET

DÜĞÜM ÖRTÜSÜ PROBLEMİ VE KABLOSUZ AĞLARDAKİ UYGULAMALARI ÜZERİNE BİR ÇALIŞMA

YİĞİT, Yasin

Doktora Tezi, Uluslararası Bilgisayar Anabilim Dalı
Tez Danışmanı: Doç. Dr. Orhan DAĞDEVİREN
Aralık 2020, 86 sayfa

Nesnelerin interneti kavramının giderek önem kazandığı günümüz dünyasında, birçok cihaz aynı ağ üzerinde ortak bir hedef için dağıtık olarak çalışarak günlük hayatımızı ve üretim süreçlerini önemli ölçüde kolaylaştırmaktadır. Bu tür ağlarda bağlantıların izlenmesi ve meydana gelen kopmalarda ağı yeniden ayaklandırmak büyük öneme sahiptir. Bu noktada ağlardaki bağlantıları izlemek için düğüm örtüsü problemi karşımıza çıkmaktadır. Düğüm örtüsü, çözüm kümesine giren düğümlerin çizge üzerindeki bütün bağlantıları izleyebildiği bir yapı sağlamaktadır. Bir hata sonucu ağdaki düğüm örtüsü özelliği bozulduğunda, ağı tekrardan düğüm örtüsü konumuna getirmemiz gerekir. Bu durumda karşımıza öz-kararlılık kavramı çıkmaktadır. Öz-kararlık sistemin olası bir hata durumunda dış müdehale olmaksızın kararlı hale erişebilmesi olarak tanımlanır. Telsiz duyarga ağları gibi tasarımsız yapılarda enerjinin ve kaynakların iyi yönetilmesi ve kullanılması gerekmektedir. Bu kısıtlamalardan yola çıkarak kapasite kısıtlı çizge teorik problemler önerilmiştir. Bu çalışmada iki önemli çalışma alanı olan öz-kararlılık kavramını ve kapasite kısıtlı problemleri düğüm örtüsü çatısı altında birleştirerek özgün algoritmalar önermekteyiz. Önerdiğimiz algoritmaların doğruluğunu teorik olarak ispatlayıp, benzetim sonuçlarını vererek tezimizi güçlendirimekteyiz. Önermiş olduğumuz SS-CVC1 ve SS-CVC2 algoritmaları gerek zaman gerek düğüm örtüsü performansı açısından tüm rakiplerinden daha başarılı sonuçlar vermişlerdir. Enerji kullanımı açısından, SS-CVC1 algoritması enerji tüketimi en az olan algoritma olmuştur. Önerilen algoritmalara ek olarak bazı önemli düğüm örtüsü algoritmalarının kapsamlı performans değerlendirmelerine yer verilmiştir.

Anahtar sözcükler: Düğüm örtüsü problemi, dağıtık algoritmalar, öz-kararlılık, çizge teorisi, kapasite kısıtlı çizge teorik algoritmalar, kapasite kısıtlı düğüm örtüsü

ABSTRACT

A STUDY ON VERTEX COVER PROBLEM AND ITS APPLICATIONS ON WIRELESS NETWORKS

YİĞİT, Yasin

PhD in International Computer Department
Supervisor: Assoc. Prof. Dr. Orhan DAĞDEVİREN
December 2020, 86 pages

In today's world, where the concept of the Internet of Things is becoming increasingly popular, many devices work distributed over the same network for a common goal, significantly simplifying our daily life and production processes. In such networks, it is of great importance to monitor connections and to make the network available in case of breaks. At this point, in order to monitor connections in networks, we encounter the vertex cover problem. The vertex cover provides a structure where the nodes entering the solution set can monitor all connections on the graph. When the vertex cover property in the network breaks down as a result of a break, we need to bring the network back to the property where satisfies the vertex cover. In this case, the concept of self-stabilization arises. Self-stabilization can be defined as the ability of the system to reach the desired stable state again without external intervention in the event of a possible fault. In ad-hoc structures such as wireless sensor networks, energy and resources need to be well managed and used. Based on these constraints, capacitated graph theoretical problems have been proposed. In this study, we propose novel algorithms by combining the two significant fields of study, the concept of self-stabilization and capacitated problems, under the roof of the vertex cover problem. Having proved theoretically the correctness of the proposed algorithms, we shore up our thesis by giving simulation results. The proposed SS-CVC1 and SS-CVC2 algorithms provide better results than all of their counterparts in terms of both time and node cover performance. In terms of energy consumption, the SS-CVC1 algorithm has become the most efficient algorithm. In addition to the proposed algorithms, comprehensive performance evaluations of some important vertex cover algorithms are included in this dissertation.

Keywords: Vertex cover problem, distributed algorithms, self-stabilization, graph theory, capacitated graph-theoretic algorithms, capacitated vertex cover

PREFACE

This doctoral thesis has been realized as part of the TUBITAK ARDEB 1001 project coded 215E115. Following the design and implementation of the integrated vertex cover algorithms discussed in the master's study, it introduces a new vertex cover problem and algorithms that provide the properties of self-stability and capacity constraints. After the one-year course period, the thesis was finalized in a total of 6 semesters with the proposal and the thesis monitoring process. One of the thesis period's semester was conducted at Heidelberg University.

I hope that the use of this thesis will not be limited within the boundaries of graph theory but in any field in which have fault-tolerance and self-stabilization.

İZMİR

18.12.2020 Yasin YİĞİT

TABLE OF CONTENTS

Page

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

LIST OF FIGURES

LIST OF FIGURES (continued)

Figure Page

LIST OF FIGURES (continued)

LIST OF TABLES

LIST OF ALGORITHM

LIST OF ABBREVIATIONS

Abbreviation	Explanation
IoT	Internet of Things
WSN	Wireless Sensor Network
UDG	Unit Disk Graph
VC	Vertex Cover
CVC	Capacitated Vertex Cover
MIS	Maximal Independent Set
TDMA	Time Division Multiple Access
IS	Independent Set
ULME	Unbounded Local Mutual Exclusion
SS-CVC	Self-Stabilizing Capacitated Vertex Cover
UAV	Unmanned Aerial Vehicle
MIMO	Multiple-In Multiple-Out

LIST OF SYMBOLS

Symbol	Explanation
$G(V, E)$	Abstract representation of a distributed system
V	Set of all v in distributed system
E	Set of all e in distributed system
v	Processing unit of a distributed system
$e = (u, v)$	Communication channel between two incident vertices
$N(v)$	Neighbors list of vertex v
$\delta(v)$	Degree of vertex v
Δ	Maximum degree of G
s_v	State of vertex v
c	Configuration of G
m_v	Move made by v
x_v	Existence of v in vertex cover
$\aleph$	Infinite set
w_v	Weight of vertex v
y_{ev}	Coverage of e by v
$b(v)$	Bounding value of vertex v
nex	Existence count of a vertex
cap	Capacity of a vertex

1. INTRODUCTION

Tiny smart devices have been frequently involved in our daily lives after the development of chip technologies. These devices create networks and are able to work independently and distributed manner to realize a common aim, such as collecting data from the environment like forests, farms, and even oceans. In the modern era, the Internet of Things (IoT), cloud computing, and wireless sensor networks (WSN) are extensively used by researchers, companies, and developers. New challenges and problems which stem from the increasing usage of distributed systems, attract the attention of researchers who come from different academic fields like computer science, electrical engineering, and combinatorial mathematics, etc.

A WSN is a set of devices that can collect various data such as temperature, light, and humidity from the environment and communicate with each other to send the gathered data to a processing center (Akyildiz et al., 2002). WSNs have been studied in the literature in areas like multi-hop data transmission, construction of beacon node-set, and coverage hole detection (Kavitha and Caroline, 2017; Bin and Jiang, 2018; Feng et al., 2018). Due to the small dimensions, the nodes in WSNs, which are also called motes, usually have limited energy source, transmission range, processing power, and memory. Hence, the applications and algorithms must use wireless sensor devices in an energy-efficient manner to increase the network lifetime (Gowrishankar et al., 2008). In such a network, due to environmental challenges or technical problems, nodes could crash and links between nodes could disconnect. Self-stabilization is used to maintain the robustness of the system by keeping its legitimate state against any fault such as link failures, crashes, or message drops. A distributed system which reaches the legitimate state without any external aid in case of any fault or arbitrary initial state and keeps the system in this stable configuration until a fault occurs, is regarded as self-stabilizing system. Self-stabilization has been introduced by Dijkstra to solve the mutual exclusion problem on ring topologies (Dijkstra, 1974). Nowadays, self-stabilization is used extensively in graph-theoretical problems like vertex cover, maximal independent set, and graph matching. Due to the restricted structure of tiny wireless devices and environmental challenges, covering and communication capacities of these devices are limited. Thus, each device can monitor and communicate the restricted number of links assigned to them. To solve this type of problem, researchers have proposed graph-theoretical problems which are restricted and more challenging versions of the original problem.

We can model a WSN as a graph $G(V, E)$, where V is the set of nodes and E

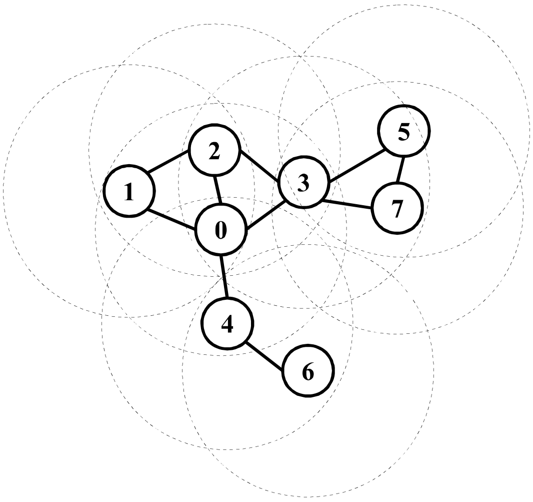


Figure 1.1. Network model.

is the set of links between the nodes. Figure 1.1 shows a sample network model as a Unit Disc Graph (UDG) where the dotted circles are the transmission ranges and solid lines show the communication links between the nodes.

The nodes in WSNs are connected in an ad-hoc manner and failure in some nodes may cut-offs the available paths and affects the routing performance. Hence, in these networks, link monitoring is one of the vital tasks which must be carried on by a minimum subset of nodes to minimize the energy consumption overhead. Vertex cover (VC) is one of the well-known problems in graph theory which has many applications in real-world tasks such as optimizing of worm propagation for network security (Filiol et al., 2007), solving Single Nucleotide Polymorphism problem in computational biochemistry (Lancia et al., 2001) and link monitoring in wireless sensor networks (Erciyes, 2013). A vertex cover of a given undirected graph *G(V,E)* is a set C such that $\forall e \in E$ is incident to at least one vertex of $C \subseteq V$. Karp has proved that the optimization version of the problem is in the NP-Hard complexity class (Karp, 1972). Capacitated vertex cover (CVC) problem is a generalized version of VC which restricts the nodes with bounding number of edges covered by an individual vertex in a graph.

This thesis contributes to the literature as follows:

- We provide an extensive literature review for vertex cover problems in different settings like distributed, self-stabilizing, and central. We investigate these algorithms and compare them theoretically, so we show their theoretical comparison against each other.

- We provide performance evaluation of the reviewed algorithms as below:
  - We evaluate central vertex cover algorithms that exploit vast of design techniques.
  - We investigate vertex cover in distributed environment where we extend the study which we conducted as a master thesis. We add new algorithms to compare and provide test-bed experiments to understand the behavior of algorithms in real-world applications.
  - In order to understand the capacitated vertex cover problem, we compare performance of the linear programming algorithms which are proposed to solve the capacitated vertex cover algorithm.
  - We implement vertex cover algorithms and independent set algorithms to perceive performance of link monitoring performance while the first group of algorithms is designed by using graph matching algorithms and the second group is using the greedy approach to solve independent set which is complementary to vertex cover solution on a given graph.
- Having made a comprehensive investigation and analysis, we see the strengths and weaknesses of different vertex cover algorithms both theoretical and practical. By using this knowledge, we propose two novel algorithms named SS-CVC1 and SS-CVC2 that combine capacitated and self-stabilizing settings. SS-CVC1 algorithm stabilizes at most $\mathcal{O}(n)$ while SS-CVC2 stabilize $\mathcal{O}(n^2)$ step. As far as we can understand from our studies, both algorithms are the first contribution to the capacitated and self-stabilizing vertex cover problem.

We organized the dissertation in two parts: the first part provides analysis and performance evaluations of algorithms while the second one presents our novel algorithms. The detailed explanation of the organization of the thesis is as follows:

- In Chapter 2, we give some information about the distributed system, distributed algorithm and self-stabilization to warm up the thesis.
- We formulate and deeply explain the vertex cover problem in Chapter 3 to clarify the problem and give our motivation and real-world application of the problem in this chapter.
- We provide related works for vertex cover problem which used in different types of domains in Chapter 4.

- We implement the analyzed algorithm and provide comprehensive performance results in Chapter 5.

- In Chapter 6, we present two algorithms to construct the capacitated vertex cover solution on a given graph with self-stabilization property. We theoretically proved these algorithms and evaluated on the simulator with self-stabilizing algorithms which were modified to satisfy the capacitated property.

- We finalize the dissertation in Chapter 7 by outlining and summarizing the results and findings of our study.

2. BACKGROUND

2.1 Distributed Systems, Graphs and Distributed Algorithms

Over the past two decades, distributed systems have gained importance in the commercial and academic areas because of geographical distribution, speeding up to computation time, resource sharing, and fault tolerance for a single point of failure. In order to talk about the existence of a distributed system, the following 3 criteria must be met.

- **Multiple processing units:** The system should contain more than one processing unit with an individual thread of control.
- **Inter-process Communication:** Each processor should communicate with its neighbor processor over an message or shared memory that is fetched in a finite amount of time.
- **Common Aim:** Each processor must interact with others to realize a common aim.

A distributed system is abstracted as a graph which is a tuple $G(V, E)$ where V is the set of vertices (processing units) and E is the set of edges (communication links). The processing units of a distributed system are represented by a set of vertices V and the communication links between these vertices are represented by E. WSNs are represented as a graph, where each sensor mote corresponds to a vertex v of the graph. Each communication channel $e = (u, v)$ between two sensor devices is an instance of set E in graph G. To provide a solution to a distributed problem, distributed algorithms are designed to be concurrently run by individual processing units on a distributed system to reach a common goal. In contrast to centralized algorithms, processors do not have global information about the given topology.

If each edge points only one end-point of it, the graph is called as a directed graph, while if the edges function in two directions, the graph is called as un-directed. In this thesis, we conducted a study on un-directed graphs referred to as ”*graph*” for simplicity. In a graph, two vertices u and v are considered as neighbor if and only if $e = (u, v)$ is in the edge set of the graph.

Definition 2.1 (Neighbors)**.** The neighbors set of an vertex v in graph $G(V, E)$ is a function $N(v) \rightarrow V$ and formulated as $N(v) = \{u \in V \mid (u, v) \in E\}$. The cardinality of $N(v)$ gives the degree of the nodes represented as $\delta(v)$ or $|N(v)|$. Furthermore, the maximum degree of graph G is denoted by Δ.

Throughout the dissertation, we use the terms node, vertex, processor to state $v \in V$ and link, edge to state $e \in E$ interchangeably.

2.2 Self-Stabilization

In large-scale distributed systems, failures are quite prevalent due to the nature of these types of systems. Restoring the system to its steady state by external intervention could be harmful because it may cause inevitable problems in the future. Therefore, the internal fault recovery system must be located on distributed systems.

Fault-tolerant techniques are divided into two basic types as follows:

- **Masking:** In such fault-tolerant systems, the effect of failure is completely invisible to applications which include safety-critical systems such as real-time systems, database systems, and financial applications.
- **Non-masking:** The effect of faults is visible until the system resumes behaving appropriately in these types of fault-tolerant systems.

Self-stabilization is a non-masking fault-tolerant technique that brings the system legitimate state without any external aid or intervention in a finite amount of time when a transient fault occurs or given any arbitrary initial states. A transient fault can randomly disrupt the stable configuration of the system. The cause including this failure can take a short time but its consequences may have an impact on the global state permanently. In order to realize a self-stabilizing system, researchers develop self-stabilizing algorithms which satisfy the following two properties (Dolev, 2000):

- **Convergence:** Regardless of the initial state of the system, the algorithm reaches the legal state after a limited amount of time.
- **Closure:** The legitimate state of the system is preserved until a transient fault occurs.

Hauck has associated self-stabilization with a wobbly man that fulfills both properties because it reaches its balanced position from the initial random position (*convergence*) and keeps its balanced state until a fault (*closure*) (Hauck, 2012).

Definition 2.2 (State). Each vertex $v \in V$ holds a finite set of variables $\{var_1, var_2,$ ..., $var_{k-1}\}$. The s_v state of a vertex v is represented by the values of variables.

Definition 2.3 (Configuration). A configuration c of graph G is the set of states of all vertices $v \in V$ which represented as $c = \{s_1, s_2, s_{n-1}\}$.

Definition 2.4 (Legitimate). Let the predicate P state that fault-free configuration called c. The configuration c is legitimate since it satisfies the predicate P.

A self-stabilizing algorithm is built with a set of rules. Each rule contains two parts named as *precondition* and *action*. *Precondition* is a *boolean* predicate based on the state of the vertex and its neighbors' state.

Definition 2.5 (Enabled Rule). A rule is *enabled* in a configuration c in case of its precondition holds *true*. A vertex is called as *enabled* if one of its rules is enabled.

When more than rules of an vertex is enabled, one of them is arbitrarily chosen for progress. However, negating the rules is another way to design a self-stabilizing algorithm in order to guarantee that only one rule is enabled per vertex.

Definition 2.6 (Move). A move is the change of the state of a vertex before execution and after the execution, and it is represented as a tuple m_v= $(s_b, s_a)_v$.

Definition 2.7 (Step). A non-empty set of simultaneous moves that carry the system from configuration c to configuration c'.

If the central scheduler is of concern, a step is equal to a single move of a node. Thus, a step includes the move $m = (s, s')$ which carries the system from c to c' (i.e. $m = (c, c')$).

Definition 2.8 (Round). A round is a minimal sequence of steps where all nodes are privileged to make their moves by the scheduler or disabled by a consequence of their neighbors' moves.

Definition 2.9 (Execution)**.** An execution of self-stabilizing algorithm is a maximum sequence of configurations $\{c_0, c_1, c_2, ...\}$ where c_{i+1} is the result of the step which is taken in c_i.

2.2.1 Schedulers

Algorithm 2.1 Shukla's MIS Algorithm

```
R1: $(s_u = 0) \wedge \forall v \in N(u) : s_v = 0$
    $\Rightarrow s_u := 1$
R2: $(s_u = 1) \wedge \exists v \in N(u) : s_v = 1$
    $\Rightarrow s_u := 0$
```

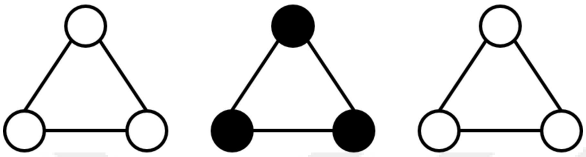


Figure 2.1. Sample execution of Shukla's algorithm when all nodes are allowed to makes their move.

The necessary property of a self-stabilizing system is its synchronization that decides which vertices will execute their enabled rules. Consider Algorithm 2.1 which has been proposed by (Shukla et al., 1995) to construct a maximal independent set (MIS) (as well as a vertex cover since both problems complement each other). The algorithm keeps running infinitely if all enabled rules are privileged to make their moves for a sample scenario given in Figure 2.1.

To prevent this situation, in this example, the schedulers (*i.e. deamon*) term is used when researchers develop an self-stabilizing algorithm. A scheduler is a decision mechanism to select a non-empty set of all enabled nodes to maintenance the progression of the algorithm. Tixeul divides schedulers into 2 parts according to their characteristics as follows (Tixeuil, 2009):

- **Spatial Scheduling:** Self-stabilizing algorithms were based on the hypothesis that two neighbor nodes can not make their moves simultaneously in its early era (Angluin, 1980). In the modern era of self-stabilizing algorithms, there are 3 main types of scheduler to decide how the system privileged their nodes to make their moves.
  - *Central scheduler:* In these type of schedulers, only one processing unit can make its move in a given moment. This scheduler was proposed earlier research on the self-stabilizing area (Dijkstra, 1974).

  - *Synchronous scheduler:* All enabled processing unit can execute their code simultaneously.
  - *Distributed scheduler:* In any configuration of the system, this scheduler allows any nonempty subset of the whole enabled nodes to make their moves. Note that the distributed scheduler subsumes two other schedulers and nodes can make moves at different time in the execution of the algorithm. Because of these properties, the distributed scheduler is more realistic than the other for real-world applications.

- **Temporal Scheduling:** This property refers to the fairness of schedulers as follows:
  - *Fair*: The scheduler allows all enabled nodes to makes their moves infinitely often.
  - *Unfair (i.e. adversary):* The scheduler does not guarantee the execution of all enabled vertices in a step but the global progression of the system. In other words, in a moment, the scheduler must privilege at least one node.

2.3 Complexity Measure of Self-stabilizing Algorithm

The maximum number of resource demand defines the complexity measure of an algorithm. For the case of distributed setting, the complexity measure increases with the number of nodes that run on the system. The resources could be execution time, sent and received messages, or memory usage. On the other hand, we encounter different types of complexity measures like move, step and round complexities. These complexities define the maximum number of moves (steps, round) needed to gain a stable configuration independent of the initial configuration.

For WSNs, message traffic is the most energy demanded action during the execution of the algorithm. After a move, vertices should send their new state to all of its neighbors. Therefore, the move count has a direct influence on the message complexity of any algorithm which runs on a self-stabilizing setting. Therefore, we should consider this knowledge to develop a self-stabilizing algorithm for WSN.

2.4 Distance-k Knowledge

Based on the nature of the distributed algorithms, a node could read its own and its one-hop neighbors’ variables (distance-1 knowledge). However, in some individual cases, it is easier to assume that each node can access the variables of nodes that are located 2 or more hops distances. To realize this kind of algorithm to execute in a distributed system, it must be transformed into the distance-1 algorithm. Several transformers were proposed in the literature to fetch distance-*k* knowledge(Gairing et al., 2004; Goddard et al., 2008; Turau, 2012).

According to Gairing's technique in which each node holds its neighbors' state along with its own state, on the condition that this information is needed, it must be updated by the node (Gairing et al., 2004). Additionally, each node can make a move by permission of its all neighbors. By the way, only one node can make a move in distance-1 neighborhood. The slowdown factor of this approach is $\mathcal{O}(n^2m)$ and memory overhead is $\Omega(\Delta \log n)$. (Goddard et al., 2008) have extended the idea that was proposed by (Gairing et al., 2004). It is a recursive algorithm that distributes distance-*k* knowledge over the topology, but its slowdown factor and memory requirement are $n^{\mathcal{O}(\log k)}$.

Turau have proposed a novel technique called as *expression model* (Turau, 2012). In this technique, a node holds its local variables and a set of expressions. The value of an expression is determined by the states of the node and its neighbors. A node can not read its 2-hop neighbors' state, but it can evaluate the expressions of its neighbors. The critical advantage of the expression model is that its slowdown factor is only $\mathcal{O}(m)$ and its memory usage is adjustable by the base algorithm. We used this technique to develop self-stabilizing and capacitated vertex cover algorithm in this thesis.

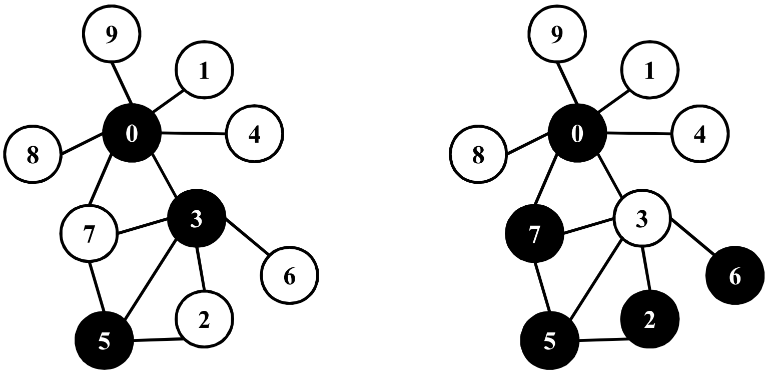
3. VERTEX COVER PROBLEM FORMULATION

VC problem is one of the major problems in graph theory, which has many important applications such as router placement and link monitoring in networks.

Given a graph $G(V, E)$, a vertex cover $C \in V$ is a set of vertices such that for any edge $(u, v) \in E$, either $u \in C$ or $v \in C$ (Erciyes, 2013).

$$\begin{aligned} \text{min} \quad & \sum_{v \in V} w_v \times x_v \\ \text{s.t.} \quad & x_u + x_v \geq 1 \qquad x \in V \\ & x_v \in \{0, 1\} \end{aligned} \tag{3.1}$$

If the problem only takes into consideration the cardinality of the solution set, this problem is called *cardinality vertex cover*. A generalized version of the vertex cover problem is the *weighted vertex cover* problem, where each vertex $v \in V$ has a weight w_v. When $w_v = 1$ for all vertices in the graph G (i.e, each vertex is unit weighted), this problem turns *cardinality vertex cover* problem. Equation 3.1 shows the linear programming formulation of the weighted vertex cover problem, where x_v states whether the vertex is in the cover set or not.



(a) Minimum C ={0, 3, 5}.

(b) Minimal C={0, 2, 5, 6, 7}.

Figure 3.1. Sample VC solutions.

V is a satisfying answer for the vertex cover problem, but in most applications, we should find a vertex cover with the minimum number of nodes which is an NP-Complete problem (Karp, 1972). However, some central heuristics and approximation algorithms can find acceptable solutions (Cormen, 2001). Formally, a minimum VC is a VC with minimum cardinality among all VC solutions. A minimal VC is a vertex cover which its size cannot be decreased (Ileri et al., 2016a).

Figure 3.1a shows a minimum $C = \{0, 3, 5\}$ and Figure 3.1b shows a minimal $C = \{0, 2, 5, 6, 7\}$.

When developing a VC algorithm, we must consider the following properties:

- VC should be as small as possible.
- Designed algorithms should be efficient in terms of time, space, message complexity, and energy consumption.

The vertex cover problem has attracted the interest of many researchers. As an approximation problem, the minimum approximation ratio of VC problem is 1.36 for central algorithms (Dinur and Safra, 2002) and the best upper bound is $2 - \Theta(\log_2(n))$ (Karakostas, 2009). There are many algorithms that use different types of techniques such as depth-first search, local-ratio theorem, semi-definite relaxation, and graph-theoretic techniques to solve the vertex cover algorithm (Savage, 1982; Bar-Yehuda and Even, 1985; Halperin, 2002; Håstad and Johan, 2001). The exact solution for the problem can be found with the fixed-parameter tractability technique in 1.28^k time complexity, where k is an integer parameter to find a vertex cover of at most k vertices (Niedermeier and Rossmanith, 2003).

In distributing settings, the solution for the vertex cover problem can end up with 2-approximation ratio on the condition that maximal matching is provided (Grandoni et al., 2008; Hanckowiak et al., 2001; Panconesi and Rizzi, 2001). Parnas and Ron proposed a greedy technique to solve the vertex cover problem in distributed settings (Parnas and Ron, 2007). Kavalci et. al. proposed a breath-first search integrated algorithm that runs on wireless sensor networks (Kavalci et al., 2014). Also, there are some studies on the self-stabilizing manner in which the system maintains by itself in the presence of any failure or arbitrary initial state (Kiniwa, 2005; Turau and Hauck, 2011). Yigit et. al. studied a comprehensive comparison of the self-stabilizing algorithm for link monitoring case study on wireless sensor networks in (Yigit et al., 2018). We give a detailed literature review of different settings of the vertex cover problem in Chapter 4.

3.1 Usage of Vertex Cover in Real Life

Vertex cover is used extensively in real-life problems from social life to WSNs. In this section, we introduce real-life applications of vertex cover with examples.

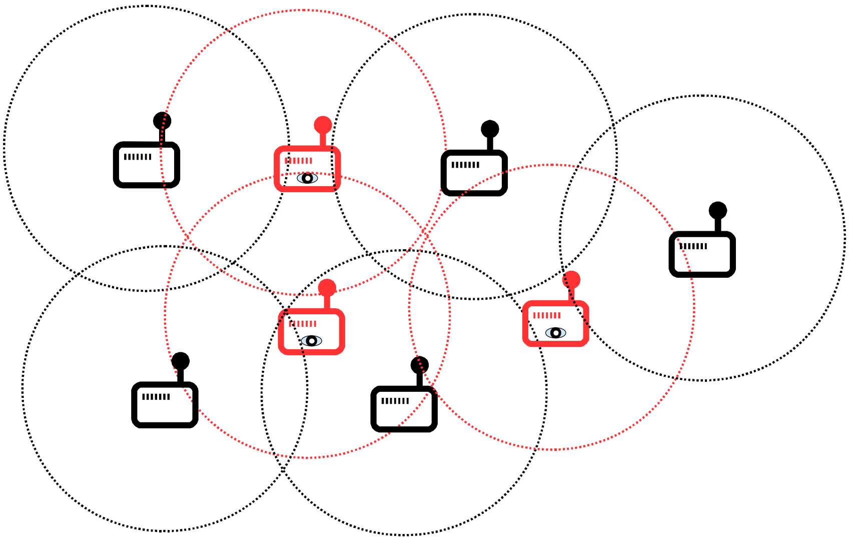


Figure 3.2. An example of link monitoring application for vertex cover problem.

In social networks where each person is represented as a vertex v of graph G, the vertex cover problem is used for modeling adaptation a newly released product. Consider each individual person has a pre-defined *threshold*. A person is considered to be influenced by the product if the number of neighbors who have already adopted the product is equal to the threshold. When the threshold value is equal to the number of neighbors (i.e. the degree of vertex $\delta(v)$), the problem becomes the vertex cover problem (Chen, 2009).

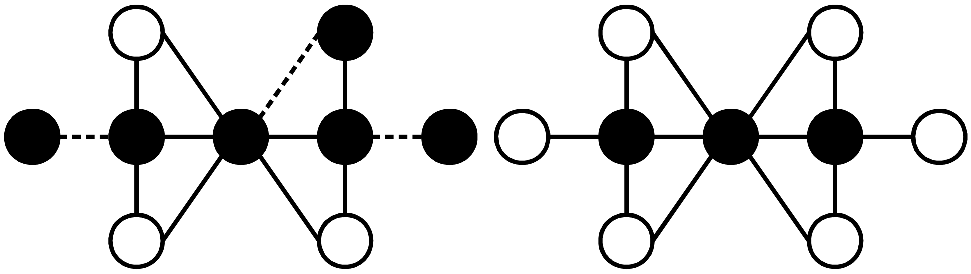
In flow networks such as transporting on roads, the vertex cover is used to monitor the roads that are modeled as edges on the graph by placing cameras on the crossroads that are modeled as vertices on the graph (Tamura et al., 2001). On the other hand, in order to simulate the propagation of worms (i.e. viruses) on large computer networks and design favorable strategies to protect networks against malicious attacks, vertex cover is used (Filiol et al., 2007).

Monitoring the health of all links in the network is crucial for wireless networks to maintain the quality of networks. Kumar and Kaur have proposed a method that is derived from the vertex cover problem to solve the beacon placement problem for monitoring links in the network (Kumar and Kaur, 2004). Figure 3.2 shows an example of link monitoring on a wireless network, where red devices are the monitoring motes (or beacons). In this example, beacons can listen and collect all network traffic. In this way, the network administrator is warned when a link failure occurs.

Replication of the desired data at the server, closer to the client, can reduce the access time to the data and bandwidth usage. Replica placement is mostly studied in tree networks to allow users to access the data efficiently. The solution selects $R \subseteq V$ and places the replicated data onto them to satisfy the request of the client in the network. Arora et al. have studied the capacitated vertex cover problem for replica placement in tree networks (Arora et al., 2013).

4. LITERATURE REVIEW of VERTEX COVER ALGORITHMS

4.1 Literature Review of Sequential Vertex Cover Algorithms



(a) Vertex cover via matching.

(b) Optimal solutions via greedy algorithm.

Figure 4.1. Detected VC by Greedy and Matching algorithms.

In any graph, the size of VC varies between 1 and $n-1$, where n is the number of vertices in the graph. Thus, we can find minimum VC by 2^{n-1} trial with a brute force algorithm which leads to $\mathcal{O}(2^n)$ time complexity.

The smallest approximation ratio for VC problem is 1.36 for central algorithms and the best upper bound is $2-\frac{1}{\log_2(n)}$ (Dinur and Safra, 2002; Karakostas, 2009). In the matching problem, a maximum matching forms a VC which redundantly covers one edge by two vertices and provides a solution with 2-approximation ratio. In higher degrees, a greedy approach can find optimal solutions in some situations (Erciyes, 2013). Figure 4.1a shows a sample network where the maximal matching covers with 6 vertices, but greedy algorithm covers with 3 vertices as an optimal solution. In both figures, the dark vertices are the members of VC.

4.1.1 Polynomial Time Algorithms

2-approximation algorithm selects and removes edge $e = (u, v)$ from given graph $G(V, E)$ and adds both end-points of the edge e until there is no edge on the G. The pseudo-code of the 2-approximation vertex cover algorithm is given in Algorithm 4.2.

Theorem 4.1. The time complexity of the algorithm is $O(m)$ where m is the cardinality of the edge set.

Proof. The algorithm runs until there is no edge left in the graph where the complete graph is the worst case for this algorithm. Thus, it runs at most $\mathcal{O}(m)$ where $m = n \times (n-1)$ for complete graph instance. □

Algorithm 4.2 2-approximation VC Algorithm

```
Require: $G(V, E)$
Ensure: $C \subseteq V$
$C \leftarrow \emptyset$
while $E \neq \emptyset$ do
    Select an $e = (u, v)\}$
    $C \leftarrow C \cup \{u, v\}$
    Remove {u,v} from V
end while
```

Theorem 4.2. 2-approximation algorithm ensures that produce the result at most 2 times worse than the optimal solution.

Proof. Let E' be the edge set which contains edges selected by algorithm and C^* be optimal vertex cover solution for given graph $G(V, E)$. C^* must cover every edge in E'. Therefore, it must include at least one of the endpoints of each edge $e = (u, v) \in E'$, where no 2 edges in E have the same endpoint. Hence, the size of the optimal vertex cover must be $|C^*| \geq |E|$. In addition to these, the 2-approximation algorithm produces a vertex cover solution at most $C = 2 \times | E'|$ in size. These two equations are combined, the result is $C = 2 \times |E'| \leq |C^*|$ and concluded that the algorithm has 2 approximation ratio. □

Although heuristic algorithms give acceptable and good results, they do not theoretically promise an optimal solution. Greedy vertex cover algorithm chooses the vertex that has maximum $\delta(v)$ and puts it to the solution set at every iteration until there is no vertex on the given graph $G(V, E)$ as seen in Algorithm 4.3. When the graph become empty, the algorithm exits from the WHILE loop and return solution set.

Algorithm 4.3 Greedy Vertex Cover Algorithm

```
Require: $G(V, E)$
Ensure: $C \subseteq V$
$C \leftarrow \emptyset$
while $E \neq \emptyset$ do
    $v \leftarrow argmax(\delta_v | v \in V)$
    $C \leftarrow C \cup \{v\}$
    Remove$\{v\}$from graph
end while
```

Theorem 4.3. Time complexity of Algorithm 4.3 is $\mathcal{O}(m)$.

Proof. The greedy algorithm continues running while there are edges in E. Assume an execution which algorithm deletes only one edge from E (i.e. chain), this takes $|E| = m$ step until all edges are removed. □

Theorem 4.4. Approximation ratio of Algorithm 4.3 is $\mathcal{O}(\log(n))$.

Proof. Since the greedy vertex cover algorithm is derived from the greedy set cover algorithm, it is easily said that the complexity of the greedy vertex cover algorithm is $\mathcal{O}(\log(n))$. □

4.1.2 Optimal Algorithms

The algorithms that use the bounded-search tree method and produce an optimal solution, are elaborated in this subsection. The bounded-search tree method searches solution on reduced graph $G' \subseteq G$ and $k' \leq k$ where k is the desired solution size. These two algorithms are modified version of given algorithms in (Grandoni, 2004).

Algorithm 4.4 VC1 Algorithm

```
function VC1($G(V, E), k$)
    for all $v : \delta_v = 0, v \in V$ do
        $G(V) \leftarrow G(v) -\{ v\}$
    end for
    if $G(E) = \emptyset$ then return $\emptyset$
    end if
    if $k = 0$ then return $\aleph$
    end if
    Select an $e = (u, v)$
    $m_1 \leftarrow VC1(G(V -\{ u\}), k - 1))$
    $m_2 \leftarrow VC1(G(V -\{ v\}), k - 1))$
    $m_1 \leftarrow m_1 \cup \{u\}$
    $m_2 \leftarrow m_2 \cup \{v\}$
    if min($|m_1|, |m_2|$) $\leq k$ then
        if $|m_1| \leq |m_2|$ then return $m_1$
        else return $m_2$
        end if
    else return $\aleph$
    end if
end function
```

VC1 algorithm that is shown in Algorithm 4.4 finds vertex cover solution by using a straightforward and useful method. Algorithm VC1 is based on a simple observation. For any edge $e = (u, v)$ of G, every vertex cover must contain at least one end-point of the edge. Thus, it can be branched by including u or v in the vertex cover. The algorithm takes $G(V, E)$ and k as arguments and returns the vertex cover set. When the algorithm starts execution, firstly removes vertices that have $\delta_v = 0$ from G. The algorithm terminates when there is no edge in the given graph or $k = 0$. The algorithm arbitrarily selects an edge $e = (u, v)$ and reduces graphs as $V \setminus \{u\}$ and $V \setminus \{v\}$, then call itself twice with those two reduced graph and $k - 1$. After both callee functions are return m_1 and m_2, caller function returns the smaller of the $m_1 \cup \{u\}$ and $m_2 \cup \{v\}$ sets.

Algorithm 4.5 VC2 Algorithm

```
function VC2($G(V, E), k$)
    for all $v : \delta_v = 0, v \in V$ do
        $G(V) \leftarrow G(v) -\{ v\}$
    end for
    if $G(E) = \emptyset$ then return $\emptyset$
    end if
    if $k = 0$ then return $\aleph$
    end if
    if $\exists v : N(v) = w$ then
        $m \leftarrow VC2(G(V -\{ w\}), k - 1))$
        $m \leftarrow m \cup \{w\}$
        if $|m| \leq k$ then return $m$
        else   return $\aleph$
        end if
    end if
    Select random vertex $v$
    $m_1 \leftarrow VC2(G(V - \{v\}), k - 1))$
    $m_2 \leftarrow VC2(G(V - N(v)), k -| N(v)|))$
    $m_1 \leftarrow \{v\}$
    $m_2 \leftarrow N(v)$
    if min($|m_1|, |m_2|) \leq k$ then
        if $|m_1| \leq |m_2|$ then return $m_1$
        else   return $m_2$
        end if
    else   return $\aleph$
    end if
end function
```

Theorem 4.5. Time complexity of Algorithm 4.5 is $\mathcal{O}(2^k n)$.

Proof. The recursion formula of VC1 algorithm is:

$$T(n,k) \leq \begin{cases} c \times n, & \text{if } k = 1 \\ 2 \times T(n,k-1) + c \times k \times n, & \text{if } k > 1 \end{cases} \tag{4.1}$$

Equation 4.1 could be solved with proof by induction.

Base case :

There is a real number c that satisfies $T(n,1) \leq c \times n \leq 2 \times n$ when $k = 1$.

Induction step: We assume that Equation 6.3 is correct for $k - 1$, and show it is also true for k.

$$\begin{aligned} T(n,k) &\leq 2 \times T(n,k-1) + c \times k \times n \\ &\leq 2 \times c \times 2^{k-1} \times (k-1) \times n + c \times k \times n \\ &\leq c \times 2^k \times n \times (k-1) + c \times k \times n \\ &= c \times 2^k \times k \times n - c \times 2^k \times n + c \times k \times n \\ &\leq c \times 2^k \times k \times n \end{aligned} \tag{4.2}$$

Equation 4.2 concludes that the time complexity of Algorithm 4.4 is $\mathcal{O}(2^k n)$.
□

In Algorithm 4.5, it is seen that the improved version of Algorithm 4.4. Algorithm 4.5 checks if there exists a vertex v which has only one neighbor w. If so, the algorithm reduces G as $G(V - w)$ and calls itself with reduced graph and $k - 1$ as seen in Line 10. VC2 algorithm branches over reduced graphs $G(V \setminus v)$ and $G(V - N(v))$ instead of branching both end points of randomly selected $e = (u,v)$ as is in Algorithm 4.4.

Theorem 4.6. Time complexity of Algorithm 4.5 is $\mathcal{O}(1.62^k n^2)$

Proof. Assume a graph G which does not have any vertex such that its degree is 1. In such a graph, the algorithm runs between lines 11 and 19 and produces a recursion formula as following:

$$T(k) \leq T(k-1) + T(k - \mid N(v) \mid) \tag{4.3}$$

Table 4.1. Theoretical comparison of sequential vertex cover algorithms.

Algorithm	**Time Complexity**	**Approximation**
2-approximation	$O(m)$	2
Greedy	$O(m)$	$O(\log(n))$
VC1	$O(2^k kn)$	Exact
VC2	$O(1.62^k n^2)$	Exact

According to our assumption, $|N(V)|$ should be greater than 1. Then, Equation 4.3 turns out $T(k) \leq T(k-1) + T(k-2)$. The characteristic function of this recursion formula is $r^k + r^{k-1} + r^{k-2} = 0$. The positive root of the characteristic function $x \approx 1.62$. Namely, the algorithm calls itself at most 1.62^k times (creates sub-problems) depending on the variable k. The solving cost of a problem by excluding the corresponding sub-problem takes 2^n. Therefore, the complexity of Algorithm 4.5 is $\mathcal{O}(1.62^k n^2)$. □

For a detailed analysis of algorithms VC1 and VC2 please refer (Grandoni, 2004).

Table 4.1 shows the time complexities and approximation ratios of the aforementioned algorithms.

4.2 Literature Review of Distributed Vertex Cover Algorithms

Parnas and Ron have proposed an algorithm for VC problem using reduction (Parnas and Ron, 2007). In each round, the nodes add themselves to VC, if their degree is greater than $\Delta/2^R$ where Δ is the maximum degree of the graph and R is the current round of algorithm. Other nodes send $DROP$ messages to their neighbors. The approximation ratio of this algorithm is $2(\log_2(\Delta)+1)$. Algorithms terminates in $\log(\Delta)$ round by sending at most sending $m = n^2$ message for a complete graph instance.

Hoepman's simple distributed weighted matching algorithm (Hoepman, 2004) has been adapted by Kavalci et al. for VC problem (Kavalci et al., 2014). In this algorithm, each node selects a node v with maximum id from $N(v)$ that have the minimum degree and sends *PROPOSE* message to the selected node v. When node v receives *PROPOSE* message from u, node v include itself VC if they send *PROPOSE* to each other and $\delta_v > \delta_u$ in the first round. The condition $\delta_v > \delta_u$ is not required in the other rounds. We can find the maximal matching on the bipartite graph after at most Δ steps. Hancowiak et. al. has proposed a distributed matching algorithm using this fact with $\mathcal{O}(n)$ time complexity (Hanckowiak et al., 2001). Polishchuk and Suomela has exploited Hancowiak's algorithm to find a VC with 3-approximation ratio (Polishchuk and Suomela, 2009). In their proposed approach the graph $G(E, V)$ is converted to an bipartite graph as $\widetilde{G}(\widetilde{V_1}, \widetilde{V_2}, \widetilde{E})$ such that:

- each vertex $v \in V$ is replaced by $\widetilde{v_1} \in \widetilde{V_1}$ and $\widetilde{v_2} \in \widetilde{V_2}$.
- The edge u, v is replaced by two edges as $(\widetilde{u_1}, \widetilde{v_2}) \in \widetilde{E}$ and $(\widetilde{u_2}, \widetilde{v_1}) \in \widetilde{E}$.

This provides two copies for each vertex in $G(V, E)$ (white and black copies). For the edge between u and v, the white copy of u is connected to the black copy of v and the black copy of u is connected to the white copy of v. After computing maximal matching in this converted graph, the minimal VC will be the white or black copies of the nodes (Erciyes, 2013; Ileri et al., 2016a). Matching based algorithms cover one edge through two nodes unnecessarily. Hence, these algorithms can not exceed 2-approximation ratio for the VC problem as seen in Figure 4.1a.

Kavalci et al. (Kavalci et al., 2014) has proposed an algorithm that uses BFST to form a VC. The algorithm asynchronously constructs a BFST on a given graph with unicast messages and then covers the vertices according to their level. If a node is in even-numbered level, it is added to VC directly. Otherwise, they are selected if they have a neighbor with a lower id . The algorithm may cover all nodes in an odd-numbered level except one that has the minimum id. The algorithm produces a vertex solution at most n round by sending n^3 messages in total.

Yigit has proposed two novel algorithms which integrate BFS and VC like Kavalci's algorithm (Yigit, 2016). Unlike the Kavalci's algorithm's, these algorithms run the greedy approach to cover the uncovered layers of the structured BFS tree. The first proposed algorithm covers even-numbered layers directly and covers odd-numbered layers by using the greedy technique. The second one is to look first into which layer (odd or even-numbered) contains more cross-edges, then decides to the layer will be covered. These algorithms are called VECO1 and VECO2, respectively. Both algorithms execute in n round at the worst case by transmitting n^2 message on all over the network.

We provide a summary of the distributed vertex cover algorithms in Table 4.2 for a better understanding. We also recommend İleri et. al's survey which deeply elaborates vertex cover algorithms in wider aspects (Ileri et al., 2016b).

Table 4.2. Theoretical comparison of distributed vertex cover algorithms.

Algorithm	**Approx. Ratio**	**Time Comp.**	**Msg. Comp.**	**Space Comp.**
VECO1	-	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(\Delta\log(n))$
VECO2	-	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(\Delta\log(n))$
Kavalci	-	$\mathcal{O}(n)$	$\mathcal{O}(n^3)$	$\mathcal{O}(\Delta\log(n))$
Parnas	$2(\log_2(\Delta) +1)$	$\log(n^2)$	$\mathcal{O}(\Delta)$	$\mathcal{O}(\Delta\log(n))$
Greedy	$\log(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(\Delta\log(n))$
Hoepman	2	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(\Delta\log(n))$
Polishchuk	3	$2\Delta +1$	4m	$\mathcal{O}(\Delta\log(n))$

4.3 Literature Review of Capacitated Vertex Cover Algorithms

The capacitated vertex cover problem has been introduced by Guha et al. (Guha et al., 2003). In this version of the problem, nodes can exist more than one time in the vertex cover solution. This version of the problem is called as *soft* capacitated vertex cover. Moreover, in this study, each vertex has a weight represented as w_v. The integer programming formulation proposed by Guha et al. is seen in Equation 7.1. Guha et al. have proposed a simple algorithm to solve the capacitated vertex cover problem. Firstly, the relaxed version which y_{ev} and x_v are fractional is solved. After then, each $y_{ev} >= \frac{1}{2}\ v \in V$ rounded to new variable y^*_{ev} as 1, otherwise $y_{ev} = 0$. Now, it can be derived $x^*_v = \lceil \frac{\sum_{e \in E(v)} y^*_{ev}}{k_v} \rceil$ from y^*_{ev} where $E(v)$ represents incident edges to v. The approximation ratio of this simple algorithm is 4. This means that $x^*_v \leq 4 \times x_v$.

$$\begin{aligned}
\text{min} \quad & \sum_{v \in V} w_v \times x_v \\
\text{s.t.} \quad y_{eu} + y_{ev} &\geq 1 \qquad e = (u,v) \in E \\
k_v \times x_v - \sum_{e \in E(v)} y_{ev} &\geq 0 \qquad v \in V \\
x_v &\leq y_{ev} \qquad v \in e \in E \\
y_{ev} &\in 0,1 \qquad v \in e \in E \\
x_v &\in \mathbb{N}_0
\end{aligned} \tag{7.1}$$

Chuzhoy and Naor have proposed two randomized rounding algorithms to solve the problem (Chuzhoy and , Seffi). In this version of the problem, each node has a bound variable $b(v)$ which restricts the number of contributions of a node in

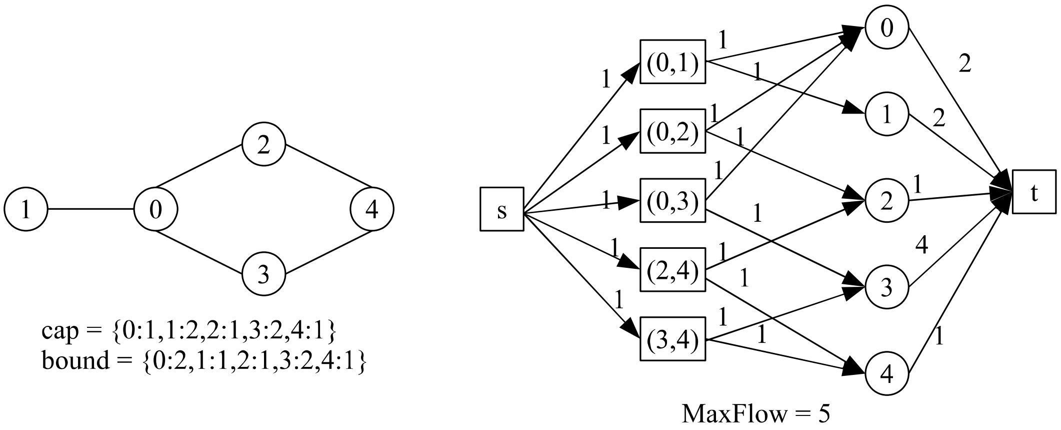


Figure 4.2. Pre-processing step that uses max-flow algorithm to detect configuration is feasible.

the vertex cover set. Nodes in this version of the linear programming formulation are unit weighted. As a first algorithm, the authors put forward a simple randomized rounding algorithm that has 8-approximation ratio in the worst case. After this algorithm, the author introduces a 3-approximation randomized rounding algorithm to solve the problem. The main difference between these two algorithms is their limit value to include a vertex in the cover set. After the linear programming solution, the first algorithm includes vertices that have $x_v \geq \frac{1}{2}$ while the second one includes vertices that have $x_v \geq \frac{1}{3}$. Now each algorithm applies its randomized rounding and alteration process.

Lastly, Gandhi et al. have proposed an algorithm which is the same as Chuzhoy's algorithm but has main 2 differences (Gandhi et al., 2003). There is a pre-processing step before the linear programming which focuses on eliminating capacity-1 vertices from the solution set. For this pre-processing step, max-flow is used to maintain the feasibility of a given configuration. Figure 4.2 depicts pre-processing on a sample graph, the explanation to which is given following. $G' = (E, V, F)$ is a bipartite graph in which the left-hand side of the bipartite is E and right-hand side is V of given graph $G(V, E)$. An edge $(e, v) \in F$ if and only e is touches to v. Construct a flow network in which the source is connected to all vertices in E and each vertex in V is connected to the sink. The capacities of the edges in F is 1. The capacities of the edges emanating from the source are all 1. The capacity of an edge from any node $v \in V$ to the sink is cap_v. G has a feasible solution if the maximum flow value from the source to the sink is $|E|$. After a linear programming solution, the algorithm includes vertices that have $x_v \geq \frac{1}{2}$, and makes randomization and alteration steps. At the end of the algorithm, y^* is created by the max-flow algorithm which is used in the pre-processing step. The approximation ratio of the algorithm is 2.

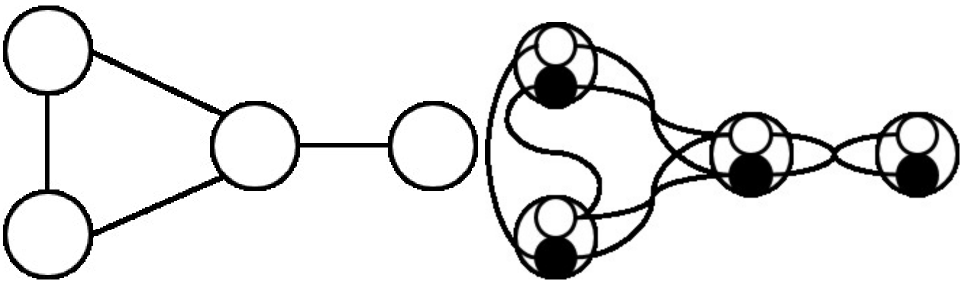
Table 4.3. Theoretical comparison of capacitated vertex cover algorithms.

Algorithm	Technique	Approximation Ratio
(Guha et al., 2003)	Rounding	4
(Chuzhoy and , Seffi)-1	Randomized Rounding	8
(Chuzhoy and , Seffi)-2	Randomized Rounding	3
(Gandhi et al., 2003)	Randomized Rounding	2

In Table 4.3 we provide a summary of the capacitated vertex cover algorithms.

4.4 Literature Review of Self-Stabilizing Vertex Cover Algorithms

In self-stabilizing settings, the first algorithm for the vertex cover problem has been proposed by Kiniwa (Kiniwa, 2005). Kiniwa's algorithm firstly constructs a maximal matching by favoring the edges connecting the heaviest nodes with the lightest nodes on the graph and then covers the nodes on the basis of this matching. Each vertex holds $cover$ and $color$ variables which describe whether the vertex in vertex cover set and color of the matched port respectively. Also, each vertex maintains three sets which are named *High, Low* and *Others*. The $High(v)$ set contains neighbors of vertex v which have bigger $color$ value. On the contrary, $Low(v)$ contains neighbors of node v which have smaller $color$ value. $Others(v)$ holds neighbors that do not point to v. The algorithm obtains $(2 - \frac{1}{\Delta})$-approximation vertex cover using the shared memory and distributed scheduler in $M + 2$ round where M is the size of the matching.



(a) Sample graph G.

(b) Kronecker graph $\mathcal{K}(G)$ of G

Figure 4.3. Production of Kronecker graph from graph G

Turau has proposed two algorithms based on the distributed approximation VC

Algorithm 4.6 Turau's first self-stabilizing vertex cover algorithm.

```
R1: $\neg whiteMatched(v) \land (v.white \neq v \lor v.white \neq selectBlack(v))$
$\Rightarrow$ if $v.white \neq v$
        $v.white \leftarrow v$
    else
        $v.white \leftarrow selectBlack(v)$
R2: $\neg blackMatched(v) \land (v.black = v \lor v.black.white \neq v.black) \land$
        $v.black \neq selectWhite(v)$
    $\Rightarrow v.black \leftarrow selectWhite(v)$
```

algorithm in (Turau and Hauck, 2011). Turau's algorithms run under the distributed unfair scheduler and link-register communication model on an anonymous network. Turau's algorithms exploit Hancowiak port numbering method on *kronecker* double graph (Hanckowiak et al., 2001). Kronecker double graph is a graph that holds two copies of each vertex as shown in Figure 4.3. Both algorithms are based on the (Polishchuk and Suomela, 2009) which cannot exceed 3 approximation by using the matching method on the anonymous networks proved by (Angluin, 1980).

Turau firstly proposes a basic algorithm that will be base of the second algorithm. The basic algorithm consists of two predicates that show the vertex v connected whether black or white copy. These predicates are shown in Equation 4.4 and Equation 4.5. The complete algorithm is shown in Algorithm 4.6 as well. The basic algorithm calculates 3-approximation ratio vertex cover set with $O(n + m)$ moves, where n and m are the number of vertices and edges, respectively.

$$blackMatched(v) \equiv v \neq null \land v.black.white = v \tag{4.4}$$

$$whiteMatched(v) \equiv v \neq null \land v.white.black = v \tag{4.5}$$

In addition to these predicates, each node use two function to select neighbor for matching. Function $selectBlack(v)$ shown in Equation 4.6 (4.7) ($selectWhite(v)$) returns the neighbor x which shows vertex v with its black pointer (white pointer).

$$selectBlack(v) \equiv v.select\{x \in N(v) | x.black = v\} \tag{4.6}$$

$$selectWhite(v) \equiv v.select\{x \in N(v) | x.white = v\} \tag{4.7}$$

In the same paper, Turau presents an improvement method which finds $(3 - \frac{2}{\Delta+1})$-approximation a vertex cover in $O(n + m)$ moves. This improvement algorithm contains the first basic algorithm with additional rules. After the execution of two basic rules, some nodes may still be un-matched with both pointer. The algorithm excludes this type of vertices from the vertex cover solution by using Equation 4.8.

$$\begin{aligned} &candidate(v) \equiv \\ &((blackMatched(v) \lor v.black = null) \land v.white = null) \lor \\ &((whiteMatched(v) \lor v.white = null) \land v.black = null) \end{aligned} \tag{4.8}$$

Independent set problem is another well-known graph-theoretical problem which has a strong connection with VC. In a graph $G(V, E)$, $S \subset V$ is an independent set if there is not adjacent nodes in S. Literally, if $S \subseteq V : \neg(u \in S \land v \in S)$ $\forall e = (u, v) \in E$. If no extra node can be added to S without violating the independent set condition, S is *maximal*. The problem of finding the IS with the maximum IS having the maximum possible cardinality is called *maximum independent set*. Every solution S to IS problem gives a feasible solution C to VC problem such that $C = V \backslash S$. Even though these two problems have different approximation behaviors and an approximation to one of the problems does not guarantee an approximation to the other (Dinur and Safra, 2005), we can use IS algorithms to find *feasible* solutions to VC. Thus, we include self-stabilizing independent set algorithms in the scope of our work.

The first self-stabilizing maximal independent set algorithm has been presented by Shukla et al. (Shukla et al., 1995). It assumes a central scheduler and anonymous network. A node joins S (updates its state to `IN`) if it has no neighbors in S, and leaves S (updates its state to `OUT`) if at least one of its neighbors is in S. Ikeda et al. have proposed an algorithm that copes with a distributed scheduler (Ikeda et al., 2002). It assumes the nodes have unique identifiers for symmetry breaking and a node can only leave S if it has a neighbor in S with a lower *id*. Goddard et al. have presented the first algorithm that guarantees to stabilize under synchronous scheduler. A node can only join S if it has the lowest *id* among `OUT` neighbors, and can only leave S if it has the highest *id* among `IN` neighbors.

Turau has presented the first distributed self-stabilizing algorithm for IS that requires a linear number of moves using an unfair distributed scheduler(Turau, 2007). It introduces a new intermediate state called `WAIT`. If a node has no neighbors in S, it changes its state to `WAIT`. A node in `WAIT` state updates becomes `OUT` if it has some at least one `IN` neighbors, or it becomes `IN` if it has no neighbor that has a lower

Table 4.4. Theoretical comparison of self-stabilizing vertex cover algorithms.

Problem	**Algorithm**	**Topology**	**Scheduler**	**Move Comp.**	**Round Comp.**	**Approx. Ratio**
VC	(Turau and Hauck, 2011)	Anonymous	Adversarial	$O(n+m)$	-	3
	(Turau and Hauck, 2011)	Anonymous	Adversarial	$O(n+m)$	$O(\Delta)$	$3-\frac{2}{\Delta+1}$
	(Kiniwa, 2005)	Unique ID	Distributed, fair	-	$M+2$	$2-\frac{1}{\Delta}$
MIS	(Shukla et al., 1995)	Anonymous	Central	$O(n)$	-	-
	(Ikeda et al., 2002)	Unique ID	Adversarial	$O(n^2)$	-	-
	(Goddard et al., 2003)	Unique ID	Synchronous	-	$O(n)$	-
	(Turau, 2007)	Unique ID	Adversarial	$O(n)$	-	-

identifier.

Vertex cover and independent set solutions complement each other on a given graph *G(V,E)*. Based on this fact, the performance evaluation of the vertex cover and independent set algorithms was examined in (Yigit et al., 2018) on the self-stabilizing setting. The study shows that self-stabilizing independent set algorithms are faster and better than matching-based self-stabilizing vertex cover algorithms.

In Table 4.4, we provide a summary of self-stabilizing VC and MIS algorithms. We refer readers to (Ileri et al., 2016b) and (Guellati and Kheddouci, 2010) for more detailed reviews on self-stabilizing versions of these problems.

5. IMPLEMENTATION of VERTEX COVER ALGORITHMS

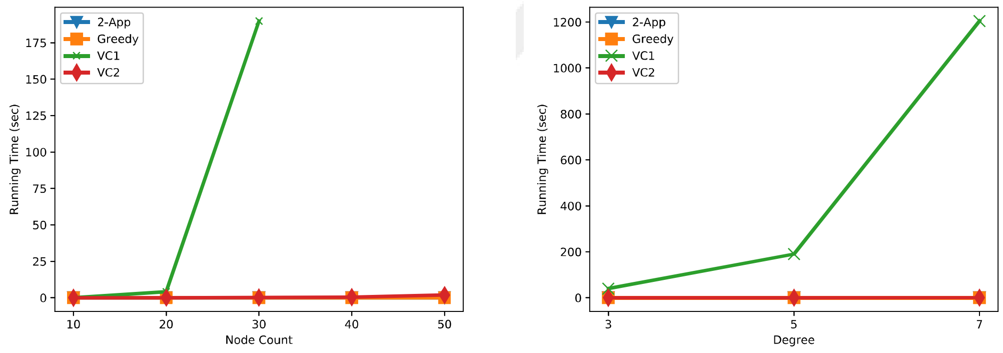
5.1 Performance Evaluation of Sequential Vertex Cover Algorithms

In this section, two optimal algorithms are compared with one heuristic and one approximation algorithm in terms of running time and cardinality of the vertex cover set.

5.1.1 Simulation Environment

Algorithms are executed on randomly created graphs of varying sizes between 10-50. Each graph has 3 different density types, which are determined by the average degree which could be 3, 5, or 7. For each type of graph, we construct 10 different topologies, which gives 150 different scenarios to clearly compare algorithms. Algorithms were implemented with Python 2.7 programming language and networkX 1.10 library was used to create topologies (Schult and Swart, 2008).

5.1.2 Running Times of Algorithms



(a) Running times of algorithms by graph size (avg. degree 5).

(b) Running times of algorithms by average degree (graph size 30).

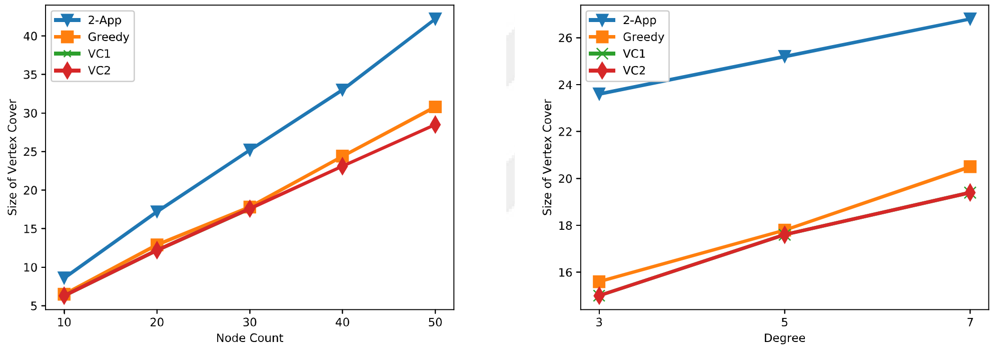
Figure 5.1. Comparisons of algorithms in terms of running time.

Figure 5.1a shows the execution times of algorithms on different sized graphs which fixed on average degree 5. VC1 algorithm takes a big step with 180 seconds running time on graphs with size 30 where the VC1 algorithm was not accounted for after this size. Algorithm VC2 produces a feasible solution even if it is a non-polynomial algorithm until the size of graphs reaches 50. However, after graph size 50, it is not possible to measure the running time of VC2 algorithm on our setup. Both Greedy and 2-approximation algorithms produce better results than both optimal algorithms since they have polynomial time complexity $\mathcal{O}(m)$. On 50 sized graphs, execution of 2-approximation, Greedy, and VC2 took 94×10^{-5}, 12×10^{-4},

$and1.8$ seconds, respectively. Since the 2-approximation algorithm puts two vertices at every iteration, it terminates faster than the Greedy algorithm which selects only one vertex at an iteration.

Running time comparison of algorithms on 30 sized graphs depending on the average degree is shown 5.1b. VC1 algorithm is more affected algorithm by the increasing of the density of the graph since its execution time jumps up to 20 minutes as the graphs get denser. In spite of being affected by the density of the graph, it gives feasible solutions when compared to VC1. Both polynomial algorithms show better and faster running time while they are not affected by graph density.

5.1.3 Cardinality of Vertex Cover Set



(a) VC size of algorithms by graph size (avg. degree 5).

(b) VC size of algorithms by average degree (graph size 30).

Figure 5.2. Comparison of algorithms in terms of VC set size.

Figure 5.2a shows the size of vertex cover sets produced by each algorithm in the case of increasing graph size. Algorithms VC1 and VC2 give the best result against Greedy and 2-approximation algorithms because they are optimal algorithms. As shown in the figure, the Greedy algorithm produces the closest result to both optimal algorithms VC1 and VC2. 2-approximation algorithm does not show the same performance which has shown in the execution time. Since the 2-approximation algorithm almost selects all vertices in set V, it gives the worst vertex cover solutions among all implemented algorithms.

Vertex cover size by increasing the average degree of graphs is shown in 5.2b. All algorithms are affected by the density of the graph since the density causes more edges to cover. For 30 sized graphs VC1, VC2, Greedy and 2-approximation produce VC solutions in size 15, 17.6 19.4, respectively. Although Greedy is a polynomial-time algorithm which does neither ensure the optimal solution nor have a constant approximation ratio, it produces feasible solutions as VC1 and VC2 produce. It is observed to give optimal solutions to some scenarios during the simulation. As expected 2-approximation algorithm produces the largest VC solutions which almost the same size as V.

We provide a summary of the charts as a table in Table 5.1 to reveal the performance of the algorithm on all graph types.

Table 5.1. Performance metrics of the implemented algorithms for all scenarios.

			Time			**VC**	
Algorithm	**Size**	3 Degree	5 Degree	7 Degree	3 Degree	5 Degree	7 Degree
Approximation	10	1.09×10^{-4}	1.10×10^{-4}	1.29×10^{-4}	8.4	8.6	9.6
	20	2.25×10^{-4}	2.49×10^{-4}	2.87×10^{-4}	15.6	17.2	17.8
	30	3.89×10^{-4}	4.49×10^{-4}	4.89×10^{-4}	23.6	25.2	26.8
	40	5.91×10^{-4}	6.22×10^{-4}	7.42×10^{-4}	31.2	33.0	35.0
	50	8.45×10^{-4}	9.38×10^{-4}	1.06×10^{-3}	39.2	42.2	43.8
Greedy	10	1.28×10^{-4}	1.38×10^{-4}	1.50×10^{-4}	5.7	6.5	7.5
	20	2.79×10^{-4}	3.17×10^{-4}	3.48×10^{-4}	10.3	12.9	13.5
	30	5.08×10^{-4}	5.49×10^{-4}	5.85×10^{-4}	15.6	17.8	20.5
	40	7.77×10^{-4}	8.30×10^{-4}	9.12×10^{-4}	20.8	24.4	27.3
	50	1.10×10^{-3}	1.19×10^{-3}	1.26×10^{-3}	25.3	30.8	33.4
VC1	10	2.16×10^{-2}	4.12×10^{-2}	7.45×10^{-2}	5.4	6.3	7.3
	20	8.52×10^{-1}	4.17	7.04	10.0	12.2	13.4
	30	3.98×10	1.90*10	1.20*10	15.0	17.6	19.4
	40	-	-	-	-	-	-
	50	-	-	-	-	-	-
VC2	10	1.94×10^{-3}	4.69×10^{-3}	4.87×10^{-3}	5.4	6.3	7.3
	20	$4.90*10^{-3}$	3.56×10^{-2}	4.44×10^{-2}	10.0	12.2	13.4
	30	1.29×10^{-2}	$1.30*10^{-1}$	2.61×10^{-1}	15.0	17.6	19.4
	40	2.66×10^{-2}	4.21×10^{-1}	1.42	19.7	23.1	26.2
	50	3.57×10^{-2}	1.88	8.26	24.7	28.5	32.2

5.2 Performance Evaluation And Test-Bed Experiment of Distributed Vertex Cover Algorithms

In this section, we implement distributed vertex cover algorithms in simulator and test-bed to compare their results. Firstly, we analyze time, space and bit complexity of these algorithms. After that, we provide simulation and test-bed evaluations on the TOSSIM simulation environment and IRIS motes.

We implemented the Greedy algorithm, Parnas and Ron algorithm, Kavalci algorithm and Hoepman algorithm and two VECO algorithms in terms of performance metrics which are received bytes, sent bytes, energy consumption, running

Table 5.2. Real experiments parameters.

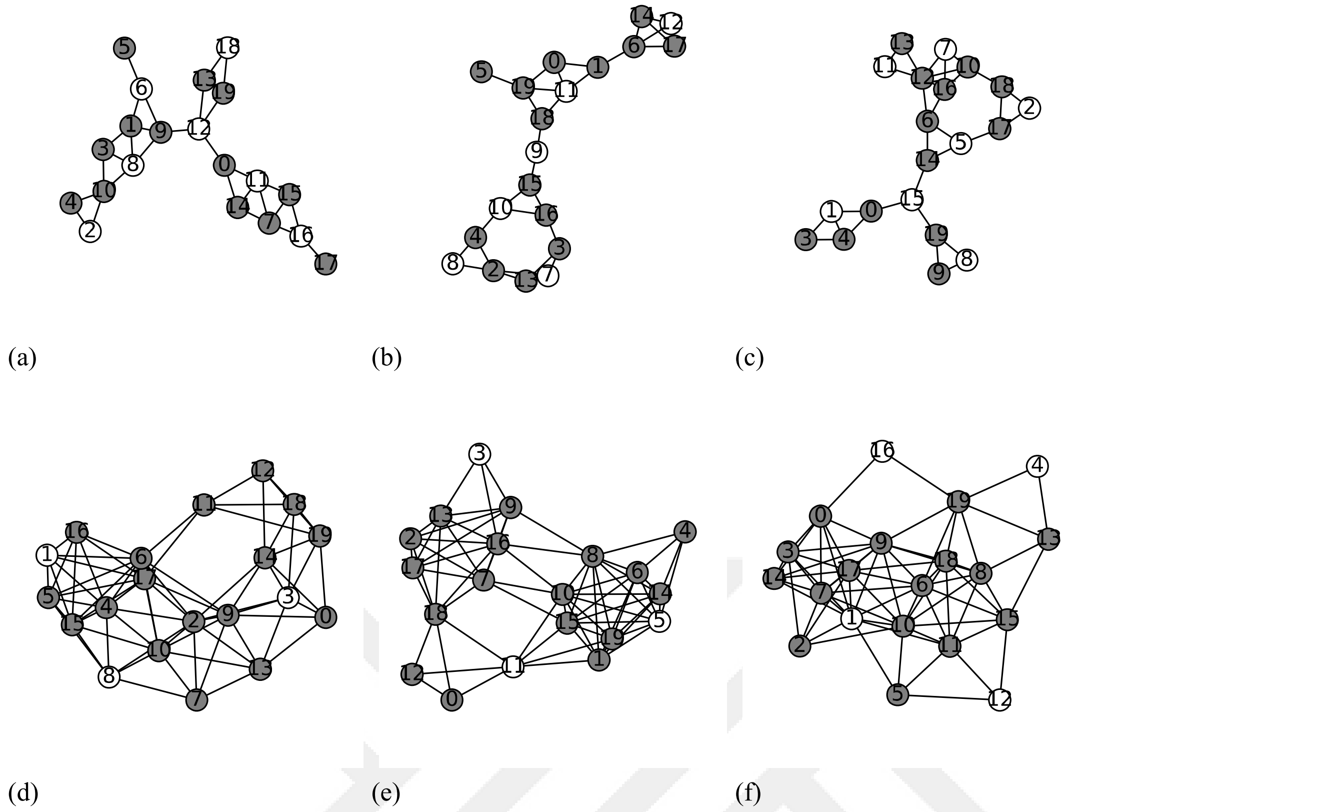
Mote	IRIS
Platform	TinyOS 2.1.2
Transceiver	2.4 GHz, IEEE 802.15.4 compatible
Transmission Rate	250 kbps
Transmission Power	3 dBm
Number of Nodes	20
MAC	TDMA
Transmission Range	50 m
Node Degrees	3 and 7

time and the performance of vertex cover size. To provide fairness in performance evaluation, the cost of BFS tree construction is added to the results of algorithms which are not based on the BFS tree.

5.2.1 Test-bed Experiments

We conduct real experiment on IRIS motes in order to compare the performance of distributed vertex cover algorithms. We implement the algorithms on TinyOS operating system (Levis et al., 2003) with nesC programming language that runs on IRIS motes. The IRIS is a 2.4 GHz mote used for wireless sensor networks that need low power consumption. IRIS motes have 250 kbps data transmission rate, 128 Kb programmable flash memory, 8 Kb ram, and 3 dBm transmission power that enable a maximum 50 m indoor transmission range (Memsic, 2011). We reduced the transmission power of IRIS motes, to establish the desired networks in a laboratory environment. Due to the different characteristic of the radio range of each mote, some link becomes unidirectional which is an undesirable situation for the reliable communication in wireless sensor networks. Each node sends its list of neighbors after receives *HELLO* message and thus unidirectional links are ignored by nodes. This setup phase does not count as the performance of algorithms since it is the same for all algorithms. We use 20 nodes in order to achieve real experiments. The algorithms are executed 10 times on randomly generated topologies that are sparse and dense. Each node has three neighbors in sparse topologies and seven neighbors in dense topologies on average. Table 5.2 shows the sum of the simulation parameters.

Six examples of sparse and dense topologies used in real practice are shown in Figure 5.3. While Figures 5.3a, 5.3b, 5.3c show sparse topology, dense topologies are shown in Figures 5.3d, 5.3e, 5.3f. Gray nodes are the vertex cover solution



(a) (b) (c)

(d) (e) (f)

Figure 5.3. Six sample topologies for real experiments.

produced by the VECO1 algorithm. Deployment of IRIS motes is seen in Figure 5.4.

Firstly, the algorithms are evaluated on sparse topology. VECO1 produces the best results for all performance parameters but VC size. The VECO1 algorithm produces 1.6 times better results than Hoepman and Greedy algorithms in terms of sent bytes. In addition, the received bytes of the VECO1 algorithm are 1.5 times better than the Hoepman algorithm and 2 times better than the Greedy algorithm. VECO1 and VECO2 give almost the same performance, but VECO2 needs a little bit more message traffic and system resources for sparse topologies. Kavalci algorithm needs higher message traffic and energy consumption because of unicast messages. Greedy and Parnas algorithms produce the best VC solution 12.6 and 12.7, respectively. After these two algorithms, VECO algorithms produce the third-best VC solution by selecting 13.2 vertices on average. However, we could say that energy consumption and message traffic performances eliminate the differences of VC solution. Snapshot of the monitoring program that contains sent bytes, received bytes, covered flag, and running time is seen in Figure 5.5. The sum-up of the experiment results is seen in Table 5.3. Sent bytes, received bytes, and energy

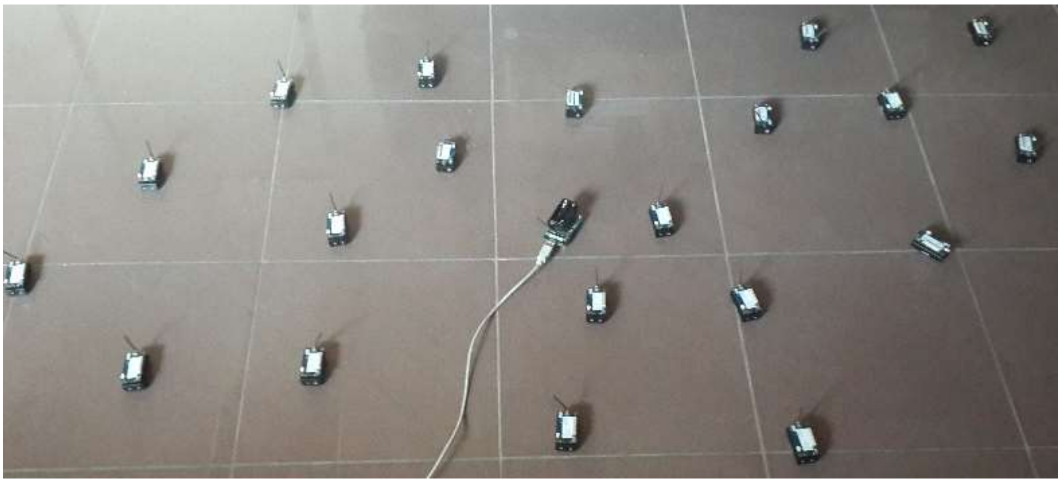


Figure 5.4. Placement of an example topology.

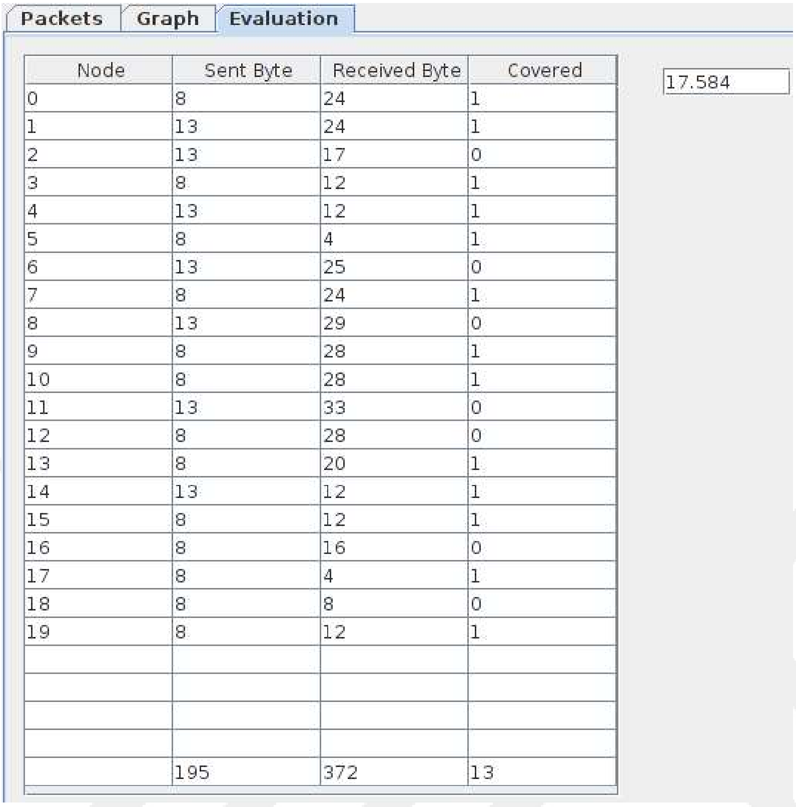
Table 5.3. Real experiment results in sparse topologies.

Algorithm	Sent Bytes	Received Bytes	Energy (mJ)	Time (s)	VC Size
VECO1	202.5	412.1	1059.8	25.1	13.2
VECO2	231.6	501.6	1263.3	30.1	13.2
Kavalci	474.0	874.0	2327.6	337.8	13.6
Parnas+BFS	205.2	500.4	1231.9	33.3	12.7
Greedy+BFS	328.8	799.5	1941.1	39.0	12.6
Hoepman+BFS	323.3	621.6	1630.7	38.3	13.6

consumption are values obtained from the entire topology.

Secondly, the algorithms are evaluated on dense topologies. In those topologies, Kavalci needs to send more messages which is 4.7 times more than VECO1 and VECO2 algorithms. Kavalci algorithm shows the worst result among the 6 algorithms and its VC size is almost equal $|V|$ of the graph size. Hoepman algorithm not only gives bad results in terms of message traffic and running time, but also the size of the VC set it produces is quite large. Parnas and Greedy have close VC size solutions, but their running time is more than the other algorithms. The running times of all algorithms are smaller than sparse topologies' since D is getting smaller for dense topologies. VECO algorithms give a feasible solution with feasible system usage.

The sum-up of the evaluation results is indicated in Table 5.4 a snapshot of the VECO2 algorithm is given in Figure 5.6.



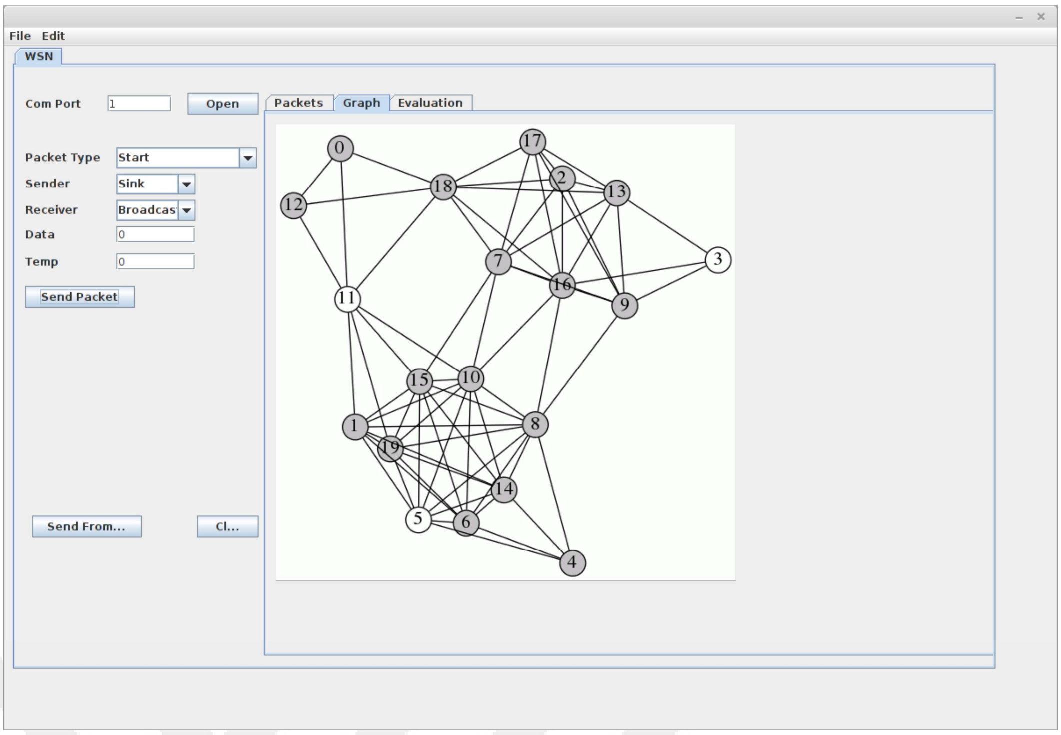
Node	Sent Byte	Received Byte	Covered
0	8	24	1
1	13	24	1
2	13	17	0
3	8	12	1
4	13	12	1
5	8	4	1
6	13	25	0
7	8	24	1
8	13	29	0
9	8	28	1
10	8	28	1
11	13	33	0
12	8	28	0
13	8	20	1
14	13	12	1
15	8	12	1
16	8	16	0
17	8	4	1
18	8	8	0
19	8	12	1
	195	372	13

17.584

Figure 5.5. Evaluation of VECO1 algorithm on a sparse topology by nodes.

Table 5.4. Real experiments result in dense topologies.

Algorithm	Sent Bytes	Received Bytes	Energy (mJ)	Time (s)	VC Size
VECO1	266.5	1165.6	2447.8	17.69	16.4
VECO2	261.2	1218.3	2527.3	22.3	15.8
Kavalci	1233.5	2433.5	6326.0	162.0	17.2
Parnas+BFST	210.4	1187.9	2384.8	27.8	15.2
Greedy+BFST	446.4	2375.3	4814.7	40.6	15.0
Hoepman+BFST	403.8	1428.2	3138.0	40.6	18.4



Com Port 1

Packet Type Start

Sender Sink

Receiver Broadcas

Data 0

Temp 0

Figure 5.6. Produced VC solution by VECO2 algorithm on a dense topology.

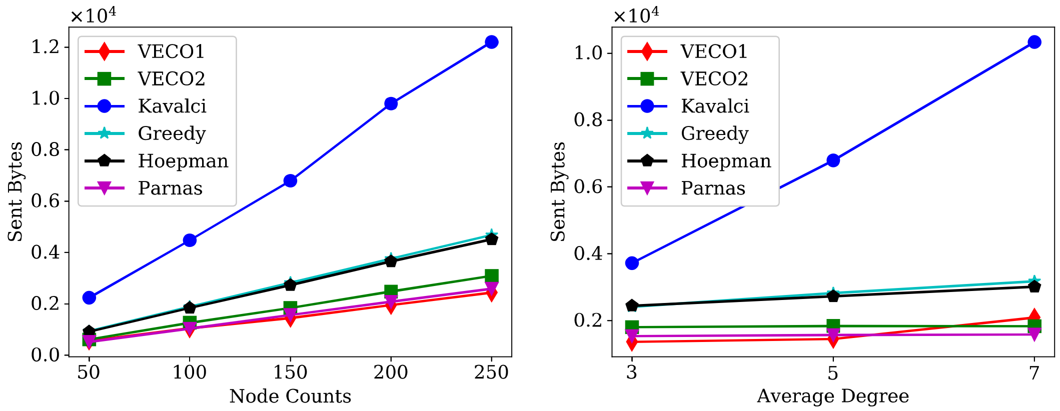
Table 5.5. Simulation Parameters

Node Distribution	Random
Sink Position	Random location in graph
Number of Nodes	50-250 (step 50)
Node Degrees	3, 5, 7
MAC	TDMA

5.2.2 Simulation Results

We conduct TOSSIM (Levis et al., 2003) simulation along with test-bed experiments because of the small number of motes. It is difficult to distinguish the difference between the implemented algorithms on 20 motes. Therefore, we create larger random topologies in order to see the behavior of algorithms.

We have generated randomly connected networks with different node counts, from 50 to 250 (step 50) nodes. In the generated topologies, the average degrees of nodes are 3, 5, and 7 and the transmission range of each node is 50 m. We measured the total the sent and received bytes, wall clock time, energy consumption, and the cardinality of the detected VC. Each measurement is the average of 10 iterations for each class of the topologies. Table 5.5 summarizes the topologies and simulation parameters. We used the simple time division multiple access (TDMA) protocol to



(a) Sent bytes of algorithms against node count. (b) Sent bytes of algorithms against node degree.

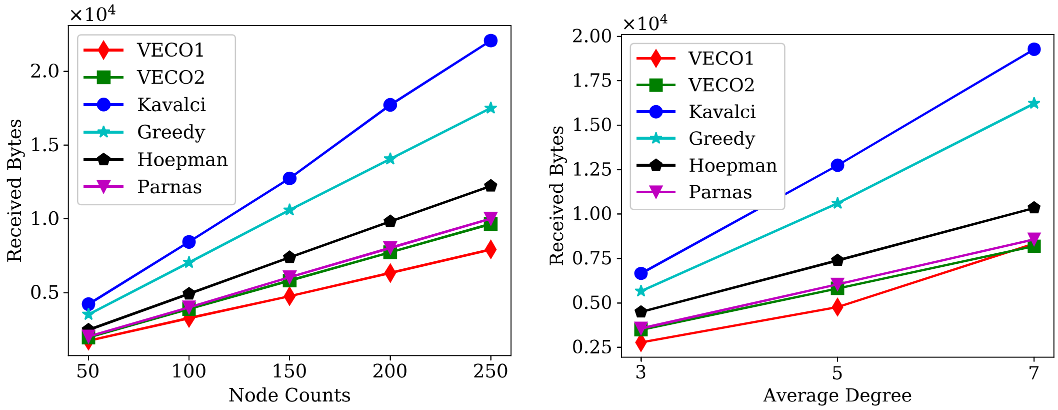
Figure 5.7. Sent bytes of algorithms.

eliminate packet interference.

A comparison of the total sent bytes of the implemented algorithms is presented in Figure 5.7a. VECO1 algorithm send the lowest amount of messages to construct BFST and VC. Parnas algorithm comes after VECO1 algorithm for all graph size. VECO2 is very close to both algorithms and needs a little bit more message to produce VC. For instance, VECO2 needs 1839 bytes in total to construct VC with BFST, while Parnas and VECO2 need approximately 1500 bytes on graph with 150 nodes. The sent bytes of VECO1 algorithm are 5, 1.9, 1.8, 1.06 times lower than Kavalci, Greedy, Hoepman and Parnas algorithm for graphs with 250 nodes.

Figure 5.7b shows the total sent bytes of algorithms against the average degree on 150 sized graphs. All algorithms are under the influence of the increase in the average degrees of the networks. Nevertheless, Kavalci is the most affected algorithm by the densities of networks. Increasing of sent bytes for Kavalci on denser topologies is stemmed from unicast messages. VECO1 algorithm sends 1364, 1450, and 2091 bytes on graphs with 3, 5, and 7 average degrees. On the other hand, the VECO2 algorithm stays around 1830 bytes for all densities. This means that selecting levels that are covered directly (odd or even), could reduce message sending for the denser graph.

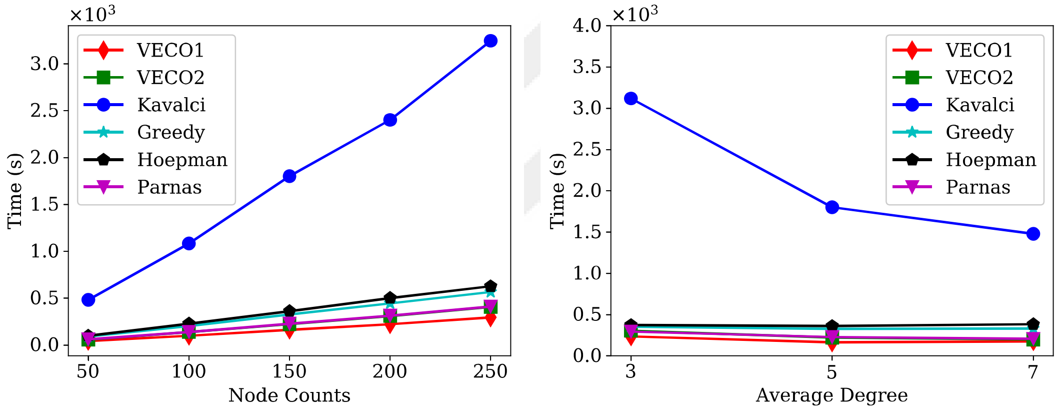
Figure 5.8a demonstrates the comparison of the total received bytes of the implemented algorithms. VECO1 and VECO2 produce the lowest received bytes among the other algorithms. VECO1 and VECO2 produce 4765 and 5817 received bytes on graphs with 250 nodes, while Parnas the nearest algorithm to these algorithms, needs 6046 bytes to produce VC with BFST. This information tells us that



(a) Received bytes of algorithms against node counts.

(b) Received bytes of algorithms against node degree.

Figure 5.8. Received bytes of algorithms.



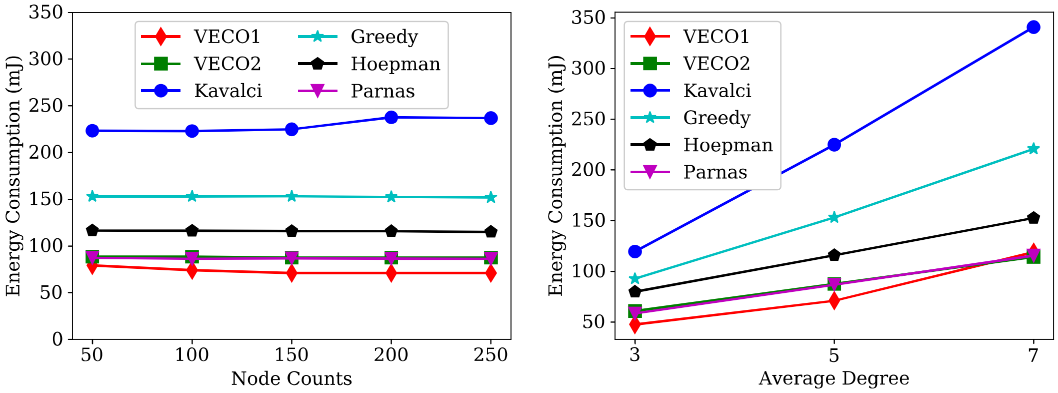
(a) Running time of algorithms against node count.

(b) Running time of algorithms against node degree.

Figure 5.9. Running time of algorithms.

VECO1 and VECO2 need 1.26 and 1.03 times lower receiving messages than Parnas. After these three algorithms, Hoepman, Greedy and Kavalci algorithms come with a tremendous amount of message traffic. For instance, Kavalci algorithm needs 12746 received bytes on 250 sized graphs.

Figure 5.8b compares the received bytes of the implemented algorithms against the average degree of each node. Kavalci algorithm is the most affected algorithm by the degree value. Increasing the average degree value increases the received bytes of Kavalci algorithm faster than other algorithms. After the Kavalci algorithm, the Greedy algorithm produces the second-worst total received bytes. The received bytes of VECO1 and VECO2 algorithms are close to each other and the degree value has a trivial impact on them. Hoepman and Parnas algorithms yield worse results than the VECO1 and VECO2 algorithms in terms of the received bytes.



(a) Energy consumption of algorithms against node count.

(b) Energy consumption of algorithms against node degree.

Figure 5.10. Energy consumption of algorithms.

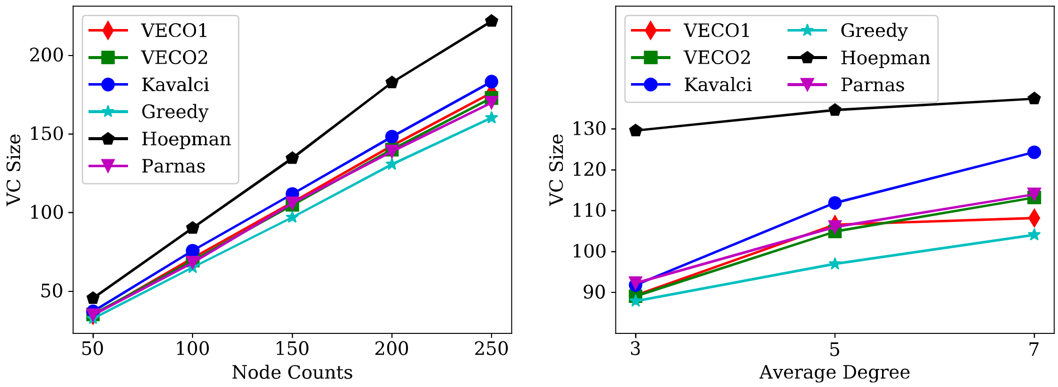
Figure 5.9a shows the comparison of the wall clock time of algorithms against the node count. VECO1 is about 1.4, 1.94, 2.14, 11 times faster than Parnas, Greedy, Hoepman, and Kavalci algorithms, respectively. After the VECO1 algorithm, VECO2 produces VC as the second fastest algorithm, namely, it is at most 1.10 faster than the Parnas algorithm. On 50 sized graphs, VECO1, VECO2, Parnas terminate in 44, 56, and 61 seconds respectively, while the Greedy algorithm needs 90 seconds to terminate on the same graph size.

The comparison of the wall clock time of algorithms against the node degree is shown in Figure 5.9b. This figure shows that when we increase the degree, the wall clock times of all algorithms decrease because of the reduction of the diameters of the graphs.

Since TOSSIM does not support calculating the energy consumption of the nodes, we used Equation 5.1 which Dagdeviren and Akram use in order to calculate the energy consumption (Dagdeviren and Akram, 2017). In Equation 5.1, S and R respectively represent sent and received bytes by the nodes.

$$E \approx ((S \times 17 + R \times 16)/31.25) \times 3.3 \quad mJ \tag{5.1}$$

The comparison of energy consumption of all algorithms against the node count is presented in Figure 5.10a. The graphics show the average individual energy consumption of a node in given networks. The VECO1, VECO2, Kavalci, Greedy, Hoepman and Parnas algorithms consume 71.0, 87.5, 224.9 153.2, 115.9, 86.8 re-



(a) VC size of algorithms against node counts.

(b) VC size of algorithms against node degree.

Figure 5.11. VC size of algorithms.

spectively on graphs with average 5 degrees with 150 nodes. All algorithms except Kavalci intend to reduce energy consumption as topologies get bigger. The energy consumptions of VECO2, Parnas algorithms are very close to each other.

The energy consumption of algorithms on graphs with 150 nodes and various degrees against the node counts are shown in Figure 5.10b. As the graphs become denser, each node needs higher energy to construct VC. VECO1 algorithm is the best algorithm on graphs with 3 and 5 average degrees with 47.6 and 71.0 mJ per node. VECO2 is the algorithm that needs to lowest energy on graphs that have 7 average degree with 114.2 mJ. Hoepman, Greedy and Kavalci algorithms consume more than 80 mJ on sparse networks and these values are increased with the average degree of the graphs. For example, the Greedy algorithm consumed 92.7 mJ on graphs that have 3 average degrees, while VECO1 and VECO2 consumed 71.0 and 87.5 mJ on graphs with 5 average degrees.

Figure 5.11a shows the comparison of the detected VC cardinality in all algorithms. The Greedy algorithm always finds smaller VCs than other algorithms. However, the cardinality of detected VCs of VECO1 and VECO1 are very close to Greedy in all topologies. Kavalci algorithm detects VC which always is larger than other algorithms except for Hoepman algorithm. Parnas and VECO algorithms produce VC solutions that have nearly the same cardinalities. Especially, VECO2 algorithm produces smaller VC than VECO1. Due to the matching technique, Hoepman algorithm produces the largest VC for all graph sizes.

Figure 5.11b shows the reactions of the algorithms when the topologies became denser on 150 sized graphs. By increasing the average degree, the cardinality of detected VC by Kavalci algorithm increases faster than the other algorithms. In sparse topologies, the Parnas algorithm gives near results to the VECO1 algorithm, but as the topologies become denser, the VECO1 and VECO2 algorithms give better results in terms of the cardinality of the detected VC. On graphs with 150 nodes and 7 average degrese, VECO1, VECO2, Parnas, and Greedy algorithms produce VC set 108.2, 113.2, 114.0, and 104.1 in size respectively. We could say that VECO algorithms produce better VC results on bigger and denser graphs.

5.3 Performance Evaluation of Capacitated Vertex Cover Algorithms

In this section, we compare linear programming capacitated vertex cover algorithm in terms of vertex cover size, execution time, and approximation ratio.

5.3.1 Experimental Setup

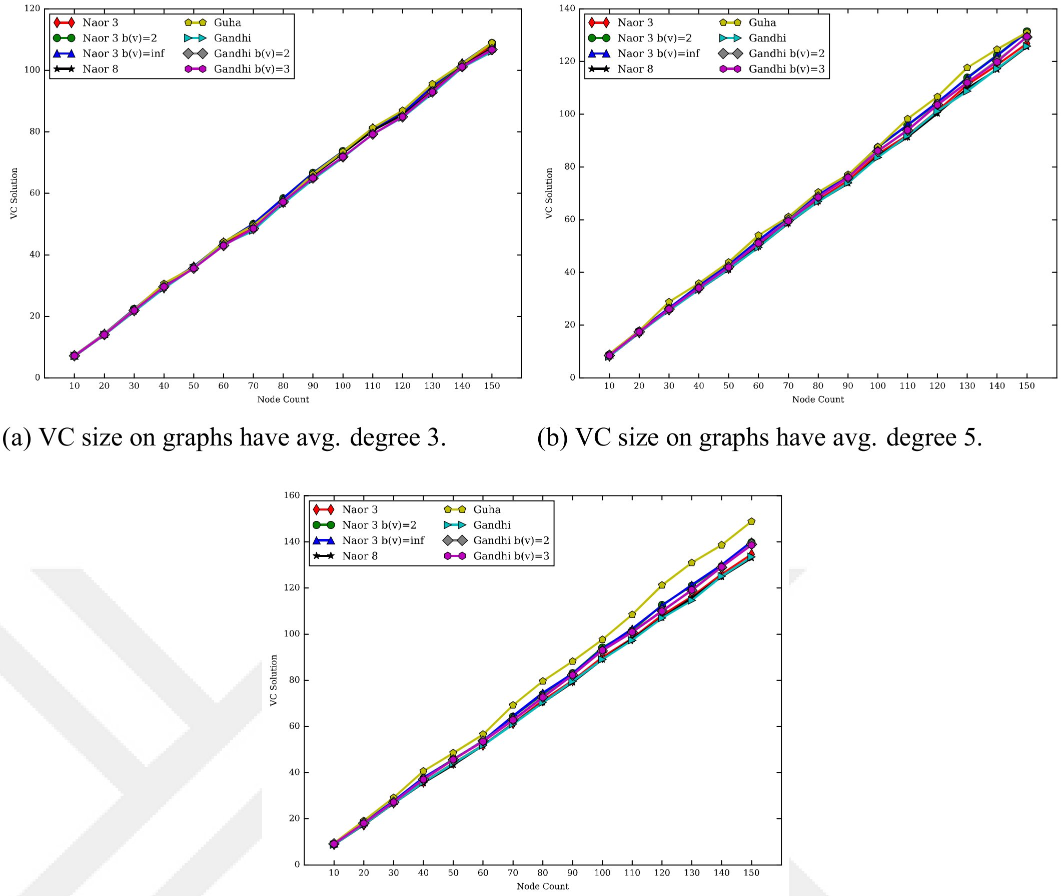
We use SageMath programming language to solve linear programming algorithms. As a solver for the mixed-integer linear program, we choose GLPK exact library at a low level. We create random geometric networks which have approximately 3, 5, 7 average degree values. We create 15 different sized topologies that have 10 to 150 nodes (with 10 steps). Therefore, we have 45 different graph topology scenarios. We run our experiment 30 random graphs for each scenario.

Guha's algorithm is proposed as a weighted algorithm, but we set all weights of the nodes to 1. Because of Chuzoy's and Gandhi's algorithms have $b(v)$ value, we run experiments on different $b(v)$ values to see the effects of bounding. For Chuzhoy's algorithm, we chose 3 different values as $b(v)$: $1, 2$ and ∞. For Gandhi's algorithm, we define 1, 2, 3 as different $b(v)$ values.

To compare these algorithms, we measure the running time, capacitated vertex cover size, and approximation ratio. Since we use different values to bound, we calculate each approximation ratio according to its setup. In order to be convinced of the fact that all graph instances provide capacitated vertex cover solution, we exploit Gandhi's max-flow method.

5.3.2 Performance Results

In this section, we investigate the performance results of all algorithms in the mentioned experimental setup.



(a) VC size on graphs have avg. degree 3.

(b) VC size on graphs have avg. degree 5.

(c) VC size on graphs have avg. degree 7.

Figure 5.12. VC size performance of algorithms.

5.3.2.1 Vertex Cover Size

Vertex cover problem which aims to find as possible as minimum vertex cover set. As introduced in 4.3, each algorithm has a different approximation ratio. In this subsection, we compare algorithms in term of vertex cover size.

Figure 5.12 shows vertex cover performance on graphs have different average degrees. As seen in Figure 5.12a the performances of the algorithms are almost the same on the sparse graphs. For example, in graphs that have 150 nodes, Gandhi's algorithm with $b(v) = 1$ produces set which has 106 nodes on average as the best result, while Naor's 3-approximation algorithm produces set 109 in size as the worst result in this experiment. This information indicates that the algorithms give almost the same performance in the sparse graph.

When we look at Figure 5.12b which shows the results of graphs with 5 average degrees, the gaps between the algorithms increase a little. For example, Guha

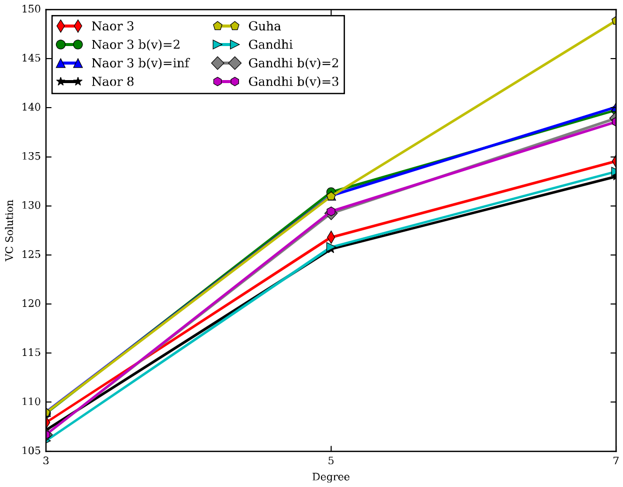


Figure 5.13. VC size performance of algorithms by average degrees.

and Naor's 3 approximation algorithms with $b(v) = 2$ and $b(v) = \infty$ produce vertex cover sets that have over 130 vertices. Interestingly, Naor 8 algorithm produces the best solution with 125 vertices.

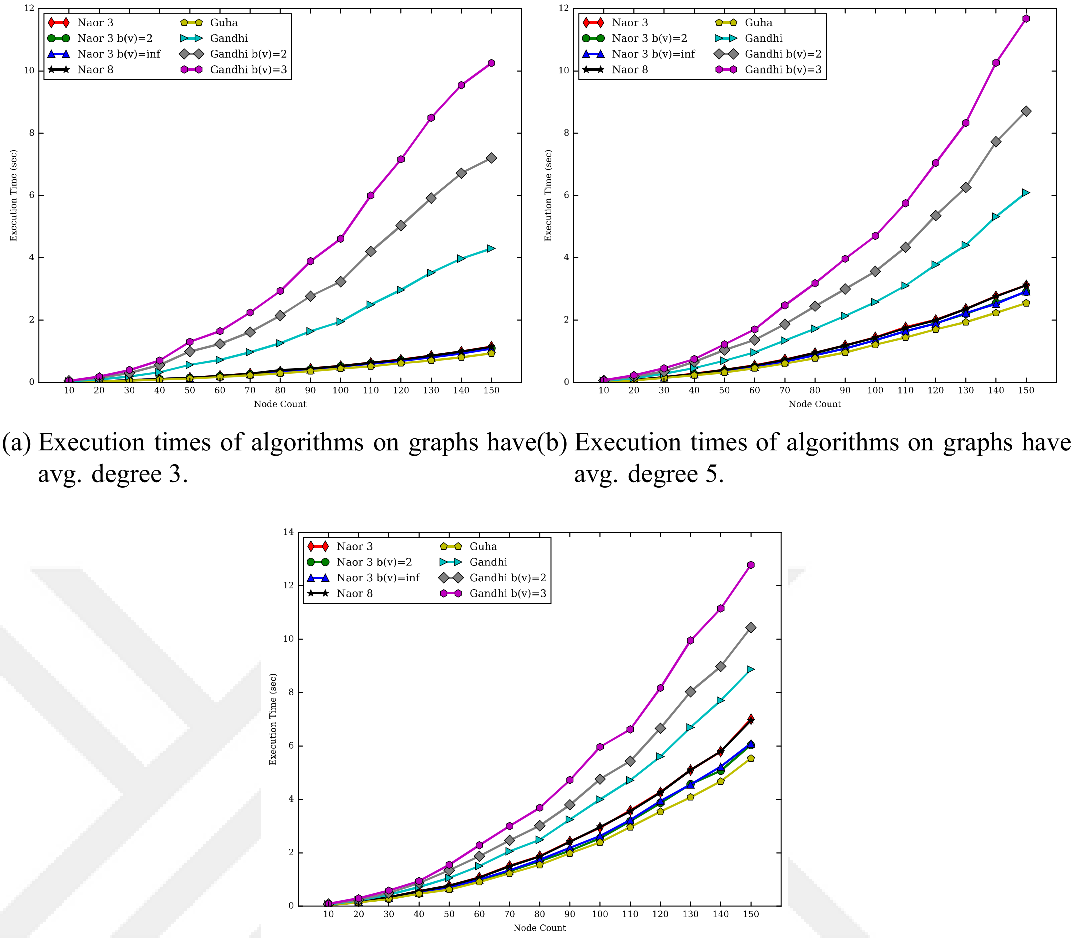
When the graphs get denser, the performances of the algorithms are seen clearly. In Figure 5.12c that contains the results of 7 average degrees, Guha's algorithm selects almost all vertices to cover graphs. In 150 sized graphs, Guha produces 148 sized vertex cover in average. Naor 8 and Gandhi algorithms give the best result with 133 vertices on graphs that have 150 vertices and 7 average degrees.

Figure 5.13 shows performances of the algorithms in case of the density of the graphs increase. Each algorithm is affected negatively by the increasing of the density while Guha's algorithm is the most affected.

5.3.2.2 Execution Time of Algorithms

The execution time of an algorithm is another crucial parameter to identify the correct algorithm to use. In this subsection, we elaborate on the execution times of the implemented algorithms. We measure the execution times of the algorithms by adding the linear programming execution times.

Figure 5.14 shows the execution times of the algorithms of different densities. For each density scenario, Gahndi's algorithm executes longer than the other algorithms. Since Gandhi's algorithm has a pre-processing step that aims to reduce $b(v)$ values as possible as for vertices v that have $k(v) = 1$, the algorithm executes longer. As seen in Figures 5.14a, 5.14b, 5.14c, as the $b(v)$ value increases, the exe-



(a) Execution times of algorithms on graphs have avg. degree 3.

(b) Execution times of algorithms on graphs have avg. degree 5.

(c) Execution times of algorithms on graphs have avg. degree 7.

Figure 5.14. Execution times of algorithms.

cution time of Gandhi's algorithm increases.

For the graphs which have 3 average degrees, the algorithms execute almost the same time except for Gandhi. For example, Guha's algorithm executes 0.93 seconds on average as the best result for these types of graphs. With 1.14 second execution time, Naor 3 algorithm ends up with worse results than Gandhi's algorithm.

When the average degrees of graphs increase, the execution times of the algorithms increase as expected, the main reason for which is the structure of the linear programming formulation. The yev variable depends on the number of edges in the graph. When the average degree of a graph increases, the linear program matrix will be getting bigger. Thus, solving the linear program will be longer. For 7 average degree graphs, the execution times of algorithms are affected more by increasing the number of nodes since problem size increases with the combination of edge numbers and node numbers. The best algorithm for execution time is Guha's algorithm with 5.53 seconds on 150 sized graphs while Naor 3 is the worst algorithm with 7 second execution time after Gandhi's implementation. In the light of such findings, it is appropriate to claim $b(v)$ value determines execution times of the algorithm when we exclude Gandhi's algorithm. In Guha's algorithm, there is no $b(v)$ value that restricts nodes, then the execution is faster. If we restrict the linear program with $b(v) = 1$ just like in Naor 3 and Naor 8, the execution times of the algorithms are longer. In contrast, the bigger $b(v)$ values in Gandhi's algorithm increase the execution time on account of the pre-processing step.

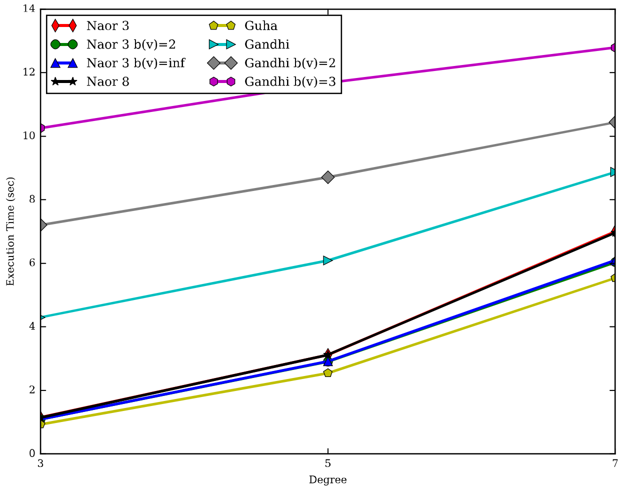


Figure 5.15. Execution times of algorithms by average degrees.

Figure 5.15 shows execution times of algorithms for different densities. For higher $b(v)$, the density of graph poses a less impact on Gandhi's algorithm. However, execution times of other algorithms increase parallel with density.

5.3.2.3 Approximation Ratio

In this subsection, we show the approximation performance of algorithms on its own optimal solution for each formulation.

The approximation ratio of algorithms on 3-degree graphs is shown in Table 5.6. This table shows us that if algorithm is restricted $b(v)$ the approximation ratio of algorithm will be low. Gandhi, Naor 3 and Naor 8 give the best approximation ratios since they do not allow a node to contribute more than one in a solution set (i.e *hard capacitated*). On the other side, when we relax the $b(v)$ as 2, 3, or infinity, the approximation ratio of the algorithm will be increase. Guha's algorithm produces the worst solution in terms of approximation ratio between 1.20-1.09. We can say

Table 5.6. Approximation ratios of algorithms on 3 average degree graphs.

Size	Naor3	Naor3 $b(v) = 2$	Naor $b(v) = \infty$	Naor8	Guha	Gandhi	Gandhi $b(v) = 2$	Gandhi $b(v) = 3$
10	1.12	1.15	1.15	1.10	1.15	1.12	1.13	1.13
20	1.10	1.12	1.13	1.10	1.12	1.09	1.11	1.11
30	1.13	1.18	1.18	1.11	1.17	1.11	1.15	1.16
40	1.14	1.16	1.18	1.13	1.20	1.13	1.16	1.16
50	1.08	1.10	1.10	1.07	1.09	1.07	1.08	1.08
60	1.10	1.12	1.12	1.09	1.13	1.09	1.10	1.10
70	1.07	1.11	1.11	1.06	1.10	1.05	1.08	1.08
80	1.09	1.13	1.13	1.08	1.11	1.08	1.11	1.11
90	1.10	1.13	1.12	1.10	1.12	1.07	1.10	1.10
100	1.10	1.13	1.13	1.10	1.13	1.08	1.10	1.10
110	1.10	1.12	1.11	1.09	1.12	1.08	1.09	1.09
120	1.07	1.09	1.09	1.07	1.11	1.06	1.08	1.08
130	1.07	1.09	1.10	1.06	1.10	1.05	1.07	1.07
140	1.09	1.11	1.11	1.08	1.11	1.08	1.10	1.10
150	1.09	1.11	1.11	1.08	1.11	1.07	1.09	1.09

that the algorithms get closer to the optimal solution as the graph gets bigger.

We can see the same situation in Table 5.7 which shows the results of 5-degree graphs. Naor 8 and Gandhi algorithm produce the best approximation ratios on these graphs with 1.16 on average. Generally, the approximation ratios of algorithms are less on the bigger graph size except for Gandhi's algorithm. Another observation about the bounding of $b(v)$ is that algorithms produce the same result after $b(v) > 2$. For example, Gandhi's algorithm produces almost the same result for 2 and 3 $b(v)$ values.

The characteristic of the algorithm on 7-degree graphs is the same as 3-degree and 5-degree graphs, as seen in Table 5.8. However, as graph become denser, the approximation ratio of the algorithm increases. When we look all table together, we can say that the approximation ratio is related to the density of a graph.

We have expected that Gandhi's algorithm will give the best approximation ratio followed by Naor 3 algorithm. Moreover, theoretical analysis has shown that Naor 8 has the worst approximation ratio. However, the practical result does not approve of that. Intuitively, we have waited the Guha's algorithm could give the worst practical approximation ratio because of its basic rounding nature which allows that an edge can be covered by two end-points. Naor 3 algorithm includes all vertices

Table 5.7. Approximation ratios of algorithms on 5 average degree graphs.

Size	**Naor3**	**Naor3** $b(v) = 2$	**Naor** $b(v) = \infty$	**Naor8**	**Guha**	**Gandhi**	**Gandhi** $b(v) = 2$	**Gandhi** $b(v) = 3$
10	1.16	1.25	1.26	1.16	1.29	1.14	1.21	1.21
20	1.18	1.23	1.25	1.17	1.24	1.17	1.21	1.21
30	1.24	1.28	1.28	1.24	1.39	1.22	1.26	1.26
40	1.21	1.26	1.26	1.20	1.29	1.19	1.23	1.23
50	1.14	1.20	1.20	1.13	1.22	1.14	1.17	1.17
60	1.17	1.23	1.23	1.16	1.27	1.15	1.20	1.20
70	1.16	1.21	1.22	1.15	1.21	1.16	1.19	1.18
80	1.16	1.20	1.21	1.16	1.23	1.15	1.20	1.19
90	1.15	1.19	1.19	1.13	1.20	1.13	1.18	1.18
100	1.19	1.23	1.24	1.18	1.24	1.17	1.21	1.22
110	1.16	1.22	1.22	1.15	1.25	1.15	1.19	1.19
120	1.16	1.21	1.21	1.15	1.24	1.16	1.20	1.20
130	1.17	1.21	1.21	1.16	1.25	1.15	1.20	1.19
140	1.15	1.20	1.20	1.13	1.22	1.13	1.18	1.17
150	1.14	1.20	1.20	1.13	1.20	1.13	1.18	1.18

Table 5.8. Approximation ratios of algorithm on 7 average degree graphs.

Size	**Naor3**	**Naor3** $b(v) = 2$	**Naor** $b(v) = \infty$	**Naor8**	**Guha**	**Gandhi**	**Gandhi** $b(v) = 2$	**Gandhi** $b(v) = 3$
10	1.11	1.18	1.18	1.09	1.20	1.09	1.16	1.16
20	1.16	1.23	1.25	1.16	1.30	1.18	1.22	1.22
30	1.17	1.20	1.22	1.17	1.28	1.16	1.19	1.20
40	1.20	1.27	1.27	1.19	1.37	1.19	1.25	1.25
50	1.14	1.20	1.19	1.12	1.27	1.13	1.19	1.19
60	1.12	1.17	1.18	1.12	1.23	1.12	1.17	1.17
70	1.14	1.21	1.21	1.14	1.30	1.13	1.18	1.18
80	1.16	1.22	1.23	1.15	1.31	1.15	1.20	1.20
90	1.17	1.23	1.23	1.16	1.31	1.17	1.22	1.22
100	1.16	1.23	1.22	1.15	1.28	1.15	1.21	1.22
110	1.18	1.23	1.24	1.17	1.31	1.17	1.22	1.22
120	1.20	1.26	1.25	1.19	1.35	1.18	1.23	1.23
130	1.17	1.23	1.24	1.17	1.34	1.16	1.21	1.22
140	1.17	1.22	1.23	1.16	1.31	1.17	1.22	1.22
150	1.19	1.25	1.25	1.18	1.33	1.18	1.24	1.24

which have $x_v \geq \frac{1}{3}$ while Naor 8 includes vertices that have $x_v \geq \frac{1}{2}$. This difference causes that Naor 3 includes more vertices than Naor 8 at the setup phase. Even if the approximation ratio of Naor 8 algorithm is bigger than Naor 3's, this algorithm shows the better result in the average case. Gandhi algorithm with $b(v) = 1$ got the same approximation ratios with Naor 8 algorithm since it allows vertices that have $x_v \geq \frac{1}{2}$ in the setup phase. Although Gandhi's algorithm has a pre-processing phase, it could not be useful to reduce the approximation ratio. As we mentioned before, relaxed $b(v)$ values cause worse approximation ratio for Naor and Gandhi.

5.4 Performance Evaluation of Self-Stabilizing Graph Covering Algorithm

In this section, we consider vertex cover in WSNs as the link-monitoring problem application. We implement and examine the practical fault tolerance performances of self-stabilizing vertex cover and independent set algorithms. Note that VC algorithms are approximation algorithms, where IS algorithms do not have an assurance for approximating the MVC.

5.4.1 Our Model

We assume a wireless sensor network where nodes are arbitrarily distributed over a 2D space. As for the communication model, we adopt the *message passing* model (Afek and Brown, 1993; Dolev et al., 1991). Nodes run independently with respect to spatially but synchronously in time. Algorithms' executions occur in phases. Nodes do calculations in even¬-numbered phases, and nodes send the results of computations in odd¬-numbered phases (i.e. moves) to inform their neighbors about their status. Note that our assumptions are well-suited and modeled under the notion of *synchronous schedulers* where each node is allowed to make at least one move in a round certainly makes one move before the end of the round (Dubois and Tixeuil, 2011). Each node has information about the state of its neighbors. However, due to concurrency, in any round t, a node knows the state of its neighbors in round $t - 1$. Note that as nodes run concurrently and independently in WSNs, the central scheduler is not a realistic choice for WSNs. However, there exist transformers which enable a self-stabilizing algorithm designed for the central scheduler to run under distributed scheduler, such as local mutual exclusion method (ULME)) (Beauquier et al., 2000) or randomized self-stabilizing (Turau and Weyer, 2006). We implement these two methods to transform Shukla et al.'s algorithm to make it run under the distributed scheduler.

Table 5.9. Parameters and Environment of Simulations

Simulator	TOSSIM
Mote	IRIS
Topology	Geometric
Number of Nodes	50-250 (step 50)
Node Degrees	3, 5, 7
Total Number of Instances	150
Scheduler	Synchronous

5.4.2 Simulation Environment

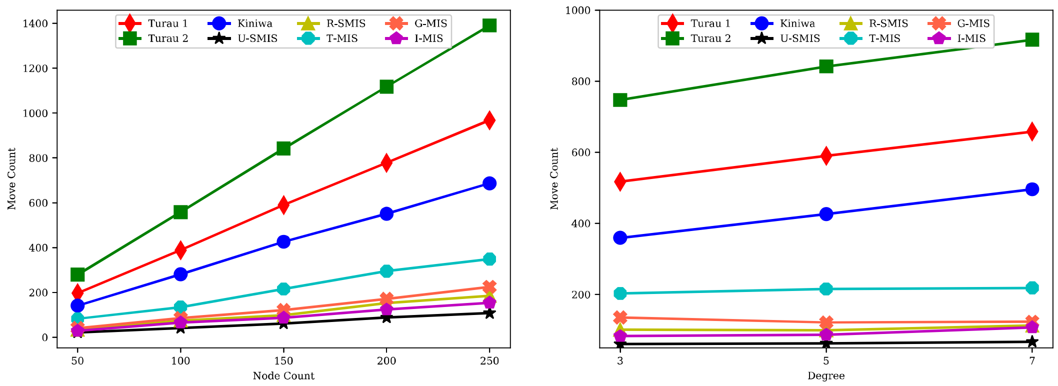
We simulated algorithms on TOSSIM wireless sensor network simulator (Levis et al., 2003) which is discrete event simulator for nodes running TinyOS (Levis et al., 2004), to compare performance evaluation of algorithms. We produced geometric graphs. The order of the mentioned graphs varies from 50 to 250 with a step increment of 50. The average degree of the produced graphs is 3, 5, and 7. For each size and average degree pair, we produce 10 different topologies. We have 150 network instances in total. We assigned the state variables of all nodes randomly so that each node is either IN or OUT (and WAIT) with equal probability for the initial configurations. The execution of each round contains two phases. Table 5.9 includes the parameters of simulation environment.

We compared the algorithms depending on the number of rounds, cardinality of vertex cover set and the move count. Hereafter, Turau's basic VC algorithm and improved VC algorithm, and Kiniwa's algorithm will be called Turau 1, Turau 2 and Kiniwa, respectively. For the MIS algorithms, we called Shukla et al.'s algorithm as U-SMIS and R-SMIS that adopted ULME and randomized self-stabilizing methods, respectively. Ikeda et al.'s, Goddard et al.'s, and Turau's algorithms are named I-MIS, G-MIS, T-MIS, respectively.

5.4.3 Computational Results

5.4.3.1 Move Count

The total number of moves made by the nodes until the system stabilizes from an adversarial configuration is represented by the move count. Since nodes must inform their neighbors of their new states after each move in the message passing communication model, the number of moves is an essential indicator for the total number of messages that should be sent throughout the execution.



(a) Move count vs. graph size. Average degree: 5. (b) Move count vs. average degree. Node count: 150.

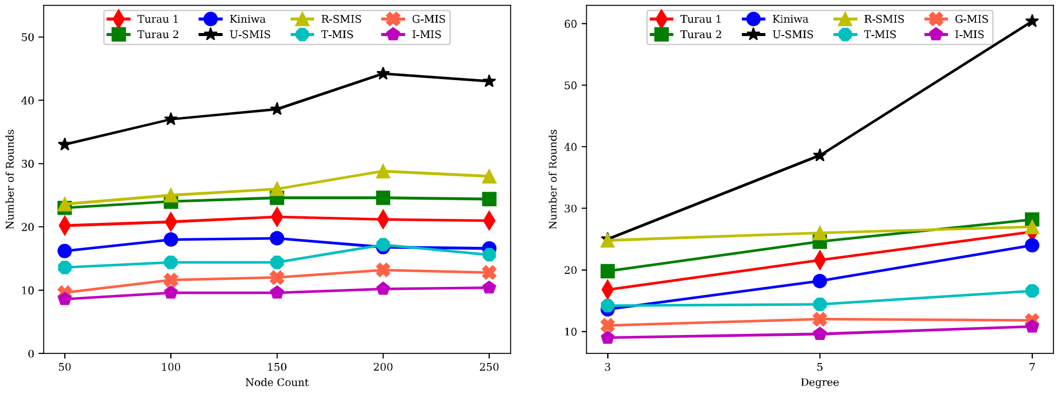
Figure 5.16. Move count of algorithms.

The required move counts for the algorithms to stabilize in the case of a arbitrary initial configuration for graph size and average degree of the graph are shown in Figures 5.16a and 5.16b, respectively. All IS algorithms require less number of moves for stabilization than VC algorithms and that is the most important observation. T-MIS algorithm needs around 1.2 moves per node, where all other IS algorithms give similar results and requires less than 1 move per node to stabilize. U-SMIS is the most agile algorithm as I-MIS comes second and G-MIS is the third. The best performing VC algorithm is Kiniwa's algorithm in which the total number of moves doubled that of the worst-performing IS algorithm (T-MIS). Turau's improvement algorithm (Turau 2) needs 14-times more moves than U-SMIS.

In Figure 5.16b, we show the move count results when we fix the graph size at 150 and change the density of the graph. We observe that IS algorithms are not affected by the changes in density and give similar results in denser graphs. On the other hand, the number of total moves required by VC algorithms goes up as the density of the graph grows.

5.4.3.2 Number of Rounds

As mentioned before, enabled nodes make their moves concurrently and every enabled node definitely makes one move under synchronous scheduler. Since the synchronous system that we assume is a time-synchronized system, an important parameter for the performance of a self-stabilizing algorithm is the total number of rounds it takes to converge to a legitimate configuration. In this subsection we provide results for round counts.



(a) Round count vs. graph size. Average degree: 5.(b) Round count vs. average degree. Node count: 150.

Figure 5.17. Round count of algorithms.

According to Figure 5.17a, we observe that most of the IS algorithms are more agile than VC algorithms. I-MIS is the quickest algorithm to stabilize in all sizes of graphs as it converges in around 9 rounds. G-MIS and T-MIS perform well, too, as they need around 11 and 13 rounds to converge, respectively. We see that U-SMIS, which is the algorithm that requires the least number of moves (Figure 5.16a), is the slowest algorithm to converge. Furthermore, R-SMIS is the second slowest although it is one of the algorithms that requires the least number of moves. Recall that U-SMIS and R-SMIS are the modified versions of Shukla et al.'s algorithm which has been designed to work with the central scheduler. We use ULME and randomization techniques to make them run under a distributed scheduler. For instance, in U-SMIS, only a single node in a 1-hop neighborhood is scheduled to make a move, which explains its longer running time.

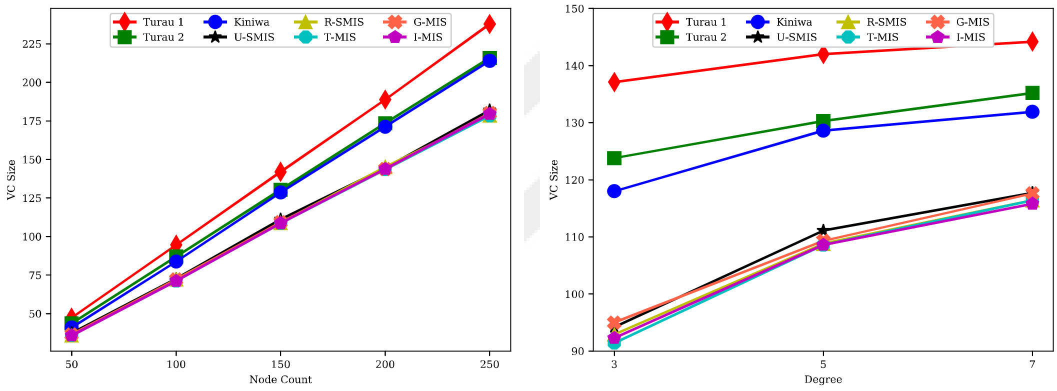
It is obviously seen that the IS algorithms are quicker than VC algorithms. Both algorithms of Turau have reached the stable state in more than 20 rounds, while the quickest VC algorithm (Kiniwa) needs around 16 rounds and could only slightly outperform T-MIS in graphs with 200 nodes.

It is observable that the running times of the algorithms are quite independent of the node count as they require the same number of rounds for all sizes of graphs. Exceptions are U-SMIS and R-SMIS since their running times slightly grow as the graphs become larger.

Figure 5.17b shows the change in required number of rounds as the graph gets denser and the size of the graph is fixed at 150. We observe that VC algorithms show linearly increasing behavior concerning time as the density increases. The only IS algorithm that shows the similar rise in time is U-SMIS. As explained in Section 5.4.3.2, U-SMIS does not allow neighbor nodes to make concurrent moves, which means that the maximum possible number of enabled nodes in a single round decreases as the density of the graph grows. Note that if the graph is complete, U-SMIS can only schedule one node in a single round.

We see that the running times of the other IS algorithms (R-SMIS, T-MIST, G-MIS, I-MIS) were not affected by the changes in the density of the graph.

5.4.3.3 Cardinality of Vertex Cover



(a) Vertex cover size vs. graph size. Average degree: 5.

(b) Vertex cover size vs. average degree. Node count: 150.

Figure 5.18. Cardinality of vertex cover set of algorithm.

In this subsection, we discuss the performance of algorithms in achieving smaller vertex covers. Figure 5.18a shows that IS algorithms considerably outperformed VC algorithms in finding vertex covers. All IS-based algorithms give similar results in all sizes of graphs. As an example, they all provide vertex cover with a size around 175 in graphs with 250 nodes. Turau's improvement algorithm and Kiniwa's algorithm result in more than 15% larger VC solutions. Turau's basic algorithm gives nearly 35% larger sets with respect to IS algorithms.

The vertex cover sets that the algorithms produce based on the increasing node degree are shown in Figure 5.18b. As shown in Figure 5.18b, there is a positive correlation between the size of the vertex cover set with the increased average node degree. The MIS-based algorithms produce the best results even though they are slightly more affected by the increase of average degree compared to matching-based VC algorithms. Turau 1, Turau 2, Kiniwa, and all MIS based algorithms in the 7-degree graphs have produced vertex cover sets in sizes 144, 134, 131, and around 117, respectively.

6. SELF-STABILIZING VERTEX COVER ALGORITHMS WITH CAPACITY CONSTRAINT

In this chapter, we introduce two self-stabilizing capacitated vertex cover algorithms that enable the system to change its state from arbitrary to desired stable state. The first proposed algorithm (SS-CVC1) is a modification of Ikeda's (Ikeda et al., 2002) algorithm. The second one, which uses the greedy technique (SS-CVC2) is a novel algorithm for the problem. To the best of our knowledge, this is the first work that combines capacitated and self-stabilizing concepts for the vertex cover problem. Additionally, we evaluate these algorithms with existing self-stabilizing vertex cover algorithms which were modified in order to satisfy the capacitated property.

The remainder of this paper is organized as follows. We propose our algorithms in subsection 6.1, then we provide correctness and self-stabilizing proof of the algorithms in subsection 6.2. Performance evaluations of the algorithms are widely discussed in subsection 6.3. Lastly, we conclude our study and findings in subsection 6.4.

6.1 Proposed Algorithms

In this section, the proposed algorithms are explained and exemplified on the sample graphs. We firstly introduce a maximal independent set based algorithm, which is called SS-CVC1. After the first algorithm, we introduce our second algorithm which needs 2-hop information about the graph.

6.1.1 SS-CVC1 Algorithm

In this subsection, we present the SS-CVC1 algorithm which is based on Ikeda's MIS algorithm (Ikeda et al., 2002). Ikeda's MIS algorithm runs under the unfair scheduler and stabilizes itself at most $\mathcal{O}(n^2)$ steps. We modify the first two rules of Ikeda's algorithm to obtain a vertex cover set. Each node $u_v \in V$ maintains $covered_u$ variable that has two different states: $\{0, 1\}$. After the first two rules, we add two additional rules which satisfy the capacity constraint. Algorithm 6.7 shows the proposed SS-CVC1 algorithm.

R1 is a simple rule that the node u changes its $covered_u$ variable by looking all of its neighbors v's $covered_v$ variable. If $covered_u$ variable of the node u is 1 and all of its neighbors v's $covered_v$ variable is 1, node u changes $covered_u$ variable to 0. Figure 6.1a shows a sample scenario from u point of view where all neighbors of

Algorithm 6.7 SS-CVC1

```
process $u$
$N(u)$ neighbors of node $u$
Variables :
    $covered_u \in \{0,1\}$
    $nex_u \in \mathbb{N}^+$
Rules :
R1: $covered_u = 1 \wedge \forall j \in N(u) : covered_v = 1$
        $\Rightarrow covered_u := 0$
R2: $covered_u = 0 \wedge \exists j \in N(u) : covered_v = 0 \wedge cost_1(v) > cost_1(u)$
        $\Rightarrow covered_u := 1$
R3: $covered_u = 1 \wedge nex_u \neq \lceil \frac{|N(u)|}{cap_u} \rceil$
        $\Rightarrow nex_u := \lceil \frac{|N(u)|}{cap_u} \rceil$
R4: $covered_u = 0 \wedge nex_u \neq 0$
        $\Rightarrow nex_u := 0$
```

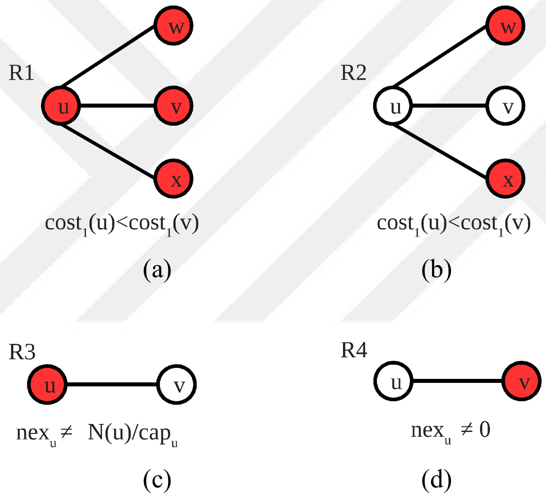


Figure 6.1. Sample scenarios for rules of SS-CVC1.

u are already covered, thus u sets $covered_u$ as 0.

In the **R2**, each node decides whether or not to join the vertex cover set by looking at the neighbors' $covered_v$ and $cost_1(v)$. We define a $cost(u)$ function to select greedily the vertex that joins the set. A vertex u simply calculates its cost with the formula seen in Equation 6.1. The algorithm chooses the vertex with the locally minimum cost. In Figure 6.1b node u enables **R2** since its cost is lower than v.

R3 and **R4** regulate the nex_u variable that is the number of existence in vertex cover set for vertex u. If $covered_u$ is 1 and nex_u is not correct for vertex u, as seen in Figure 6.1c, the vertex sets nex_u to $\lceil \frac{N(u)}{cap_u} \rceil$. On the other hand, if $covered_u$ is 0 and nex_u is not equal to 0, the vertex sets it to 0 as that illustrated in Figure 6.1d.

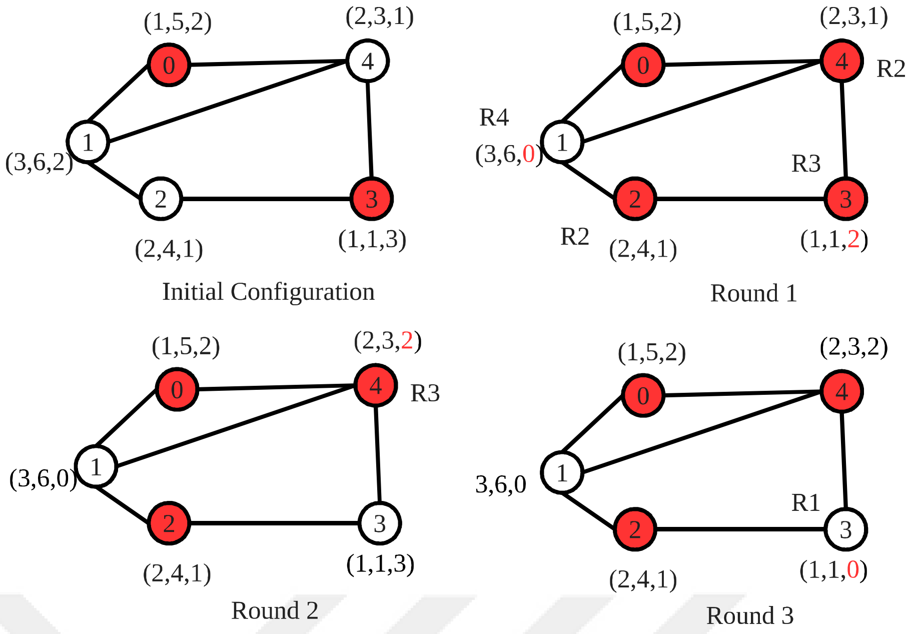


Figure 6.2. Execution of the SS-CVC1 algorithm.

$$cost_1(u) = \frac{\lceil \frac{|N(u)|}{cap_u} \rceil * weight_u}{|N(u)|} \tag{6.1}$$

In Figure 6.2, an example of the execution of the SS-CVC1 algorithm is shown. The algorithm starts with an arbitrary initial configuration where each vertex is labeled as (cap, weight, nex). The red color represents that the vertex is already in the vertex cover set. At a given initial configuration, the costs of vertices are 5, 2, 2, 3, 2, respectively. Although vertex 1 and vertex 2 have the same cost 2, the algorithm uses a vertex identifier to prevent the neighbor nodes to enter into the vertex cover set together. In the first round, vertices 2 and 4 execute **R2** to set their $covered$ variable to 1. Vertex 1 executes **R4** and sets nex_1 to 0. Vertex 3 executes **R3** to justify nex_3 to 2. In the second round, vertex 3 executes **R1** and sets $covered_3$ variable to 0 since all of its neighbors are currently in the vertex cover set. Also, vertex 4 sets nex_4 to 2 by executing **R3**. In the last round, vertex 3 executes **R4** to set nex_3 to 0. Optimal solution weight is 9 and SS-CVC1 produces the optimal solution for this example.

6.1.2 SS-CVC2 Algorithm

In this subsection, we improve our SS-CVC1 algorithm to reduce the weight of the produced vertex cover solution and round complexity. In order to achieve this, we facilitate a model defined by Turau (Turau, 2012). The model is named as the

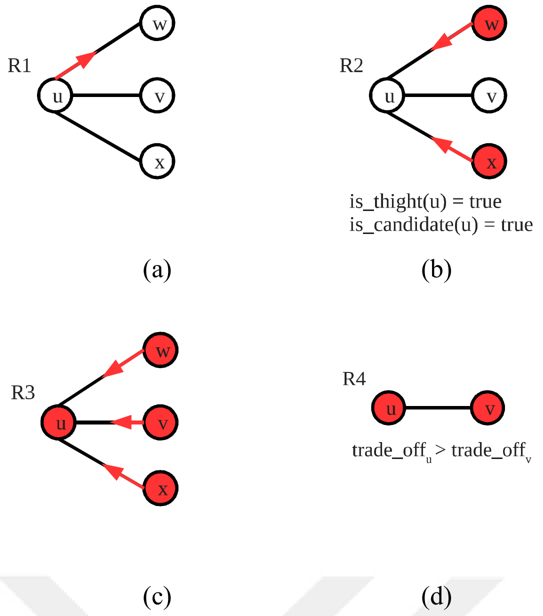


Figure 6.3. Sample scenarios for rules of SS-CVC2.

expression model in which each vertex u has expressions that show their states and states of their neighbors. 2-distance model is a special case of the expression model. In the expression model, each vertex not only reads the states of its neighbors but also reads the expressions of all of its neighbors in an atomic step. We present SS-CVC2 capacitated self-stabilizing vertex cover algorithm consisting of two expressions. The pseudo-code of the algorithm is shown in Algorithm 6.8.

The expression $is\_tight(u)$ checks if the node u is already in vertex cover set and nex_u is correct. We use this expression to include a vertex in the vertex cover set. The expression $is\_candidate(u)$ returns true if a node u has cost locally optimal among all of its neighbors $N(u)$ which could enter the vertex cover set. The macro $trade\_off_u$ is used to calculate the trade-off value of a vertex. We need to calculate the trade-off value for vertex u, so we count double-covered edges and then multiply this value with the payback value $\frac{nex_u \times weight_u}{covered\_by\_me_u}$ for an edge.

The expression $has\_max\_trade\_off_u$ is used to find the locally optimal vertex to exclude it from the vertex cover set. If a node has max trade-off among all of its neighbors with double covered edges and is already tight, this expression returns true.

Note that its sequence number defines the priority of each rule.

Algorithm 6.8 SS-CVC2

```
process $u$
Variables :
    $covered_u \in \{0, 1\}$
    $nex_u \in \mathbb{N}^+$
    $uncovered\_edges_u = \{\forall(u, v) \in E(u) \mid u \not\rightarrow (u, v) \wedge v \not\rightarrow (u, v)\}$
    $covered\_by\_me = \{\forall(u, v) \in E(u) \mid u \rightarrow (u, v)\}$
Macros:
    $double\_covered_u, = \{(u, v) \in E(u) : i \rightarrow (u, v) \wedge v \rightarrow (u, v)\}$
    $trade\_off_u \ = \frac{nex_u \times weight_u}{covered\_by\_me_u} \times double\_covered_u$
Predicates:
    $all\_edges\_covered\_by\_neighs(u) \equiv (u, v) \in E(u) : v \rightarrow (u, v) \ \ \forall v \in N(u)$
Expression:
    $is\_tight(u) \equiv (nex_u = \lceil \frac{|uncovered\_edge_u| + |covered\_by\_me|}{cap_u} \rceil) \wedge covered_u = 1$
    $is\_candidate(u) \equiv (\forall v \in N(u) : uncovered\_edges_v \neq \emptyset \vee (covered\_by\_me_v \neq \emptyset \wedge is\_tight(v) = false),\ cost(u) < cost(v))$
    $has\_max\_trade\_off(u) \equiv (\forall v \in N(u) : trade\_off_u > trade\_off_v)$
R1: $(uncovered\_edge_u \neq \emptyset \vee covered\_by\_me_u \neq \emptyset) \wedge \neg is\_tight(u) \wedge is\_candidate(u)$
        $\Rightarrow covered_u := 1$
        $\Rightarrow nex_u := \lceil \frac{|uncovered\_edge_u| + |covered\_by\_me|}{cap_u} \rceil$
        $\Rightarrow \forall(u, v) \in uncovered\_edges_u$ do $u \rightarrow (u, v)$
R2: $uncovered\_edge_u \neq \emptyset \wedge is\_tight(u) \wedge is\_candidate(u)$
        $\Rightarrow \forall(u, v) \in uncovered\_edges_u$ do $u \rightarrow (u, v)$
R3: $all\_edges\_covered\_by\_neighs(u) \wedge (covered_u \neq 0 \vee nex_u \neq 0 \vee |covered\_by\_me| > 0)$
        $\Rightarrow covered_u = 0$
        $\Rightarrow nex_u = 0$
        $\Rightarrow \forall(u, v) \in covered\_by\_me_u$ do $i \not\rightarrow (u, v)$
R4: $double\_covered \neq \emptyset \wedge has\_max\_trade\_off(u)$
        $\Rightarrow \forall(u, v) \in E(u) : u \rightarrow (u, v) \wedge v \rightarrow (u, v)$ do $u \not\rightarrow (u, v)$
        $\Rightarrow nex_u = \lceil \frac{|covered\_by\_me_u|}{cap_u} \rceil$
        if $nex_u = 0$
            $covered_u = 0$
```

R1 selects a locally optimal node to enter the solution set by calculating the cost of each node that is candidate to enter this set. To calculate the cost of a vertex u, Equation 6.2 is used. If a node u enables **R1** and is privileged by the scheduler, it sets $covered_u$ to 1, nex_u to the correct value and covers all edges in $uncovered_{e}dge_{u}$. Figure 6.3a gives an example configuration for the activation of **R1** from u's perspective. In this configuration, u is not tight, because $covered_u = 0$ and it covers edge (u, w) but edges (u, v) and (u, x) are not covered by others. The cost of u is lower than its neighbors for this scenario, therefore the only enabled node becomes u which enables **R1**.

$$cost_2(u) = \frac{\lceil \frac{|uncovered\_edge|}{cap_u} \rceil * weight_u}{|uncovered\_edge|} \tag{6.2}$$

If a node u's $uncovered\_edge_u$ set is not empty but the node is tight, the node checks whether is a candidate. If the node is a candidate to join the vertex cover set, it enables **R2**. In Figure 6.3b, node u activates **R2** since it is locally optimal candidate to join vertex cover set and it already satisfies the tightness that distinguishes **R1** and **R2**. If the node makes a move, it just covers all edges in $uncovered\_edge\_u$ so $covered_u$ and nex_u variables become correct.

In **R3**, if at least one of $covered_u$, nex_u, $|covered\_by\_me|$ variables of a node u, which means its all incident edges are covered by all of its neighbors, is not 0, the node executes **R3** and resets these variables. As seen in Figure 6.1c, node u activates **R3** since all of its neighbors covers all incident edges to it but $covered_u = 0$

We use **R4** to exclude unnecessarily selected vertices from the solution set by calculating their trade-off value. As depicted in Figure 6.3d, the edge (u, v) is covered by two endpoints, each node calculates its trade-off value and the node with the maximum trade-off value, u in this case, reduces its nex variable. If the newly calculated nex variable is 0, the node sets $covered$ variable to 0 as well.

Figure 6.4 shows the execution of the SS-CVC2 algorithm on the arbitrary initialized graph. The nodes are labeled as in the execution of SS-CVC1 in Figure 6.2. We also randomly initialize $covered\_by\_me$ sets for each vertex. The red vertices represent the covered vertices and red arrows show the edges covered by the vertices. In Round 1, vertex 2 excludes itself from the vertex cover set by executing **R4** since its $trade\_off_2$ value 8 is more than vertex 3. Vertex 4 excludes itself from the vertex cover set by executing **R3** since its neighbors cover all of its edges. Vertex 3 covers itself by executing **R1** with $cost_1 = 0.75$ as the local minimum cost.

In Round 2, vertices 1 and 6 execute **R1** since they have 3 and 2 costs which are locally minimum, respectively. 0 and 5 execute **R1** since they have 1 cost in Round 3. Each vertex contributes to the solution set only once, as a result the total weight of this solution is 15, namely, the optimal solution.

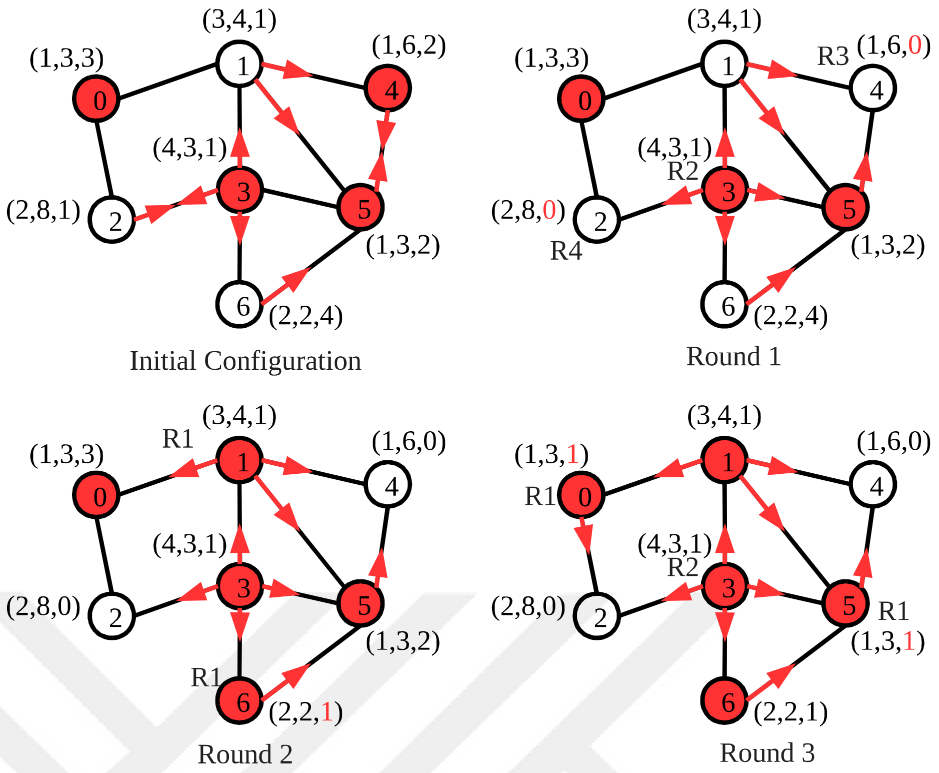


Figure 6.4. An execution of the SS-CVC2 algorithm.

6.2 Theoretical Analysis Of Algorithms

In this section, proof of the correctness and step complexities of algorithms are provided. SS-CVC1 algorithm is proved via Ikeda's proof method since this algorithm is designed thanks to the inspiration gained from Ikeda's algorithm. It is shown that the SS-CVC2 algorithm stabilizes under unfair distributed scheduler in $O(n)$ step complexity.

6.2.1 Theoretical Analysis of SS-CVC1

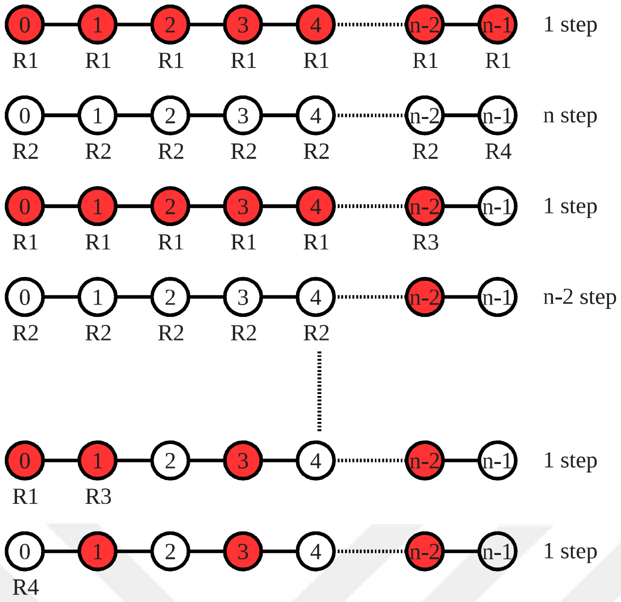


Figure 6.5. Worst case scenario for SS-CVC1 algorithm.

Theorem 6.1. SS-CVC1 algorithm stabilizes after $\mathcal{O}(n^2)$ steps under unfair distributed scheduler.

Proof. Assume that a scheduler gives permission immediately to all nodes which attempt to execute **R1** and **R3** simultaneously, and gives permission one by one to the other nodes which attempt to execute **R2** and **R4**. Consider a configuration as shown in Figure 6.5 whose cost values are increasing as the following order $cost(0) < cost(1) < ... < cost(n-1)$.

The scheduler allows all nodes to make their moves in one step for the first phase. In the second phase, all nodes are privileged one by one and this process takes to n step. The third phase is ended in only one step because each node wants to execute **R1** or **R3**. In phase 4, $n-1$ and $n-2$ stabilize and do not want to make any move. Thus, the rest of the nodes make their moves in $n-2$ steps. The number of nodes that want to make a move decreases by 2 in each consecutive two rounds. We can formalize this relation as $1+n-(2 \times r)$ where r is the sequence number of two consecutive phases. Such a scenario is shown in Figure 6.5, in which the system stabilizes when $r = \frac{n-1}{2}$. We can formulate Equation 6.3 using these information.

$$\begin{aligned}
&\sum_{r=0}^{\frac{n-1}{2}} 1+n-(2\times r)=\sum_{r=0}^{\frac{n-1}{2}} 1+\sum_{r=0}^{\frac{n-1}{2}} n-\sum_{r=0}^{\frac{n-1}{2}} 2r\\
&=\frac{n+1}{2}+\frac{n(n+1)}{2}-\frac{(n-1)(n+1)}{4}\\
&=\frac{2(n+1)}{4}+\frac{2n(n+1)}{4}-\frac{(n-1)(n+1)}{4}\\
&=\frac{(n+1)(2+2\,n-n+1)}{4}=\frac{(n+1)(n+3)}{4}\\
&\qquad=\mathcal{O}(n^2) \qquad (6.3)
\end{aligned}$$

When we solve the summation formula, we get to $\mathcal{O}(n^2)$ step and this concludes the proof of the theorem. □

6.2.2 Theoretical Analysis of SS-CVC2

In this subsection, we have proved that the SS-CVC2 algorithm is a self-stabilizing capacitated vertex cover algorithm. Firstly, we show the SS-CVC2 algorithm to produce a capacitated vertex cover solution when it reaches a stable configuration. Following that, we will show that the algorithm reaches the stable configuration in a finite number of moves under the unfair distributed scheduler.

Lemma 6.2. When the algorithm is in the stable configuration $\forall e \in (u,v) \in E$: $(u \to e \wedge \ covered_u = 1) \vee (v \to e \wedge covered_v = 1)$.

Proof. If there is such an edge as $e = (u,v)$, both endpoints $i,\ j$ do not cover e, u or v execute **R1** or **R2** by their costs and sets $covered = 1$. □

Lemma 6.3. In the stable configuration $\forall i \in V : nex_u = \lceil \frac{covered\_by\_me_u}{cap_u} \rceil$.

Proof. If nex_u variable of a vertex u is not equal to $\lceil \frac{covered\_by\_me_u}{cap_u} \rceil$. The vertex could update its nex_u according to the following four different ways,

- If the $is\_tight(u)$ expression of u is false and neither $covered\_by\_me$ nor $uncovered\_edges$ are empty, u executes **R1** and updates $nex_u = \lceil \frac{|uncovered\_edge_u|+|covered\_by\_me|}{cap_u} \rceil$. After the execution of **R1**, all edges in $uncovered\_edges$ are covered by u.

- If vertex u is tight but it has at least one edge in $uncovered\_edges_u$, it executes **R2** to set $nex_u = \lceil \frac{|uncovered\_edge_u|+|covered\_by\_me|}{cap_u} \rceil$ and covers all edges in $uncovered\_edges_u$.
- If all incident edges to u are covered by $N(u)$, u executes **R3** and sets $nex_u = 0$ and uncovers all edges that are covered by itself.
- If an edge $e = (u, v)$ is covered by both endpoints, the edge is considered as a double-covered edge. The vertex u which has the maximum $trade\_off$ among its 1-hop local neighborhood, uncovers the double covered edges and updates its nex_u by $covered\_by\_me_u$.

□

Lemma 6.4. In the stable configuration, an edge $e = (u, v)$ is covered by its only one endpoint u or v.

Proof. Assume a situation that e is covered by both endpoints. u or v will execute **R4** according to their $trade\_off()$, which is a contradiction and proves the lemma. □

Theorem 6.5. A stable configuration SS-CVC2 is a minimal capacitated vertex cover.

Proof. When the system stabilizes, the $covered$ variable of one endpoint of an edge is 1 (Lemma 6.2). According to Lemma 6.3, nex variables of the covered vertices are equal to $\lceil \frac{covered\_by\_me_u}{cap_u} \rceil$. If a vertex provides these properties, the vertex is considered as tight.

If all neighbors of vertex u already cover each connected edge, the node executes **R3**. To prevent the double covered edge, **R4** is executed by the neighbouring vertices with the maximum trade-off. Thus, these give us the minimality property for the vertex cover solution. □

Lemma 6.6. Each vertex could execute either **R1** or **R2** only once.

Proof. **R1** and **R2** are used to enter the vertex cover set and they are mutually exclusive rules due to $is\_tight()$ expression. Once a vertex enters into the vertex cover set, it does not execute neither **R1** nor **R2** until a fault occurs. □

Lemma 6.7. Each vertex executes **R3** only once.

Proof. Due to the arbitrary initial configuration property of self-stabilizing, the given graph could start a configuration where vertices cover all edges. In such a configuration, all nodes except the one which executes **R1** or **R2**, execute **R3** only once. In the configuration where a vertex u has vertices in $covered\_by\_me_u$ but its cost is higher than its neighbor's cost, u executes **R3** only one time when its neighbors enter the solution set. □

Lemma 6.8. Each vertex executes **R4** only once.

Proof. Two neighbor vertices could cover the same edge due to the arbitrary initial configuration or execution of the algorithm. To prevent this, each node with double covered edges and maximum trade-off value executes **R4** only one time during the execution of the algorithm. □

Lemma 6.9. If a vertex v executes **R3**, it does not execute **R4**.

Proof. If a vertex v makes an **R3** move, then it does not have double-covered edges due to the removing edges from $covered\_by\_me_u$. □

Theorem 6.10. The step complexity of the SS-CVC2 algorithm under the unfair distributed scheduler is $\mathcal{O}(n)$.

Proof. The unfair distributed scheduler does not guarantee the privilege to activate all nodes in any round, but at least one node in one round. Because of Lemma 6.6 and 6.9, each vertex could make a maximum of two moves. By the definition of the unfair scheduler, it takes $2 \times n$ step to stabilizing system. According to this information, the step complexity of SS-CVC2 is $\mathcal{O}(n)$. □

6.3 Performance Evaluation

6.3.1 The Model

We assume that each node has a unique identifier and a local variable that stores its neighbors. We run algorithms under the unfair distributed scheduler which is the most restricted scheduler type since it does not guarantee that all active nodes are privileged eventually. The scheduler prevents nodes from making moves with 0.5 probability but guarantees global progress by privileging at least one vertex in each step. We execute our algorithms on a simulator which was proposed in (Ileri and Dagdeviren, 2018) to execute self-stabilizing algorithms.

6.3.2 Experimental Setup

We use random geometric network models in order to simulate wireless sensor networks. Each vertex of the given graph *G(V,E)* is scattered to 2D area A. If Euclidean distance between two vertices is smaller than their unit range r, these nodes are considered connected. The size of randomly generated graphs varies from 50 to 250 (with 50 steps). Graphs are divided into three sparsity groups which have 3, 5, and 7 average degrees for each vertex in a graph. Each performance metric is obtained through 30 different simulation scenarios. We randomly assign a weight to each vertex in the interval $[1 - 50]$. Moreover, we assign each vertex u a cap value in the interval $[1 - \Delta]$ where Δ represents the maximum degree of a graph.

In addition to the algorithms we propose, we implement the algorithms of Kiniwa and Turau. To provide the capacity constraint for the algorithms of Kiniwa and Turau, we add nex and cap and $weight$ variables and two rules which are shown in the Algorithm 6.9. Furthermore, we provided the 1-hop implementation of SS-CVC2 algorithm thanks to the transformer proposed in (Turau, 2012).

All variables of each node are initialized randomly before the algorithm starts. When a node changes its state, all 1-hop neighbors can see this move (for SS-CVC2 2-hop information is provided).

We compare algorithms in terms of move count, step count, the total weight of vertex cover, cardinality of vertex cover multi-set, message traffic, and energy consumption. Move count plays a crucial role in the wireless sensor network because a node must send their new state to its neighbor after a move, which affects the message complexity of wireless media. Step complexity is vital to see how long it takes to reach a stable algorithm configuration. Message traffic and energy consumption are another important metrics to measure lifetime of the networks. Weight and cardinality of the vertex cover are the other important metrics to facilitate when comparing the algorithms since we want to formulate a desirable solution in the shortest possible time. Although Kinawa's and Turau's algorithms were proposed for the un-weighted graphs, we carried out weighted experiments to compare all algorithms.

Thereafter, Turau's basic and improved algorithms and Kiniwa's algorithm will be called TURAU1, TURAU2, and KINIWA respectively. The transformed version of the SS-CVC2 algorithm is called as T-SS-CVC2.

Algorithm 6.9 Additional rules

```
process $u$
R1: $covered_u = 1 \wedge nex_u \neq \lceil \frac{|N(u)|}{cap_u} \rceil$
    $\Rightarrow nex_u := \lceil \frac{|N(u)|}{cap_u} \rceil$
R2: $covered_u = 0 \wedge nex_u \neq 0$
    $\Rightarrow nex_u := 0$
```

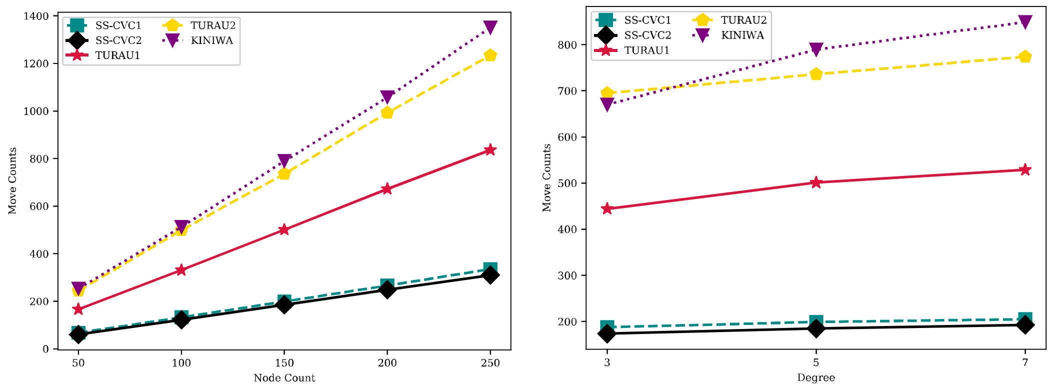
6.3.3 Performance Results

Move counts of the algorithms are shown in Figure 6.6 on the fixed average degree and fixed graph size. It is seen clearly that move counts increase with the number of the nodes in the graph for each algorithm as shown in Figure 6.6a. KINIWA algorithm makes the maximum number of moves to reach the stable configuration and it is followed by TURAU2. In the graphs including 250 vertices, KINIWA made 1351 moves to stabilize while our proposed algorithms SS-CVC1 and SS-CVC2 made 335 and 309 moves on the same graph. The closest algorithm to our algorithm regarding move count is TURAU1, which makes 884 moves until stabilizing on 250 sized graphs.

The density of the graph does not significantly affect the move counts of the algorithm as seen in Figure 6.6b. Especially, SS-CVC1 and SS-CVC2 have been affected as minimum as by density as opposed their counterparts. SS-CVC2 algorithm needs 173, 184, and 192 moves to stabilize on graphs that have 3, 5, and 7 average degrees. KINIWA needs 669, 789, and 848 moves on the same types of graphs.

Step count of SS-CVC2 varies between 40-60 while SS-CVC1 has lower than 33 step counts for all graph types as shown in Figure 6.7a. KINIWA needs 80-140 steps to become stabilized for each size of the graph. The algorithm which has the nearest step count to our algorithm is TURAU1 with 46-71 steps. Compared with TURAU1, The SS-CVC1 and SS-CVC2 algorithms need 1.10 and 2.15 times less steps to stabilize. SS-CVC1 and SS-CVC2 are 4.22 and 2.17 times faster than KINIWA which has the most step size for all graph sizes.

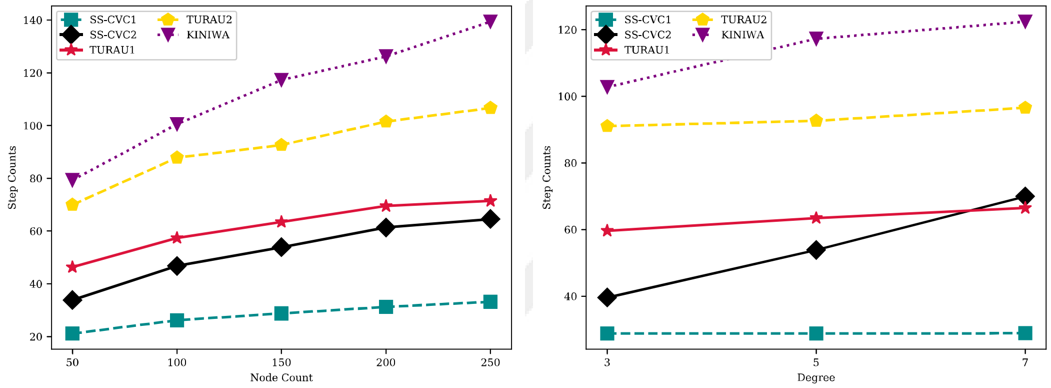
As seen in Figure 6.7b, the step count of SS-CVC1 has stayed stable on 28 as the density of the graph increased while the SS-CVC2's step count has increased with the density. On the graphs with the average degree of 7, SS-CVC2 algorithm



(a) Move count against node count on the fixed avarage degree 5.

(b) Move count against node count on the fixed graph size 150.

Figure 6.6. Move counts of algorithms on weighted graphs.



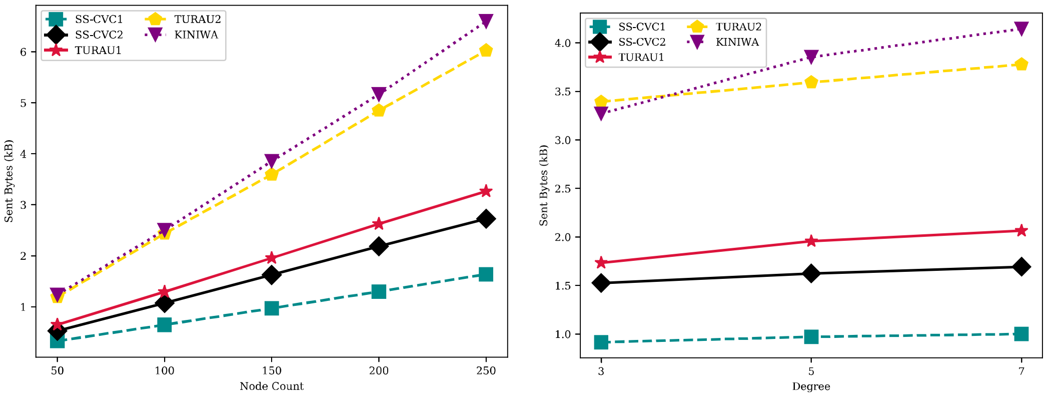
(a) Step count against node count on the fixed average degree 5.

(b) Step count against node count on the fixed graph size 150.

Figure 6.7. Step counts of algorithms on the weighted graphs.

exceeds the TURAU1 algorithm slightly. However, in the most graph types, SS-CVC1 and SS-CVC2 algorithms over-performed other algorithms in terms of step count.

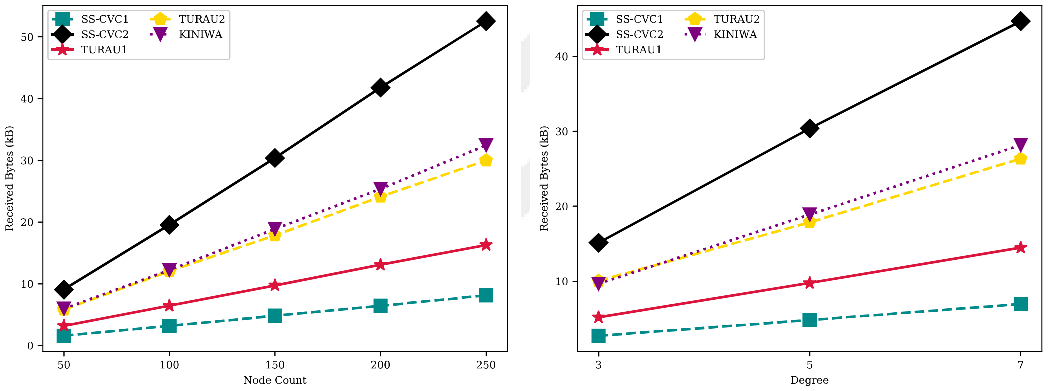
Figure 6.8 depicts sent byte for each algorithm on the whole network in kilobyte. SS-CVC1 and SS-CVC2 algorithms need the lowest message passing traffics related to their move count performance which directly impacts message traffic because after each move nodes must inform their neighbors. Note that the other factor for sent byte performance is message size. For example, the message size for SS-CVC1 algorithm is 5 byte, while SS-CVC2 holds 9 byte for each package. Because of this difference, the slope of sent byte and move count line of SS-CVC2 stay stable, but sent byte exceed SS-CVC1 as seen in Figure 6.8a and 6.8b. Graph size directly influences sent byte of algorithms since a larger graph needs more move and mes-



(a) Sent byte against node count on the fixed average degree 5.

(b) Sent byte against node count on the fixed graph size 150.

Figure 6.8. Sent byte of algorithms on weighted graphs.



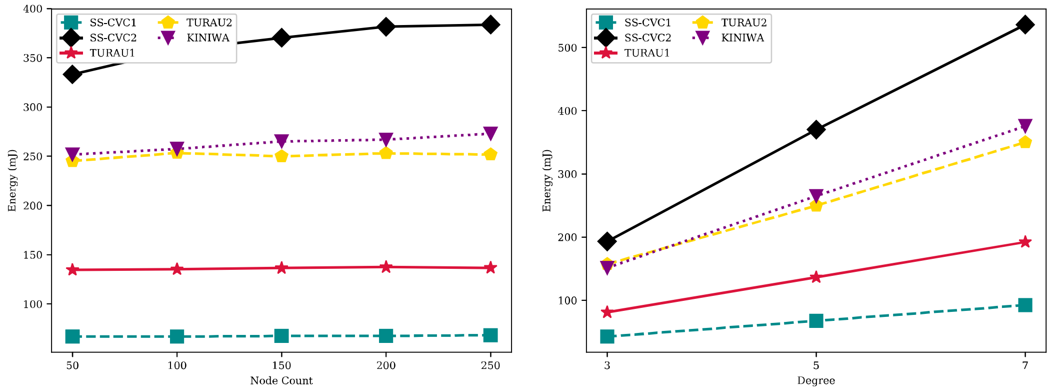
(a) Received byte against node count on the fixed average degree 5.

(b) Received byte against node count on the fixed graph size 150.

Figure 6.9. Received byte of algorithms on weighted graphs.

sage passing. For the graphs with 250 nodes, TURAU1 and TURAU2 and KINIWA send 3.26 kB, 6.02 kB and 6.59 kB messages in total while SS-CVC1 and SS-CVC2 algorithm needs 1.63 kB and 2.55 kB messages to stabilize. SS-CVC1 algorithm shows 2 times better performance against its closest competitor TURAU1. Graph density does not play a crucial role on sent byte performance of algorithm as seen in Figure 6.8b.

Received byte is another important measurement to determine algorithm quality because it affects the network lifetime. Figure 6.9 shows the performance of algorithms in terms of received byte until the system reaches to stable configuration. Figure 6.9a compares sent byte performance of algorithms with respect to graph size. SS-CVC1 algorithm over-performs all other algorithms since it has smaller packages to send and less message traffic. For example SS-CVC1 algorithm has 4.81 kB av-



(a) Energy consumption against node count on the fixed average degree 5.

(b) Energy consumption against node count on the fixed graph size 150.

Figure 6.10. Energy consumption of algorithms on weighted graphs.

erage received byte for graphs with 150 nodes, on the other hand KINIWA needs 18.88 kB to stabilize on same type of graphs. The best algorithm after SS-CVC1 is TURAU1 algorithm which needs 2 times more received byte in total. SS-CVC2 algorithm shows poor results since it assumes messages are passed to the 2-hop neighborhood.

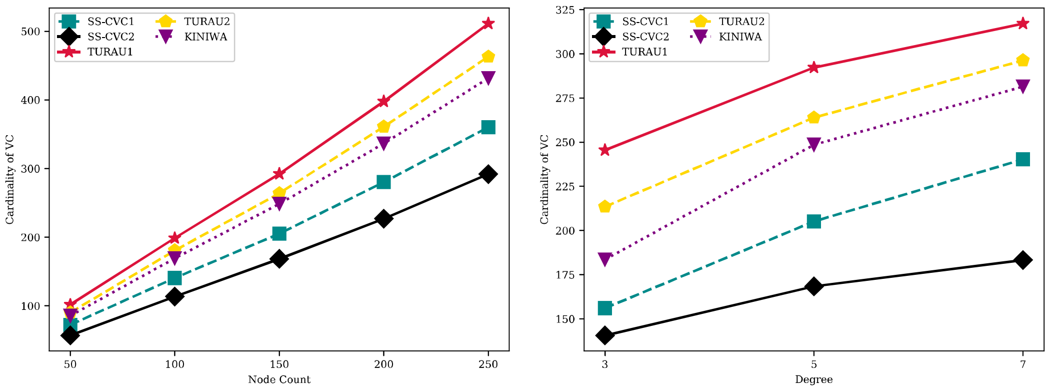
Figure 6.9b depicts the performance of algorithms when the average degree of graphs gets larger. Unlike the sent byte performance of algorithms, the density of graphs impacts received byte performances of algorithms. However, this impact is minimum for SS-CVC1 in comparison to the other algorithms as seen from slopes of lines. On graphs with 150 nodes, SS-CVC1 always stays under 7 kB for all density types.

Figure 6.10 illustrates the energy consumption of each individual node in the network. In order to calculate the energy consumption of nodes, we exploit Equation 5.1 that takes sent byte count and received byte count as parameters. As seen in Figure 6.10a, SS-CVC1 algorithm consumes 67.44 mJ per node as the most energy-efficient algorithm among all implemented algorithms. Since the energy calculation takes into account the received byte count, SS-CVC2 algorithm consumes more energy than its counterparts. After the SS-CVC1 algorithm, TURAU1 consumes 136.08 mJ per node to reach the stable configuration. It is obvious that SS-CVC1 algorithm solves the capacitated vertex cover problem 2 times efficiently in comparison to its nearest counterpart.

Figure 6.10b shows impacts of graph density on energy consumption for algorithms. We see exactly the same graphic as 6.9b except for numbers, due to received byte dominates the energy consumption calculation. But still density affects less SS-CVC1 than others since its energy consumption increase lower than 1.6 times for each average degree. Besides, TURAU1's energy consumption increase 1.7 times for each average degree but its energy consumption always higher than SS-CVC1. SS-CVC2 algorithm is the most energy consumer algorithm among all implemented algorithms and its slop is higher than others'.

Figures 6.11a and 6.11b show that the cardinality of VC solution is directly affected by the graph size and density of the graph. Intuitively, when the graph gets larger and denser we must choose more vertices to cover all edges in the graph because edge count increases with the number of the nodes and the average degree of the graph. SS-CVC2 algorithm produce the best VC solution for all graph sizes and densities. SS-CVC2 produced VC multi-set that contains 291 vertices on the 250 sized graphs. After the SS-CVC2 algorithm, the SS-CVC1 algorithm comes with a multi-set which contains 360 vertices on the graphs which contain 250 vertices. The SS-CVC2 algorithm produces 1.75, 1.59, and 1.48 times smaller vertex cover solutions in comparison to TURAU1, TURAU2, and KINIWA.

As the density of the graph rises, SS-CVC2 algorithm is less affected by the density when we compare it with the other algorithms. On the 150 sized graphs, SS-CVC2 produces 140, 168, and 183 sized VC solutions for the graphs that have 3, 5, and 7 average degrees while the best matching based algorithm KINIWA produces 183, 248, 281 sized VC solutions.

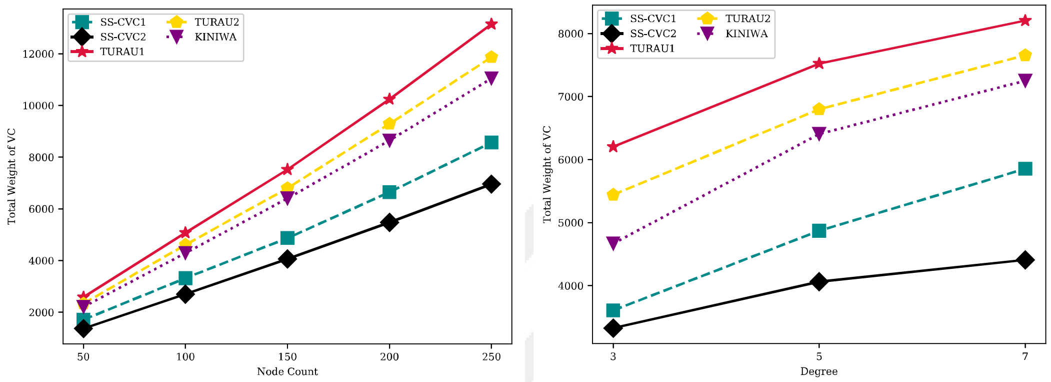


(a) VC size against node count on the fixed average degree 5.

(b) VC size against node count on the fixed graph size 150.

Figure 6.11. VC size of algorithms on the weighted graphs.

When we look at Figure 6.12, it can be clearly seen that the weight of VC has the same characteristic as the cardinality of VC multi-set because the weight of the solution is a function of the cardinality. SS-CVC1 and SS-CVC2 algorithms produce vertex cover solutions which have less weight than the other algorithms. SS-CVC1, KINIWA, TURAU2, and TURAU1 produced weighted vertex covers which have 1.23, 1.58, 1.70, and 1.88 times higher than SS-CVC2's solutions for 250 sized graphs. The density of graphs impacts the weight of the solution as seen in Figure 6.12 for all algorithms, however, our proposed algorithms SS-CVC1 and SS-CVC2 produced lower weighted solutions for all density in our experiments. Note that the accrual of the weight for SS-CVC2 is lowest among all other implemented algorithms as seen in graphic.



(a) VC weight against node count on the fixed average degree 5.

(b) VC weight against node count on the fixed graph size 150.

Figure 6.12. VC Weight of algorithms on the weighted graphs.

We elaborate the weighted approximation ratios of algorithms in Tables 6.1, 6.2, 6.3. Accordingly, we obtain the optimal solution in SageMath programming language by implementing Guha's integer linear programming algorithm proposed in (Guha et al., 2003).

Tables show us that the SS-CVC2 algorithm has better approximation ratios among all algorithms. After the SS-CVC2 algorithm, the second-lowest approximation belongs to the SS-CVC1 for all graph types. Matching based vertex cover algorithms produced more than 2 approximation ratios. Since these algorithms have not been proposed for the weighted graphs, they exceeded the theoretical approximation ratios.

The density of the graphs affects the approximation ratios of all algorithms while the graph size has not any correlation to the approximation ratio.

We provide the move count and step count performances of the transformer proposed by Turau in Figure 6.13. The transformer provides an interface that enables to execute an algorithm which is designed for 2-hop, with 1-hop information with

Table 6.1. Approximation ratios of the algorithms (Avg. degree = 3)

Size	SS-CVC2	SS-CVC1	KINIWA	TURAU2	TURAU1
50	1.55	1.71	2.17	2.50	2.88
100	1.53	1.65	2.16	2.48	2.90
150	1.56	1.69	2.19	2.56	2.92
200	1.56	1.71	2.18	2.54	2.92

Table 6.2. Approximation ratios of the algorithms (Avg. degree = 5)

Size	SS-CVC2	SS-CVC1	KINIWA	TURAU2	TURAU1
50	1.62	2.03	2.61	2.77	3.06
100	1.61	1.98	2.56	2.75	3.03
150	1.61	1.93	2.54	2.70	2.99
200	1.62	1.96	2.55	2.74	3.03

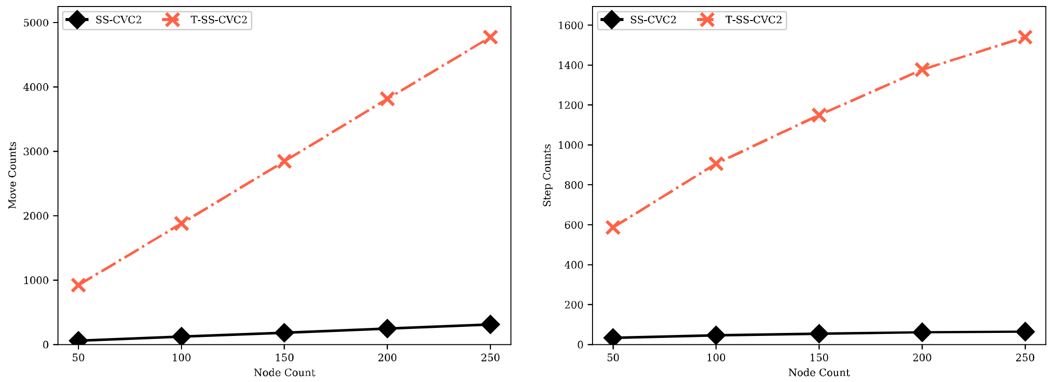
$\mathcal{O}(m)$ slow-down factor. The move count and step count of transformed SS-CVC2 (namely T-SS-CV2) are always higher than SS-CVC2 due to slow-down factor. T-SS-CVC2 needs to make moves at least 15.4 times more than SS-CVC2 to reach the minimal capacitated vertex cover solution since SS-CVC2 algorithm stabilizes after 309 moves while T-SS-CVC2 algorithm needs 4773 moves to reach the stable configuration on the 250 sized graph. We could infer the same regarding the step count which is 15 times more for the T-SS-CVC2 on the 50 sized graph.

To reach a stable configuration T-SS-CVC2 needs more message traffic as depicted in 6.14. As we have said before, message sending is tightly related to move count of algorithm. Nodes on T-SS-CVC2 send 15 times more byte and receive 4 times more message byte in comparison with SS-CVC2.

In terms of weight of vertex cover, SS-CVC2 and T-SS-CVC2 algorithms produced the same results as seen in Figure 6.15b but T-SS-CVC2 needs 5 times more energy to produce the same results in average.

Table 6.3. Approximation ratios of the algorithms (Avg. degree = 7)

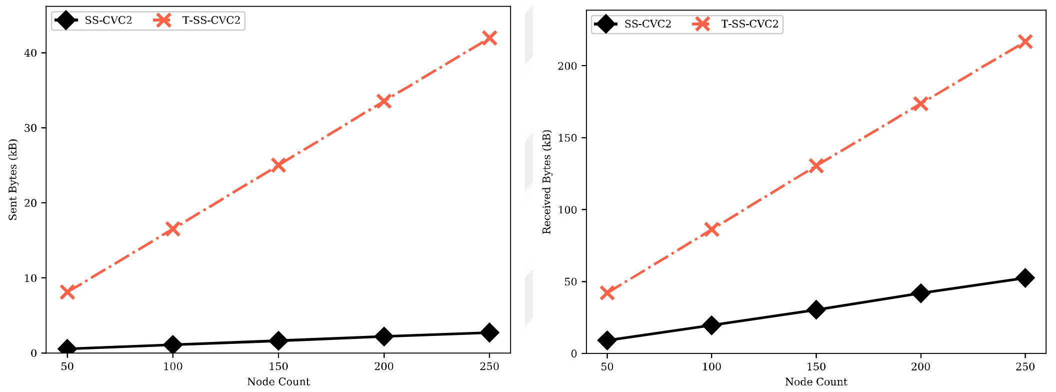
Size	SS-CVC2	SS-CVC1	KINIWA	TURAU2	TURAU1
50	1.67	2.23	2.78	2.99	3.18
100	1.66	2.18	2.74	2.84	3.07
150	1.63	2.16	2.68	2.83	3.03
200	1.62	2.11	2.67	2.78	3.01



(a) Move count against node count on the fixed average degree 5.

(b) Step count against node count on the fixed average degree 5.

Figure 6.13. Move and step count performances of T-SS-CVC2 algorithm against to SS-CVC2.



(a) Sent byte against node count on the fixed average degree 5.

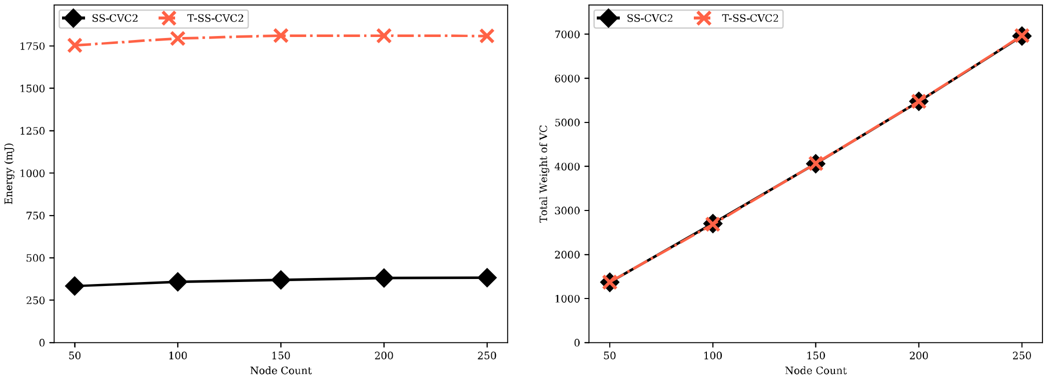
(b) Received byte against node count on the fixed average degree 5.

Figure 6.14. Sent and received byte performances of T-SS-CVC2 algorithm against to SS-CVC2.

6.4 Conclusion

In this chapter, we have proposed two capacitated vertex cover algorithms that run on the self-stabilizing setting. We have modified Ikeda's algorithm by adding two new rules and changing the existing rules. Also, we have proposed a new algorithm based on greedy heuristic. We analyzed these two algorithms and have shown theoretical step complexities that are $\mathcal{O}(n^2)$ and $\mathcal{O}(n)$ under the unfair scheduler.

We have evaluated the performance of our algorithms and compared the results with the existing algorithms in the literature. The experimental results show that SS-CVC2 and SS-CVC1 algorithms over-performed the existing matching based algorithms (by modifying them) in step count, round count, and cardinality of vertex cover metrics. SS-CVC1 algorithm needs the lowest message traffic as opposed to



(a) Energy consumption against node count on the fixed average degree 5.

(b) VC weight against node count on the fixed average degree 5.

Figure 6.15. Energy consumption and solution performances of T-SS-CVC2 algorithm against to SS-CVC2.

SS-CVC2 in which nodes receive more messages than other algorithms. Since the message traffic directly affects energy usage, the SS-CVC2 algorithm is the one with the most energy need. However, the approximation ratio of SS-CVC2 is not greater than 1.7 for all graph types, while the matching-based algorithm produces at least 2 times bigger than the optimal solution. In addition to these, we have provided performance results of the transformed version of SS-CVC2 which needs more time to stabilize under unfair scheduler but produce the same solution as SS-CVC2.

As a conclusion, we can state that the SS-CVC2 algorithm is better than the others in terms of VC and execution time performance when 2-hop information is provided. Yet, if it is not provided, the SS-CVC1 algorithm is another option to find the capacitated vertex cover rather than matched based vertex cover algorithms with the advantage of lowest energy consumption. Using the transformer provides the same result in terms of the vertex cover, but it does take longer than SS-CVC2.

7. CONCLUSION

In this chapter, we conclude the thesis and present future works about the thesis.

7.1 Summary

In this thesis, we make an extensive performance evaluation about vertex cover problems from the central setting to the self-stabilizing setting. The performance evaluations include the following titles:

- Sequential vertex cover algorithms
- Linear-programming vertex cover algorithms
- Distributed vertex cover algorithms
- Self-stabilizing vertex cover algorithms

The performance evaluation of sequential algorithms shows that the Greedy algorithm is the best candidate to construct a vertex cover in a graph with feasible running time and vertex cover solution. Finding an exact solution is hard to get in most situations because of the time constraint. Therefore, some intuitive methods like the greedy approach give a feasible solution most of the time if an exact solution is not necessary.

For the capacitated vertex cover algorithm in linear programming setting, there is a trade-off between Naor 8 and Guha's algorithm. If a faster algorithm with the right solution is required, we should choose Guha's algorithm. Nonetheless, if it is required that a better result with acceptable running time, we can choose Naor 8 algorithm to solve our capacitated vertex cover problem. Gandhi's pre-processing step does not give success as promised despite its longer execution times. Guha's 3 approximation algorithm gives fine results in all aspects, but it is not better than Guha 8. We conclude that the theory does not match the practice findings at every time, and sometimes a theoretically weak algorithm could give the best performance among its opponents.

We evaluated performances of distributed vertex cover algorithms in the TOSSIM simulator and on IRIS motes for test-bed experiment. VECO algorithms remained consistent with increasing of node count and degree as expected. The result shows that the cardinalities of the detected VCs of the proposed algorithms are lower than the other algorithms except Greedy. VECO1 and VECO2 algorithms produced good results for energy efficiency and running time, albeit slightly behind the Greedy algorithm in terms of the detected VC algorithm. However, as the sizes of the graphs grow, the difference reduces in favor of VECO algorithms. Hoepman algorithm gives poor results in terms of energy consumption but also gives poor results in terms of cardinality of VC. Since Kavalci is designed asynchronously with unicast messaging, VECO algorithms over-performed it in all performance parameters. Considering the trade-off between energy consumption and optimization, the proposed algorithms provide a feasible and more efficient approach for finding the VCs with BFST in WSNs. We saw that we should combine the two algorithms to get a quicker and feasible solution.

In most of the tests, self-stabilizing IS algorithms have outperformed VC algorithms. Firstly, IS algorithms provided better VC solutions as they find around 15% smaller sets that Turau's improvement algorithm and Kiniwa's algorithm, and 35% smaller sets than that of Turau's basic algorithm. Secondly, IS algorithms that are designed to run under distributed scheduler (T-MIS, G-MIS, I-MIS) converged to a solution in significantly less number of rounds than VC algorithms. The quickest algorithm is I-MIS with a running-time of 9 rounds, while the quickest VC algorithm has required 14 rounds on average. Finally, considering the total number of moves required for an algorithm to converge, none of the VC algorithms could outperform any IS algorithms. These results suggest that self-stabilizing IS algorithms can be more efficient than VC algorithms in providing vertex cover sets for link monitoring in wireless sensor networks. We conclude that changing the way to solve problems in some situations like restricted areas could be beneficial in terms of efficiency.

By using this information, we have proposed two novel self-stabilizing capacitated vertex cover algorithms that combine two different paradigms. SS-CVC1 algorithm is inspired by Ikeda's MIS algorithm and SS-CVC2 is our distance-2 knowledge algorithm that uses the expression model proposed by Turau. After a detailed explanation of these algorithms, we provided a theoretical analysis of them. Due to the absence of algorithms to compare with our algorithms, we modified self-stabilizing vertex cover algorithms to construct capacitated vertex cover solutions. Both proposed algorithms over-performed their counterparts. SS-CVC2 solved the problem at most 1.7 approximation ratio for all graph types, while the other matching-based algorithms produced at least 2 times bigger than the optimal solution (for the weighted setting). Because of the SS-CVC2 algorithm needs distance-2 knowledge, we compared the performance of SS-CVC2 and its transformed version T-SS-CVC2 which uses a transformer proposed by Turau. This comparison showed that using a transformer needs more time and energy to stabilize the algorithm since it adds extra rules, yet it has produced the same vertex cover performance.

7.2 Future Works

The capacity constraint is not only applied to the vertex cover problem, but it can be applied to other well-known graph-theoretical algorithms such as independent set, matching, and dominating set. We touched on the independent set and matching technique since they have a close relationship with the vertex cover problem. We could apply our inferences to these problems.

We made simulations of our proposed algorithm on a simulator written in Python language. It could be turned into test-bed experiments on IRIS motes. However, to achieve this, an infrastructure that provides distance-2 knowledge on wireless networks is essential. We can implement a transformed version of SS-CVC2 on IRIS motes as well. We will work on reducing the energy consumption, as well, of the SS-CVC2 algorithm by reducing its message traffic.

We would evaluate other generalized versions of vertex cover problems such as the connected vertex cover problem to build a backbone over the given network. Our performance evaluations would be improved by adding new algorithms and new comparison techniques like the maximum lifetime of the network in distributed settings.

The main focus of this thesis is WSNs, but it could be extended to other research trends such as complex networks, IoT, and cloud computing, which have close relationships with distributed algorithms. For example, the findings of this dissertation can be extended and implemented to UAV networks to handle coverage and connectivity issues since drone networks attract a lot of researchers from different areas. The proposed algorithms could be improved for the purpose of using on devices which have multiple antennas, namely MIMO antennas, to manage capacity constraint by assigning a predefined number of edge to cover on each antenna.

REFERENCES

Afek, Y. and Brown, G.M., 1993. Self-stabilization over unreliable communication media. *Distributed Computing*, 7(1):27–34.

Akyildiz, I., Su, W., Sankarasubramaniam, Y. and Cayirci, E., 2002. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422.

Angluin, D., 1980. Local and global properties in networks of processors (Extended Abstract). *Proceedings of the twelfth annual ACM symposium on Theory of computing - STOC '80*, p. 82–93. URL `http://portal.acm.org/citation.cfm?id=804655`.

Arora, S., Chakaravarthy, V.T., Gupta, N., Mukherjee, K. and Sabharwal, Y., 2013. Replica Placement via Capacitated Vertex Cover. In IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013), A. Seth and N.K. Vishnoi, eds.. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, volume 24 of *Leibniz International Proceedings in Informatics (LIPIcs)*, p. 263–274. URL `http://drops.dagstuhl.de/opus/volltexte/2013/4378`.

Bar-Yehuda, R. and Even, S., 1985. A Local-Ratio Theorem for Approximating the Weighted Vertex Cover Problem. *North-Holland Mathematics Studies*, 109(C):27–45.

Beauquier, J., Datta, A.K., Gradinariu, M. and Magniette, F., 2000. Self-stabilizing local mutual exclusion and daemon refinement. In International Symposium on Distributed Computing. Springer, p. 223–237.

Bin, T. and Jiang, C., 2018. A cost-sensitive method for wireless sensor network min beacon node set construction. *Cluster Computing*. URL `https://doi.org/10.1007/s10586-017-1418-y`.

Chen, N., 2009. On the Approximability of Influence in Social Networks. *SIAM Journal on Discrete Mathematics*, 23(3):1400–1415. URL `https://doi.org/10.1137/08073617X`.

Chuzhoy, J. and (Seffi) Naor, J., 2006. Covering Problems with Hard Capacities. *SIAM Journal on Computing*, 36(2):498–515. URL `https://doi.org/10.1137/S0097539703422479`.

Cormen, T., 2001. Introduction to Algorithms. MIT Press, Cambridge, Mass, 2nd ed edition, 1203 p.

REFERENCES (continued)

Dagdeviren, O. and Akram, V.K., 2017. PACK: Path coloring based k-connectivity detection algorithm for wireless sensor networks. *Ad Hoc Networks*, 64:41 – 52. URL http://www.sciencedirect.com/science/article/pii/S1570870517301063.

Dijkstra, E.W., 1974. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644. URL http://portal.acm.org/citation.cfm?doid=361179.361202.

Dinur, I. and Safra, S., 2002. The Importance of Being Biased. In Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing. ACM, New York, NY, USA, STOC '02, p. 33–42.

Dinur, I. and Safra, S., 2005. On the hardness of approximating minimum vertex cover. *Annals of mathematics*, p. 439–485.

Dolev, S., 2000. Self-stabilization. MIT press.

Dolev, S., Israeli, A. and Moran, S., 1991. Resource Bounds for Self Stabilizing Message Driven Protocols. p. 281–293.

Dubois, S. and Tixeuil, S., 2011. A taxonomy of daemons in self-stabilization. *arXiv preprint arXiv:1110.0334*.

Erciyes, K., 2013. Distributed Graph Algorithms for Computer Networks. Computer Communications and Networks. Springer London, London, 217–228 p.

Feng, X., Zhang, X., Zhang, J. and Muhdhar, A.A., 2018. A coverage hole detection and repair algorithm in wireless sensor networks. *Cluster Computing*. URL https://doi.org/10.1007/s10586-017-1665-y.

Filiol, E., Franc, E., Gubbioli, A., Moquet, B. and Roblot, G., 2007. Combinatorial Optimisation of Worm Propagation on an Unknown Network. *World Acad. Sci. Eng. Technol.*, 23(34):41–47.

Gairing, M., Goddard, W., Hedetniemi, S., Kristiansen, P. and Mcrae, A., 2004. Distance-two information in self-stabilizing algorithms. *Parallel Processing Letters*, 14:387–398.

Gandhi, R., Halperin, E., Khuller, S., Kortsarz, G. and Srinivasan, A., 2003. An Improved Approximation Algorithm for Vertex Cover with Hard Capacities. *Journal of Computer and System Sciences*, 72(1):164–175.

REFERENCES (continued)

Goddard, W., Hedetniemi, S.T., Jacobs, D.P. and Srimani, P.K., 2003. Self-stabilizing protocols for maximal matching and maximal independent sets for ad hoc networks. In Parallel and Distributed Processing Symposium, 2003. Proceedings. International. IEEE, p. 14—-pp.

Goddard, W., Hedetniemi, S.T., Jacobs, D.P. and Trevisan, V., 2008. Distance-k knowledge in self-stabilizing algorithms. *Theoretical Computer Science*, 399(1):118 – 127. URL `http://www.sciencedirect.com/science/article/pii/S0304397508001151`, structural Information and Communication Complexity (SIROCCO 2006).

Gowrishankar, S., Basavaraju, T.G., Manjaiah, D.H. and Sarkar, S.K., 2008. Issues in Wireless Sensor Networks. In Proceedings of the World Congress on Engineering 2008 Vol I, S.I. Ao, International Association of Engineers and International Conference of Wireless Networks ; (London) : 2008.07.02-04, eds.. IAENG, Hong Kong, volume I of *World Congress on Engineering, WCE 2008*, p. 978–988.

Grandoni, F., 2004. Exact Algorithms for Hard Graph Problems (Algoritmi Esatti per Problemi Difficili su Grafi). *Citeseer*. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.107.100&rep=rep1&type=pdf`.

Grandoni, F., Könemann, J. and Panconesi, A., 2008. Distributed Weighted Vertex Cover via Maximal Matchings. *ACM Trans. Algorithms*, 5(1):6:1—-6:12. URL `http://doi.acm.org/10.1145/1435375.1435381`.

Guellati, N. and Kheddouci, H., 2010. A Survey on Self-stabilizing Algorithms for Independence, Domination, Coloring, and Matching in Graphs. *J. Parallel Distrib. Comput.*, 70(4):406–415. URL `http://dx.doi.org/10.1016/j.jpdc.2009.11.006`.

Guha, S., Hassin, R., Khuller, S. and Or, E., 2003. Capacitated vertex covering. *Journal of Algorithms*, 48(1):257–270.

Halperin, E., 2002. Improved Approximation Algorithms for the Vertex Cover Problem in Graphs and Hypergraphs. *SIAM Journal on Computing*, 31(5):1608–1623. URL `http://epubs.siam.org/doi/10.1137/S0097539700381097`.

Hanckowiak, M., Karonski, M. and Panconesi, A., 2001. On the Distributed Complexity of Computing Maximal Matchings. *SIAM Journal on Discrete Mathematics*, 15(1):41–57.

REFERENCES (continued)

Håstad, J. and Johan, 2001. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859. URL `http://portal.acm.org/citation.cfm?doid=502090.502098`.

Hauck, B., 2012. Time- and space-efficient self-stabilizing algorithms. doctoralthesis, Technische Universität Hamburg. URL `http://tubdok.tub.tuhh.de/handle/11420/1093`.

Hoepman, J.H., 2004. Simple distributed weighted matchings. *arXiv preprint cs/0410047*.

Ikeda, M., Kamei, S. and Kakugawa, H., 2002. A Space-Optimal Self-Stabilizing Algorithm for the Maximal Independent Set Problem.

Ileri, C.U. and Dagdeviren, O., 2018. Evaluating Fault Tolerance Properties of Self-Stabilizing Matching Algorithms in Wireless Sensor Networks. In 2018 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom). p. 1–5.

Ileri, C.U., Ural, C.A., Dagdeviren, O. and Kavalci, V., 2016a. On Vertex Cover Problems in Distributed Systems. In Advanced Methods for Complex Network Analysis, chapter 1, pp. 461.

Ileri, C.U., Ural, C.A., Dagdeviren, O. and Kavalci, V., 2016b. On Vertex Cover Problems in Distributed Systems, IGI Global, chapter 1, p. 1–29.

Karakostas, G., 2009. A better approximation ratio for the vertex cover problem. *ACM Transactions on Algorithms*, 5(4):1–8.

Karp, R., 1972. Reducibility among combinatorial problems. In Complexity of Computer Computations, R. Miller and J. Thatcher, eds., Plenum Press, p. 85–103.

Kavalci, V., Ural, A. and Dagdeviren, O., 2014. Distributed Vertex Cover Algorithms for Wireless Sensor Networks. *International journal of Computer Networks & Communications*, 6(1):95–110. URL `http://www.airccse.org/journal/cnc/6114cnc07.pdf`.

Kavitha, R.J. and Caroline, B.E., 2017. Secured and reliable data transmission on multi-hop wireless sensor network. *Cluster Computing*. URL `https://doi.org/10.1007/s10586-017-1227-3`.

Kiniwa, J., 2005. Approximation of Self-stabilizing Vertex Cover Less Than 2. In Self-Stabilizing Systems, S. Tixeuil and T. Herman, eds.. Springer Berlin Heidelberg, Berlin, Heidelberg, p. 171–182.

REFERENCES (continued)

Kumar, R. and Kaur, J., 2004. Efficient beacon placement for network tomography. *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement - IMC '04*, pp. 181. URL `http://portal.acm.org/citation.cfm?doid=1028788.1028810`.

Lancia, G., Bafna, V., Istrail, S., Lippert, R. and Schwartz, R., 2001. SNPs Problems, Complexity, and Algorithms, Springer Berlin Heidelberg, Berlin, Heidelberg, p. 182–193.

Levis, P., Lee, N., Welsh, M. and Culler, D., 2003. TOSSIM: accurate and scalable simulation of entire TinyOS applications. p. 126–137.

Levis, P., Madden, S., Polastre, J., Szewczyk, R., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E. and Culler, D., 2004. TinyOS: An operating system for sensor networks. In in Ambient Intelligence. Springer Verlag.

Memsic, 2011. IRIS Sensor Node Module. p. 1–2. URL `http://www.memsic.com/userfiles/files/datasheets/wsn/iris_datasheet.pdf`.

Niedermeier, R. and Rossmanith, P., 2003. On efficient fixed-parameter algorithms for weighted vertex cover. *Journal of Algorithms*, 47(2):63–77.

Panconesi, A. and Rizzi, R., 2001. Some simple distributed algorithms for sparse networks. *Distributed Computing*, 14(2):97–100. URL `http://link.springer.com/10.1007/PL00008932`.

Parnas, M. and Ron, D., 2007. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1-3):183–196. URL `http://linkinghub.elsevier.com/retrieve/pii/S0304397507003696`.

Polishchuk, V. and Suomela, J., 2009. A simple local 3-approximation algorithm for vertex cover. *Information Processing Letters*, 109(12):642–645.

Savage, C., 1982. Depth-first search and the vertex cover problem. *Information Processing Letters*, 14(5):233–235. URL `https://www.sciencedirect.com/science/article/pii/0020019082900229`.

Schult, D.A. and Swart, P., 2008. Exploring network structure, dynamics, and function using NetworkX. In Proceedings of the 7th Python in Science Conferences (SciPy 2008). volume 2008, p. 11–16.

Shukla, S.K., Rosenkrantz, D.J. and Ravi, S.S., 1995. Observations on Self-Stabilizing Graph Algorithms 1 Introduction. 1(518):0–15.

REFERENCES (continued)

Tamura, H., Sugawara, H., Sengoku, M. and Shinoda, S., 2001. Multiple cover problem on undirected flow networks. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 84(1):67–74.

Tixeuil, S., 2009. Self-stabilizing Algorithms. In Algorithms and theory of computation handbook, Chapman & Hall/CRC, p. 26.1–26.45. URL `https://hal.sorbonne-universite.fr/hal-00569219`, 50 pages.

Turau, V., 2007. Linear self-stabilizing algorithms fotr the independent and dominating set problems using an unfair distributed scheduler. *Information Processing Letters*, 103(3):88–93. URL `http://www.sciencedirect.com/science/article/pii/S0020019007000488`.

Turau, V., 2012. Efficient transformation of distance-2 self-stabilizing algorithms. *Journal of Parallel and Distributed Computing*, 72(4):603 – 612. URL `http://www.sciencedirect.com/science/article/pii/S0743731511002486`.

Turau, V. and Hauck, B., 2011. A fault-containing self-stabilizing (3-2/Δ+1)-approximation algorithm for vertex cover in anonymous networks. *Theoretical Computer Science*, 412(33):4361–4371. URL `http://dx.doi.org/10.1016/j.tcs.2010.11.010`.

Turau, V. and Weyer, C., 2006. Randomized Self-stabilizing Algorithms for Wireless Sensor Networks. In Self-Organizing Systems, H. de Meer and J.P.G. Sterbenz, eds.. Springer Berlin Heidelberg, Berlin, Heidelberg, p. 74–89.

Yigit, Y., 2016. Graph-theoretic Topology Control Algorithms for Wireless Sensor Networks.

Yigit, Y., Ileri, C.U. and Dagdeviren, O., 2018. Fault tolerance performance of self-stabilizing independent set algorithms on a covering-based problem: The case of link monitoring in WSNs. In 2018 5th International Conference on Electrical and Electronic Engineering (ICEEE). p. 423–427.

ACKNOWLEDGEMENT

I would like to thank my supervisor Assoc. Prof. Dr. Orhan Dagdeviren who supported and directed me since I had registered to International Computer Institute.

I would like to thank Prof. Dr. Serdar Korukoglu and Assoc. Prof. Dr. Hasan Bulut for their valuable feedbacks and comments during the thesis monitoring term.

This thesis is part of the ARDEB 1001 project which was supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) with project serial number 215E115. I would like to thank TÜBİTAK for their valuable support.

I would like to thank all academic, managerial, and technical staff of International Computer Institute and Network Science Research Group (NEToS) for providing me with a comfortable research environment.

I express my deepest thankfulness to my family members who always support me in every situation.

CURRICULUM VITAE

Yasin YİĞİT

Mobile: +90 554 135 75 45
E-mail: yasin.yigit@hotmail.com.tr

Education

PhD: 2016-2020, Ege University, International Computer Institute
MSc: 2013-2016, Ege University, International Computer Institute
BSc: 2009-2013, Ege University, Agricultural Engineering

Experience

2020 - : Software Developer, Kronnika, Izmir, Turkey
2019 - 2020 : Software Developer, Zenkronn Bilisim, Izmir, Turkey
2016 - 2018 : Researcher at TUBITAK Project, International Computer Institute, Ege University, Izmir, Turkey

Foreign Languages

English : Upper Intermediate (Yokdil: 83.75)
German: Beginner

Projects

2016-2018 : *Distributed and self-stabilizing capacitated graph theoretical algorithms*. Funded by TUBITAK (215E115). Role: Researcher
2015-2018 : *Integrated graph-theoretical algorithms for complex wireless netoworks*. Funded by EGE BAP (15-UBE-002) Role: Researcher
2017-2019 : *A study on the model-driven development of internet of things software based on contiki operating system* Funded by EGE BAP (17-UBE-002) Role: Team Member

Publications

Journal Papers:

Suvari, M., Cabuk, U. C., Yigit, Y., and Dagdeviren, O., 2018. An Android-based Microprocessor Programmer Software for Embedded Systems, *International Journal of Informatics Technologies, Gazi University*. 11,4, 321-332.

International Conferences:

Yigit, Y., Ileri, C. U., and Dagdeviren, O., 2018. Fault tolerance performance of self-stabilizing independent set algorithms on a covering-based problem: The case of link monitoring in WSNs. in IEEE International Conference on Electrical and Electronic Engineering (ICEEE), 5th, pp.423-427.

National Conferences:

Yigit, Y., and Dagdeviren, O., 2018. Evaluation of fault tolerant link monitoring algorithms for wireless sensor networks (Turkish). in 26th Signal Processing and Communications Applications Conference (SIU), pp. 1-4.

Akram V. K., Yigit, Y., and Dagdeviren, O., 2018. Movement based connectivity restoration system for wireless sensor and actor networks (Turkish). in 26th Signal Processing and Communications Applications Conference (SIU), pp. 1-4.

Yigit, Y., and Dagdeviren, O., 2018. Performance evaluation of vertex cover algorithms (Turkish). in 20. Akademik Bilisim Konferansi, pp. 205,210.

Akram V. K., Yigit, Y., and Dagdeviren, O., 2018. Performance evaluation of k-connectivity detection algorithms on wireless sensor and actor networks (Turkish). in 20. Akademik Bilisim Konferansi, pp. 217,222.

Book Chapters:

Ileri, C. U., Yigit, Y., Arapoglu, O., Evcimen, H. T., Asci, M., and Dagdeviren, O., 2019. Capacitated Graph Theoretical Algorithms for Wireless Sensor Networks towards Internet of Things. *Handbook of Research on the IoT, Cloud Computing, and Wireless Network Optimization*. IGI Global.