1-1-2008

Self-stabilizing k-clustering in mobile ad hoc networks

Priyanka Vemula
University of Nevada, Las Vegas

Follow this and additional works at: https://digitalscholarship.unlv.edu/rtds

Repository Citation
https://digitalscholarship.unlv.edu/rtds/2365

This Thesis is brought to you for free and open access by Digital Scholarship@UNLV. It has been accepted for inclusion in UNLV Retrospective Theses & Dissertations by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.
SELF-STABILIZING K-CLUSTERING IN MOBILE AD HOC NETWORKS

by

Priyanka Vemula

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science Degree in Computer Science
School of Computer Science
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
August 2008
INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.
The Thesis prepared by

PRIYANKA VEMULA

Entitled

SELF-STABILIZING K-CLUSTERING IN MOBILE AD HOC NETWORKS

is approved in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

Examination Committee Chair

Dean of the Graduate College

Examination Committee Member

Examination Committee Member

Graduate College Faculty Representative
In this thesis, two silent self-stabilizing asynchronous distributed algorithms are given for constructing a $k$-clustering of a connected network of processes. These are the first self-stabilizing solutions to this problem. One algorithm, FLOOD, takes $O(k)$ time and uses $O(k \log n)$ space per process, while the second algorithm, BFS-MIS-CLSTR, takes $O(n)$ time and uses $O(\log n)$ space; where $n$ is the size of the network. Processes have unique IDs, and there is no designated leader. BFS-MIS-CLSTR solves three problems; it elects a leader and constructs a BFS tree for the network, constructs a minimal independent set, and finally a $k$-clustering. Finding a minimal $k$-clustering is known to be $\mathcal{NP}$-hard. If the network is a unit disk graph in a plane, BFS-MIS-CLSTR is within a factor of $O(7.2552k)$ of choosing the minimal number of clusters.

A lower bound is given, showing that any comparison-based algorithm for the $k$-clustering problem that takes $o(diam)$ rounds has very bad worst case performance.

**Keywords:** BFS tree construction, K-clustering, leader election, MIS construction, self-stabilization, unit disk graph.
# TABLE OF CONTENTS

ABSTRACT ................................................................. iii

LIST OF FIGURES ....................................................... vi

ACKNOWLEDGMENTS ................................................ vii

CHAPTER 1  INTRODUCTION ......................................... 1
  Contributions ......................................................... 1
  Outline ................................................................. 2

CHAPTER 2  WIRELESS NETWORKS ............................... 5
  Mobile Wireless Networks ........................................ 5
  Wireless Sensor Networks ....................................... 7
  Mobile Ad Hoc Networks ....................................... 11

CHAPTER 3  SELF-* SYSTEMS ........................................ 19
  Distributed Systems .............................................. 19
  Overview ............................................................ 20
  Self-Stabilizing Systems ....................................... 22

CHAPTER 4  CLUSTERING ............................................. 24
  Need for Clustering .............................................. 25
  Cost of Clustering ............................................... 26
  Classifying Clustering Schemes ............................. 27

CHAPTER 5  K-CLUSTERING ........................................ 31
  Amis et al. Algorithm ........................................... 31
  Fernandess and Malkhi Algorithm ........................ 32
  Related Work ....................................................... 33

CHAPTER 6  PRELIMINARIES ......................................... 35
  Self-Stabilization ................................................ 35

CHAPTER 7  THE ALGORITHM FLOOD ......................... 38
  Functions and Actions of FLOOD .......................... 40
  An Example Computation ..................................... 42
  Proofs for FLOOD ................................................ 46
  A Worst Case Example ......................................... 50
  A Lower Bound for Comparison Based Clustering Algorithms ........................................ 51
LIST OF FIGURES

7.1 An Example Network. ................................................................. 43
7.2 Sequence of Configurations Illustrating FLOOD. ................. 44
7.3 The Line Graph $L_{23,3}$. ......................................................... 50

8.1 (a) An Example Network. (b) The BFS Tree. ..................... 58
8.2 Sequence Illustrating BFS. Start Configuration is Clean. ........ 62
8.3 Sequence illustrating BFS. Start Configuration has Errors. .... 66
8.4 An $\Omega(n)$-Time Example. .................................................. 75
8.5 (a) BFS Tree. (b) MIS Tree (c). Steps of MIS Computation. .... 80
8.6 Values of $\alpha(x)$ for a Rooted Tree, where $k = 4$. .............. 86
8.7 Various Steps of BFS-MIS-CLSTR in an Example where $k = 4$. 90
8.8 Proof of Lemma 8.11. ............................................................. 94

10.1 $G_{7,5}$, a $(7,5)$-Cluster. ..................................................... 103
ACKNOWLEDGMENTS

I would like to express my sincere gratitude to Dr. Ajoy K. Datta for his guidance, support, encouragement, patience, and enthusiasm throughout my graduate study. I am indebted to Dr. Datta and Dr. Lawrence L Larmore for their technical contributions in this research.

I would also like to thank Dr. Yoohwan Kim and Dr. Emma Regentova for their time in reviewing my report and their willingness to serve on my committee.

My special gratitude goes to my family and friends, without whose continuous support and faith this work would not have been possible.
CHAPTER 1

INTRODUCTION

In this thesis, we present two self-stabilizing asynchronous distributed algorithms for the $k$-clustering problem in mobile ad hoc networks. Thus, this research covers several domains of distributed computing, viz. mobile ad hoc networks (discussed in Chapter 2), asynchronous algorithms (discussed in Chapter 3), self-stabilizing systems (discussed in Chapter 3), and clustering (discussed in Chapter 4).

1.1 Contributions

The existing solutions to $k$-clustering problem are not self-stabilizing [21, 23]. A self-stabilizing system, regardless of the initial states of the processes and initial messages in the links, is guaranteed to converge to the intended behavior in finite time. In this work, we present the first self-stabilizing solutions to the $k$-clustering problem.

Algorithm FLOOD, given in Chapter 7, is similar to that of Amis et al. [3]. FLOOD uses only $(2k + O(1)) \log_2 n$ bits per process, approximately half that of [3]. FLOOD is self-stabilizing and silent, and takes $3k + O(1)$ rounds.

Algorithm BFS-MIS-CLSTR, given in Section 8, consists of three modules, BFS, MIS, and CLSTR. Each module by itself is a contribution. The module BFS takes $O(n)$ rounds, and elects a leader and constructs a BFS tree of the network rooted at that leader. The module MIS takes at most $n$ additional rounds, and selects a maximal independent set of the network. The final module, CLSTR, constructs a $k$-clustering of the network.
The number of $k$-clusters constructed by BFS-MIS-CLSTR is $O\left(\frac{n}{k}\right)$ in any case, and within a factor of $\left(\frac{4\pi}{\sqrt{3}}k + O(1)\right)$ of the smallest possible number of clusters if the network is a unit disk graph in the plane. This result is an improvement over that of [29].

We say that an algorithm is *comparison-based* if the only operation it can use to distinguish two IDs is comparison. In contrast, an algorithm that examines individual bits of an ID is not comparison based. In Section 7.5, we prove that there is no competitive comparison-based self-stabilizing distributed asynchronous algorithm for the $k$-clustering algorithm which takes $o(diam)$ rounds, even if all processes have unique IDs.

1.2 Outline

In Chapter 2, we give an overview of the wireless networks; we describe both sensor networks and mobile ad hoc networks (MANET). As this research is on MANET, we present various characteristics, issues, and applications of this type of network. In Chapter 3, we describe different types of fault-tolerant systems within the common framework of self- systems; we also give an overview of the self-stabilizing systems.

The detailed introduction to clustering, including the cost, applications, and classifications, is presented in Chapter 4. In Chapter 5, we introduce the type of clustering researched in this thesis, namely $k$-clustering. We formally define the problem, and give a brief overview of two algorithms in the current literature that are similar to our algorithms; we also mention other related work.

In Chapter 6, we describe the model of computation used in the paper, formally define self-stabilization, and give some additional needed definitions.

In Chapter 7, we define the algorithm FLOOD. We first give an overview. In Section 7.1, we give the formal definition of FLOOD. In Section 7.2, we give the details of an ex-
ample computation of FLOOD, using the network shown in Figure 7.1. The same network will be used for example computations throughout the thesis. In Section 7.3, we prove the correctness and time complexity of FLOOD.

In Chapter 8, we define the algorithm BFS-MIS-CLSTR, which uses only $O(\log n)$ space per process, but takes $O(n)$ rounds to converge. This algorithm is described as the concatenation of three modules.

In Section 8.1, we give BFS, the first module. BFS is a leader election algorithm which elects the process of lowest ID as leader. It also constructs a BFS tree rooted at that process, which we call $Root$. In Subsection 8.1.1, the formal definition of BFS is given. In Subsection 8.1.3, we give an example computation of BFS, using the same example network as for FLOOD, starting from a configuration where all processes are in a "clean" state. BFS converges in $O(n)$ rounds, regardless of the initial configuration. In Subsection 8.1.4, we give an example computation of BFS starting from a configuration which is not "clean." In Subsection 8.1.5, we give the proofs of correctness and time complexity of BFS.

In Section 8.2 we give MIS, the second module of BFS-MIS-CLSTR. This module constructs a maximal independent set, $S$, of the network, as well as the MIS spanning tree $T_{MIS}$, a tree also rooted at $Root$, which has the property that the members of $S$ are precisely the processes which are at even levels in $T_{MIS}$. In Subsection 8.2.1 we give an overview of MIS. In Subsection 8.2.2, we give the formal definition of MIS. In Subsection 8.2.3, we give an example computation of MIS. In Subsection 8.2.4, we give the proofs of correctness and time complexity of MIS.

In Section 8.3, we give the CLSTR, the third and final module of BFS-MIS-CLSTR. In Subsection 8.3.1, we give an overview of CLSTR. In Subsection 8.3.2, we give the formal definition of CLSTR. In Subsection 8.3.3, we give the proof that CLSTR constructs
the optimal $k$-clustering of $\mathcal{T}_{\text{MIS}}$. (Note that this does not imply that it is an optimal $k$-clustering of the network.) In Subsection 8.3.4, we give the proofs of correctness and time complexity of CLSTR.

In Chapter 9, we prove that CLSTR is $\left(\frac{4\pi}{\sqrt{3}} k + O(1)\right)$-competitive. and we generalize that result to the case of bounded independence graphs.

In Chapter 10, we discuss a generalization of $k$-clustering, which we call $(d,r)$-clustering.

Chapter 11 concludes the thesis.
CHAPTER 2

WIRELESS NETWORKS
In this chapter, we will present a brief overview of wireless networks which include wireless sensor networks and ad hoc wireless networks. A wireless network is used to refer to a telecommunications network where interconnections between nodes is implemented without the use of wires. It is an information transmission system that uses electromagnetic waves such as radio waves. Examples of wireless networks are WLAN (wireless local area networks), wireless PAN (personal area networks), UMTS (universal mobile telephone service), GSM (Global System for Mobile communications), and D-AMPS (Digital Advanced Mobile Phone Service).

2.1 Mobile Wireless Networks

In recent years, our society has become more information oriented and the demand of information accessibility has been growing rapidly. The advantage of using a wireless network is its convenience. Via WLAN, users can access the Internet anywhere outside their work place such as remote offices or even coffee shops. With these advantages, mobile wireless networks have been experiencing a tremendous growth in popularity amongst people who want information and connectivity anytime and anywhere. This growth has led to many technological advances in this field, and has resulted in the rapid development of small, inexpensive and powerful mobile computing devices such as Personal Digital Assistants (PDAs), various hand-held devices, and laptop computers. The ease of mobility of
these units makes it both critical and challenging to maintain communication amongst the various types of mobile devices. Recent advances in wireless communication technologies have enabled wireless mobile devices to communicate with each other in various ways. The aim of such wireless communication is to provide reliable communications and computing environment where users are not tethered to their information source. Mobile Wireless Networks can be classified into two branches; infrastructured (cellular) and infrastructureless (ad hoc) wireless networks [32].

**Infrastructured Wireless Network.** An infrastructured wireless network is a wireless network in which access points are distributed along a wired backbone, and mobile devices connect to each other by communicating directly with these access points. These access points do not move and are present just to act as routers and forward packets for other nodes, thus allowing the mobile nodes to save power. Also, the access points are usually connected to the fixed network infrastructure or to the Internet. Mobile nodes that are within the coverage area of an access point are able to send and receive signals to that access point, and can thus communicate directly with that access point. However, as a mobile node moves out of the coverage area of one access point and into that of another, it must cease communication with the old access point and begin communication with the new access point. This process is called a hand-off, and should be completely undetectable to the user [61]. Examples of this kind of wireless networks are Wireless Local Area Network (WLAN), cellular networks, Wireless Local Loop (WLL), Universal Mobile Telecommunication System (UMTS), and Global System for Mobile Communications (GSM).

Infrastructured wireless networks are typically used in locations where access points
can be easily installed and connected to an existing network, such as office buildings and college campuses.

**Infrastructureless/Ad Hoc Wireless Networks.** There may be a need for efficient and dynamic communication of independent mobile users when no fixed wired infrastructure is available. A few examples are emergency/rescue operations, disaster recovery, and military networks. In such situations, organized communication networks can not be relied upon. Thus, establishing reliable networks quickly among a collection of mobile hosts without any centralized administration is required. As such, the development of mobile devices and their networks have been receiving more and more interest.

This thesis is on infrastructureless wireless networks which we will present in the next two sections.

2.2 Wireless Sensor Networks

Wireless sensor networks have been recognized as one of the most important technologies for the future. It allows us to instrument, observe, and respond to phenomena in the surrounding environment. A number of sensors spread across a geographic area composing sensor networks. Recent technological advancement has enabled the production of small, low-cost, low-powered, distributed sensing devices called sensor nodes. Each sensor node has wireless communication capability and some level of computational ability for signal processing and networking of data, but has a limited energy source. Sensor nodes are usually static. However, some nodes can be mobile. Sensor networks have similarities with wireless ad hoc networks such as MANET and mobile cellular networks. However the following characteristics of sensor networks [18] suggest that many recommended protocols
for the above two platforms may not be well suited for sensor networks.

Sensor nodes have limited energy supply, communication (transmission) range, and memory. Signals detected by physical sensors have an inherent uncertainty. Sensor nodes may or may not be supported by satellite location determination system such as GPS. Sensor nodes are usually densely deployed for the purpose of fault tolerances. In most cases, physical maintenance may not be infeasible. The topology of sensor networks may change dynamically, due to change of position, reachability (e.g., jamming, noise, obstacles, etc.), available energy, malfunctioning, etc. Also sensor nodes can fail easily due to the low cost in manufacturing or environmental threats such as destruction by animals or vehicles. Therefore sensor networks should be self-healing, as well as self-organizing (Chapter 3).

Sensors are used as both data generators and routers. Networked sensor nodes can aggregate data to provide a rich, multi-dimensional view of the environment. Source sensors detect the event or gather data. Sources are usually located where the environmental activities of interest are expected to take place. Sink nodes are basically monitoring terminals such as mobile PDAs or laptops. They are connected to other networks such as the Internet and provide remote access to data from the sensor network.

**Architecture.** Sensor nodes are typically composed of on-board sensors, a processor, a small amount of memory, a wireless modem, and a limited energy source. Overall prototypes of currently available sensor nodes are very similar, but their size and shape are come in great varieties. Five requirements for networked sensor systems were given in [37]. They are (a) small physical size and low power consumption, (b) concurrency-intensive operation, (c) limited physical parallelism and controller hierarchy, (d) diversity in design and usage, and (e) robust operation.
Applications. Today, there are many different types of sensors such as seismic, infrared, acoustic, visual, and radar amongst others. Hence there are a wide variety of conditions that can be monitored by sensor nodes that include temperature, humidity, pressure, noise, and vehicular movement. Also, sensor nodes can be used for continuous sensing or event detection. Consequently, application fields of sensor networks are limitless. The followings are a few examples: (a) Military applications: Sensor networks can be used to detect biological and chemical attacks and create warning systems. Also they can be used to monitor an ally's condition and status. (b) Environmental applications: One interesting example of this area was presented in [49]. Sensors were deployed on Great Duck Island in Maine for habitat monitoring. Forest fire detection and flood detection systems are also good examples in this category. (c) Health applications: Doctors can monitor the current condition of patients by using sensors which may detect heart rate or blood pressure. (d) Commercial applications: There are numerous applications in this field. Inventory management, intruder detection, and vehicle tracking use sensor networks to attain a so-called smart environment.

Many requirements for above mentioned application areas may be very unique and not suitable for traditional ad hoc networks. For instance, in military applications, there is heightened chance that nodes will be destroyed by an enemy. Because sensor nodes are cheap and disposable, they can be deployed densely to tolerate a node's fault. Therefore, in the future, wireless sensor networks will be an integral part of our lives.

Sensor Network Services. Several services must be provided by sensor networks in addition to low-level networking. Such services are unique to sensor networks. Some examples are described in [36].
Ubiquitous/Pervasive Computing.

"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it" [70]. These are the words of the late Mark Weiser who was the chief technologist at Xerox PARC and considered the father of ubiquitous computing. He described this new era as that of most computers vanishing into the background and being "nearly invisible" from users, but would always be available, which was called invisible computing, and one of the key concepts of his vision. This invisible tool is one that does not intrude on our consciousness so that we can focus on the task. An example of this concept is eyeglasses. We look at the world, not the eyeglasses. Computers should be the same. They would be available and prevalent throughout the physical environment without users actually having awareness of them. Another key concept was presented in [72], known as calm technology. The goal of "calm" technology is to send information in a calm manner. Technology such as cellphones and TVs are often the antithesis of this concept. However, calm technology allows the user to choose what information is needed and what information is peripheral (or sensory) to reduce information overload, while still allowing the user to move easily from the center of information to periphery and back. This can be performed by giving more detail to the periphery. In [72], an example of this calm technology is shown by comparing a video conference and a phone conference. The video conference can give participants visual knowledge of details such as facial expression or body posture, so that participants are more confident about what information is important, hence a more "calm" environment than that of a phone conference.

Ubiquitous computing is about making our lives more simple through digital environments that are sensitive, adaptive, and responsive to human needs. It is now a framework
for new and exciting research in the field of computer science, which includes mobile devices, sensors, and many smart appliances.

2.3 Mobile Ad Hoc Networks

A Mobile ad hoc network (MANET) consists of a set of mobile hosts operating without the aid of the established infrastructure of centralized administration. In fact, roughly speaking two or more users can become a mobile ad hoc network simply by being close enough to meet the radio constraints, without any external intervention. In this type of network, communication between mobile hosts is peer-to-peer; so, each host has direct communication with another. Hosts also act as relay nodes to forward data packets. This is a very important part of communication technology that supports truly pervasive/ubiquitous computing, because in many contexts, information exchange among mobile units cannot rely on any fixed network infrastructure but on the rapid configuration of wireless connections on the fly [65].

MANETs are gaining momentum because they help realize network services for mobile users in areas with no pre-existing communications infrastructure, or when the use of such infrastructure requires wireless extension. Ad hoc nodes can also be connected to a fixed backbone network through a dedicated gateway device enabling IP networking services in the areas where Internet service is not available due to the lack of a preinstalled infrastructure. All these advantages make ad hoc networking an attractive option in future wireless networks.

Minimal configuration and quick deployment make ad hoc networks suitable for emergency situations like natural or human-induced disasters, military conflicts, emergency medical situations, etc. The earliest MANETs were called packet radio networks, and were
sponsored by DARPA in the early 1970s. It is interesting to note that these early packet radio systems predated the Internet, and indeed were part of the motivation of the original Internet Protocol suite.

**Characteristics.** Mobile ad hoc networks involve all networking layers, ranging from the physical to the application layer. The characteristics of mobile ad-hoc networks architecture differ significantly from other networks as described below:

- Each node is free to move while communicating with other nodes.
- The bandwidth available is of the order of 1 Mbps, an order of magnitude less than that of wired networks.
- All communication in a wireless network is broadcast, which means that broadcast is no more expensive than unicast.
- Mobile nodes have limited battery power.
- Wireless links are much more error prone than wired links.
- The topology of ad hoc network is dynamic in nature due to constant movement of the participating nodes, causing the inter-communication patterns among nodes to change continuously.
- Every computer may not be within the communication range of every other computer. So, multiple hops may be needed. Hence, the nodes must serve as routers for other nodes in the network so that data packets can be forwarded to their destinations.

**Ad Hoc Networking Issues.** In general, mobile ad hoc networks are formed dynamically by an autonomous system of mobile nodes that are connected via wireless links
without using the existing network infrastructure or centralized administration. Routes between nodes in an ad hoc network may include multiple hops. Hence, it is appropriate to call such networks multi-hop wireless ad hoc networks.

The ad hoc networks flexibility and convenience do come at a price. Ad hoc wireless networks inherit the traditional problems of wireless communications and wireless networking (IEEE P802.11/D10, January 14, 1999.) as described below:

- The wireless medium has neither absolute, nor readily observable boundaries outside of which stations are known to be unable to receive network frames.

- The channel is unprotected from outside signals.

- The wireless medium is significantly less reliable than the wired media.

- The channel has time-varying and asymmetric propagation properties.

- Hidden-terminal and exposed-terminal phenomena may occur.

To these problems and complexities, the multi-hop nature, and the lack of fixed infrastructure add a number of characteristics, complexities, and design constraints that are specific to ad hoc networking [16, 15], and are described below:

- **Autonomous and Infrastructureless.** MANET does not depend on any established infrastructure or centralized administration. Each node operates in distributed peer-to-peer mode, acts as an independent router, and generates independent data. Network management has to be distributed across different nodes, which brings added difficulty in fault detection and management.

- **Multi-Hop Routing.** No default router is available. Every node acts as a router and forwards each other’s packets to enable information sharing among mobile hosts.
Routing protocols are self-starting, adapt to the changes in network conditions, and also offer multi-hop paths from a source to a destination across the network. Routing protocols designed for ad hoc networks can be adopted to greatly improve the scalability of routing protocols designed for use in the global Internet, which would be an enormous payoff for ad hoc network research. More detailed information on routing in MANET, is given in [58].

- **Dynamically Changing Network Topologies.** In mobile ad hoc networks, nodes can move arbitrarily. So the network topology, which is typically multi-hop, can change frequently and unpredictably, resulting in route changes, frequent network partitions, and possibly packet losses.

- **Variation in Link and Node Capabilities.** Each node may be equipped with one or more radio interfaces that have varying transmission/receiving capabilities and operate across different frequency bands [40]. This heterogeneity in node radio capabilities can result in possibly asymmetric links. In addition, each mobile node might have a different software/hardware configuration, resulting in variability in processing capabilities. Designing network protocols and algorithms for this heterogeneous network can be complex, requiring dynamic adaptation to the changing conditions (power and channel conditions, traffic load/distribution variations, congestion, etc.).

- **Energy Constrained Operation.** Batteries carried by each mobile node have limited power supply, processing power is limited, which in turn limits services and applications that can be supported by each node. This becomes a bigger issue in mobile ad hoc networks because, as each node is acting as both an end system and a router at the same time, additional energy is required to forward packets from other nodes.
• **Network Scalability.** Currently, popular network management algorithms were mostly designed to work on fixed or relatively small wireless networks. Many mobile ad hoc network applications involve large networks with tens of thousands of nodes, as found for example, in sensor networks and tactical networks [58]. Scalability is critical to the successful deployment of these networks. The steps toward a large network consisting of nodes with limited resources are not straightforward, and present many challenges that are yet to be solved in areas, such as addressing, routing, location management, configuration management, interoperability, security, high capacity wireless technologies, etc.

**Applications.** The set of applications for MANETs is diverse, ranging from large scale, mobile, and highly dynamic networks, to small and static networks that are constrained by power sources. Typical application domains of MANET include commercial sector, military battlefield, civilian environments, emergency operations, and personal area network (PAN). Some of the specific applications are mentioned below [58]:

• **Conferencing.** When mobile computer users gather outside their normal office environment, the business network infrastructure is often missing. The whole point of the meeting might be to make some further progress on a particular collaborative project. As it turns out, the establishment of an ad hoc network for collaborative mobile computer users is needed even when the Internet infrastructure support already exists.

• **Home Networking.** Consider the scenario that will result if wireless computers become popular at home. These computers will probably be taken to and from the office work environment and on business trips. Such computers will not have topologically related
IP addresses. Assigning multiple IP addresses to each wireless node for identification purposes would add an administrative burden, and the alternative of deploying an ad hoc network seems more attractive.

- **Emergency Services.** Network applications will become increasingly important for emergency services, and thus, it will be important to find ways to enable the operations of networks even when infrastructure elements have been disabled as part of the effects of a disaster. Ad hoc networks can help overcome network impairment during disaster emergencies.

- **Personal Area Networks.** The idea of a personal area network (PAN) is to create a very localized network populated by some network nodes that are closely associated with a single person. When people meet in real life, their PANs are likely to become aware of each other. Mobility becomes more important when interactions between several PANs are needed. Since people usually do not stay in a fixed location with respect to each other for a long time, dynamic nature of this inter-PAN communication is obvious. Ad hoc networks can be used to establish communications between nodes on separate PANs.

- **Embedded Computing Applications.** Some researches predict a world of ubiquitous computing [71], in which computers will be around us, constantly performing mundane tasks to make our lives a little easier. These ubiquitous computers will often react to the changing environment in which they are situated and will themselves cause changes to the environment in ways that are, we hope, predictable and planned. These capabilities can be provided with or without the use of ad hoc networks, but ad hoc networking is likely to be more flexible and convenient that the continual al-
location and reallocation of endpoint IP addresses whenever a new wireless
communication link is established.

• *Sensor Dust.* Consider a situation in which some hazardous chemicals were dispersed
in an unknown manner because of an explosion or some other sort of accident. Instead
of sending in emergency personal who might be subjected to lethal gas and forced
to work in unwieldy protective clothing, it would be better to distribute sensors con­taining wireless transceivers [27] [43]. The sensors could then form an ad hoc network
and cooperate to gather the desired information about chemical concentrations and
identification.

• *Automotive/PC Interaction.* Ad hoc networks can be used to provide interactions
between automotive computers and laptops or PDAs that may accompany us as we
travel in our cars.

• *Educational Applications.*
  
  – Setup virtual classrooms or conference rooms.

  – Setup ad hoc communication during conferences, meetings, or lectures.

• *Commercial Environments.*

  – E-Commerce: e.g., Electronic payments from anywhere (i.e., taxi).

  – Business.
    
    * Dynamic access to customer files stored in a central location on the fly.

    * Provide consistent databases for all agents.

    * Mobile office.
- Vehicular Services.

* Transmission of news, road condition, weather, music, etc.

* Local ad hoc network with nearby vehicles for road/accident guidance.

In spite of the various applications served by the ad hoc networks, they still have to overcome the defects such as the limited wireless transmission range, link quality, fading, noise, interference caused due to its broadcast nature, route changes and packet losses induced due to the node mobility, battery constraints, and potentially frequent network partitions. Security and interception problems are of a major concern, especially in military applications. Therefore, designing the protocol for MANET is very crucial, and these issues must be carefully examined before widespread commercial deployment.
CHAPTER 3

SELF-* SYSTEMS

One of the main topics of research in this thesis is self-* systems. In the following, we start with a brief description of distributed systems which is a computational model commonly used in the design of self-stabilizing algorithms. Then we will give an overview of self-* systems in Section 3.2. We will describe many terms currently being used in the broad area of fault-tolerant computing. Also, an overview of the concept of self-stabilization which is currently a very active area of research will be given in Section 3.3.

3.1 Distributed Systems

A number of definitions have been proposed in the literature to capture the meaning of distributed systems. A distributed system is a communication network, multiprocessor computers, and can be a single multitasking computer [23]. Also, the existence of the collection of these nodes must be transparent to the system user. Although the processors in distributed systems are autonomous in nature, they may need to communicate with each other to coordinate their actions and achieve a reasonable level of cooperation [56]. A program composed of executable statements are run by each computer. Each execution of a statement changes the computer's local memory content, hence the computer's state. Consequently, a distributed system is modeled as a set of n state machines that communicate with each other. There are mainly two models for communications between machines; message passing and shared memory. In the message passing model, machines
communicate with each other by sending and receiving messages. While in the shared
memory model, communication is carried out by writing in and reading from the shared
memory.

3.2 Overview

Software systems are used everywhere. Thusly commercially available software systems
must be able to adjust to different inputs and handle different faults so that they can be
used in many different environments. The different concepts or terms encapsulated in self-*
have been introduced to characterize different ways of detecting, adjusting, and recovering
from such changes. Because these terms have not been formally defined, we will informally
describe them with examples from other sources of literature.

A self-* system should be self-configuring, self-organizing, self-contained, self-healing,
and self-managing. According to [64], research in a self-* system is "a direct response
to the shift from needing bigger, faster, stronger computer systems to the need for less
human-intensive management of the systems currently available. System complexity has
reached the point where administration generally costs more than hardware and software
infrastructure." The goals of the self-* systems are reduction of human administration and
maintenance, and an increase of reliability, availability, and performance.

A system is considered to be self-configuring if starting from an arbitrary state and
an arbitrary input, the system will eventually satisfy the specification of an application or
start behaving properly in finite steps. Therefore, a self-configuring system is the system
which can configure and reconfigure itself under varying conditions or faults. A similar
concept of self-organizing was defined in [4]. In this paper, this concept was applied to
study peer-to-peer systems based on the locality principle. Example applications can be
seen in the field of robotics [20]. The problem considered in these papers is for a system of multiple mobile robots to be able to communicate with each other and form a certain geometric pattern. Since each robot can start from any arbitrary position, but eventually converges to a final shape, proposed solutions are considered to be self-configuring.

A self-contained system is a system in which only local neighbors are affected by any faults or topology change. Thus, if a fault occurs, nodes which are located more than several hops away should not be aware of it.

A self-healing system automatically recovers from different perturbations and dynamic changes. In [66], a self-healing network (SHN) for supporting scalable and fault-tolerant runtime environments was presented. It was designed to support message transmission via multiple nodes while protecting against failures. Finally, within a self-maintaining system, all tasks in all phases in the life cycle of the system are automatic so that it can reduce the system administrator’s tasks. As the number of computer devices continue to increase exponentially, planned maintenance of computers are becoming more and more of an impossible task to manage. As well, the cost of employing network administrators to keep these computers up and running has been rising. In [8], the authors defined this concept from the system administrators perspective as a system which maintenance will only be required at fixed intervals and the required tasks will be clearly defined at maintenance time. Autonomic computing is IBM’s solution to the above management problem [41]. On October 15th, 2001, Paul Horn, Senior Vice President of IBM Research suggested a solution: “Build computer systems that regulate themselves much in the same way our autonomic nervous system regulates and protects our bodies.”

Another approach which was introduced in [31, 54] was recovery-oriented computing, with such systems being called self-repairing computers. This concept can be applied to
designing highly dependable Internet services.

3.3 Self-Stabilizing Systems

In 1973, Dijkstra introduced the term \textit{self-stabilization} in the world of computer science [22, 21], which was a concept of fault-tolerance. Unfortunately, only a handful of people had become aware of its importance until Lamport endorsed this as "Dijkstra's most brilliant work" and "a milestone in work on fault-tolerance" in his invited talk at the ACM Symposium on Principles of Distributed Computing in 1983. Today, it is one of the most active area of research in the field of computer science.

A system is considered self-stabilizing if starting from any arbitrary state (possibly a fault state) it is guaranteed to converge to a legitimate state which satisfies its problem specification in a finite number of steps. Once it converges to a legitimate state, it must stay in that legitimate state thereafter unless a fault occurs. With respect to behavior, it can also be defined as a system starting from an arbitrary state, reaching a state in finite time from which it starts behaving correctly according to its specification. This self-stabilization enables systems to recover from a transient fault automatically.

According to [5, 6], the self-stabilization can be defined in terms of two properties; \textit{closure} and \textit{convergence}. Closure means that if a system is in a correct (or legitimate) state, it is guaranteed to stay in a correct state, if no fault occurs. On the other hand, convergence means that starting from any arbitrary state, it is guaranteed that the system will eventually reach a correct state in finite steps. In order for a system to be self-stabilizing, it must satisfy both of these properties.

In the area of network protocols, self-stabilization has been extensively studied. Protocols like routing, high-speed networks, sensor networks, and connection management are
just a part of many applications of self-stabilization. Also, there exist many self-stabilizing distributed solutions for graph theory problems. For example, spanning tree constructions, maximal matching, search structures, and graph coloring. Many self-stabilizing solutions for numerous classical distributed algorithms were proposed as well. Those include mutual exclusion, token circulation, leader election, distributed reset, termination detection, and propagation of information with feedback [23].

In the study of self-stabilization, several aspects of models have been considered, such as the following:

- **Interprocess Communication**: shared registers or message passing.
- **Fairness**: weakly fair, strongly fair, or unfair.
- **Atomicity**: composite or read/write atomicity.
- **Types of Daemon**: central or distributed.

All together proving stabilization programs are quite challenging. Two techniques have been commonly used in research literature: convergence stair [34] and variant function [44] methods. Furthermore, many general methods of designing self-stabilizing programs have been proposed which include diffusing computation [7], silent stabilization [24], local stabilizer [1], local checking and local correction [9, 67], counter flushing [68], self-containment [33], snap-stabilization [17], super-stabilization [25], and transient fault detector [11].

Self-stabilization is a significant concept in the study of MANETs. Due to the dynamic nature of MANET topology (Section 2.3), the protocols for setting up and organizing MANETs are desirable to be self-stabilizing.
CHAPTER 4

CLUSTERING

Among the many challenges for ad hoc network designers and users, scalability is a critical issue. In particular, when a flat-topology network contains a large number of nodes, control overhead, such as routing packets, requires a large percentage of the limited wireless bandwidth.

One promising approach is to build hierarchies among the nodes, such that the network topology can be abstracted. This process is commonly referred to as clustering and the substructures that are collapsed in higher levels are called clusters.

A cluster is a subset of the nodes of the underlying network that satisfies a certain property P. At the network initialization stage, a cluster initialization algorithm is invoked and the network is partitioned into individual clusters each satisfying property P. Due to node mobility, new links may form and old ones may break, leading to changes in the network topology and thus, to possible violations of property P. When property P is violated, a cluster maintenance algorithm must be invoked.

Under a cluster structure, mobile nodes may be assigned a different status or function, such as clusterhead, cluster gateway or cluster member. A clusterhead normally serves as a local coordinator for its cluster, performing intra-cluster transmission arrangement, data forwarding, and so-on. A cluster gateway is a non-clusterhead node with inter-cluster links, so it can access neighboring clusters and forward information between clusters. A cluster
member is usually called an ordinary node, which is a non-clusterhead node without any inter-cluster links.

4.1 Need for Clustering

A cluster structure with an effective topology control mechanism provides at least three benefits [59] as outlined below:

- A cluster structure facilitates the spatial reuse of resources to increase the system capacity [48, 47]. With the non-overlapping multi-cluster structure, two clusters may deploy the same frequency or code set if they are not neighboring clusters [38]. Also, a cluster can better coordinate its transmission events with the help of a special mobile node, such as a clusterhead, residing in it. This can save much resources used for retransmission resulting from reduced transmission collision.

- The second benefit is in routing. In any network, the delay incurred by a packet at each hop is a function of the processing and queuing delays at the transmitting nodes and the transmission and propagation delays over the link. Thus, a high number of hops between two nodes causes extra delays. One way to handle this problem is to increase the transmission range of the nodes, thereby reducing the average number of hop distance between any pair of nodes. Some methods of designing reduced-hop backbone topologies have been given in [76, 62]. The clustering scheme using clusterheads and cluster gateways can be used to construct a low-hop backbone network. The backbone network can be utilized to design inter-cluster routing, and thus, the generation and spreading of routing information can be restricted to this set of nodes [28, 55].
• A cluster structure makes an ad hoc network appear *smaller and more stable* in the view of each mobile terminal [50]. When a mobile node moves out of its cluster, its cluster ID may change; only mobile nodes residing in the corresponding clusters need to update the information. Thus, local changes need not be seen and updated by the entire network, and information stored and processed by each mobile node is greatly reduced.

4.2 Cost of Clustering

A cluster-based MANET has its side effects and drawbacks because constructing and maintaining a cluster structure usually requires additional cost compared with a flat-based MANET. The cost of clustering is a key issue to validate the effectiveness and scalability enhancement of a cluster structure. By analyzing the cost of a clustering scheme in different aspects qualitatively or quantitatively, its usefulness and drawbacks can be clearly specified. The clustering cost terms are described as follows [75]:

• To maintain a cluster structure in a dynamically changing scenario often requires explicit message exchange between mobile node pairs. When the underlying network topology changes quickly and involves many mobile nodes, the clustering-related information exchange increases drastically. Frequent information exchange may consume considerable bandwidth, and drain mobile nodes energy quickly. Thus, the upper layer applications cannot be implemented due to the lack of available resources or support from related mobile nodes.

• Some clustering schemes may cause the cluster structure to be completely rebuilt over the whole network when some local events take place, *e.g.*, the movement or failure of a mobile node, resulting in some clusterhead re-election (re-clustering). This is
called the ripple effect of re-clustering, which may greatly affect the performance of upper layer protocols.

- Most schemes separate the clustering into two phases, cluster formation and cluster maintenance, and assume that the nodes do not move when the cluster formation process is in progress. However, this assumption may not be applicable in an actual scenario, where mobile nodes may move randomly all the time.

- Another metric is the computation round, which indicates the number of rounds in which a cluster formation procedure can be completed. But in these schemes, not all mobile nodes can decide their status at the same time (within one round), and they may require a non-constant number of rounds to finish the initial cluster construction. Thus, the time required for these algorithms cannot be bounded and may vary noticeably for different network topologies.

4.3 Classifying Clustering Schemes

The clustering schemes of MANETs can be classified according to different criteria as clusterhead-based clustering [73, 14, 12, 74, 10, 51, 2, 42, 53, 13] and non-clusterhead-based clustering [48]. The grouping can also be based on the hop distance between node pairs in a cluster — 1-hop clustering [73, 14, 12, 74, 10, 42, 13], 2-hop clustering [52, 69], and multi-hop clustering. We will present multi-hop clustering in more detail in Chapter 5.

The clustering protocols are classified based on their objectives, into the following categories [75, 26]:

- **Dominating-Set-based Clustering.** DS-based clustering [73, 14, 19] tries to find a DS for a MANET so that the number of mobile nodes that participate in route search
or routing table maintenance can be reduced. Taking a MANET as an un-weighted graph G, a vertex(node) subset S of G is a DS if each vertex in G either belongs to S or is adjacent to at least one vertex in S. The vertices of a DS act as clusterheads. A DS is called a connected DS (CDS) if all the dominating nodes are directly connected with each other. Only the nodes in CDS are required to construct and maintain the routing tables when table-driven routing is applied. When on-demand routing is adopted, the route search space is limited to the CDS. However, local network topology updates may require global adjustment of the structure of CDS. Thus, DS-based clustering is more feasible for static networks or networks with low mobility. Our work presented in Chapters 7 and 8 falls into this category.

- **Low Maintenance Clustering.** Low maintenance clustering schemes [12, 74, 48] aim at providing stable cluster architecture for upper layer protocols with little cluster maintenance cost. By limiting re-clustering situations or minimizing explicit control messages for clustering, the cluster structure can be maintained well without excessive consumption of network resources for cluster maintenance. However, for a network with bursty traffic, the cluster structure is difficult to maintain and cannot be promised to be ready for serving upper-layer routing or data forwarding.

- **Mobility-Aware Clustering.** It takes the mobility behavior of mobile node into consideration [10, 51]. This is because the mobile nodes movement is the main cause of changes to the network topology. By grouping mobile nodes with similar speed into the same cluster, the intra-cluster links can become more tightly connected, and the cluster structure can be correspondingly stabilized in the face of moving mobile nodes. However, in practice, a mobile node needs to collect speed information from
neighbors to decide whether it is with the lowest relative speed in its local area. The information collected may be inaccurate because of the continuous movement of mobile nodes.

- **Energy Efficient Clustering.** Energy-efficient clustering \([2, 42]\) manages to use the battery energy of mobile nodes more wisely in a MANET. A MANET should strive to reduce its energy consumption greedily in order to prolong the network lifespan. By eliminating unnecessary energy consumption of mobile nodes or by balancing energy consumption among different mobile nodes, the network lifetime can be remarkably prolonged. The lack of mobile nodes due to energy depletion may cause network partition and communication interruption.

- **Load Balancing Clustering.** Load-balancing clustering schemes \([2, 53]\) attempt to limit the number of mobile nodes in each cluster to a specified range so that clusters are of similar size. A too-large cluster may put heavy load on the clusterheads, causing clusterheads to become the bottleneck of a MANET. A too-small cluster, however, may produce a large number of clusters and thus increase the length of hierarchical routes, resulting in longer end-to-end delay. Load-balancing clustering schemes set upper and lower limits on the number of mobile nodes that a cluster can deal with. If this limit is exceeded, re-clustering procedures are invoked to adjust the number of mobile nodes in that cluster.

- **Combined-metrics-based Clustering.** Combined-metrics based clustering \([13]\) takes multiple metrics into account, such as node degree, cluster size, mobility speed, and battery energy, in cluster configuration, especially in deciding the clusterheads. With the consideration of more parameters, clusterheads can be better chosen without
giving bias to mobile nodes with specific attributes. This scheme can adjust the weighting factors for each metric to adjust to different scenarios. Re-clustering takes place when a node moves out of a cluster and is not covered by any other clusterhead.

- **Power Control Based Clustering.** In the design of wireless networks, it is essential to consider power economy because most portables are powered by batteries with very limited weight and life. In conventional wireless telephony, where the cellular approach is prevalent, mobile units exercise power control to achieve not only longer battery life but also better communication quality. Power control is applied to clustering in order to improve the performance and the power economy, yet retaining stability and ease of operation. With power control, we can also expect better channel utilization. We can provide better service if we control the number of nodes in a cluster by increasing/decreasing pilot transmission power of the clusterhead and thus the physical cluster size. In [46], a power-control-based two-hop clustering algorithm was proposed in which a clusterhead can adjust the cluster size by exercising power control.

- **Access-based Clustering Protocol.** To minimize the clustering overhead resulting from the control signaling overhead in a hierarchical ad hoc network, the access-based clustering protocol uses MAC layer process for cluster formation [39]. In access-based clustering protocol (ABCP), the cluster formation is heavily influenced by the outcome of the multiple access. The ABCP provides a generic, flexible, rapidly deployed, and stable cluster architecture for the upper layer protocols. Since ABCP makes clustering decision directly based on the result of channel access, it requires fewer control overheads and has shorter convergence time than the other clustering protocols.
CHAPTER 5

K-CLUSTERING

Given a graph $G = (V, E)$, a $k$-cluster of $G$ is defined to be a subset $C \subseteq V$, together with a designated clusterhead $\text{Leader}(C) \in C$, such that each member of $C$ is within distance $k$ of $\text{Leader}(C)$. A $k$-clustering of $G$ is a partition of $V$ into disjoint $k$-clusters.

In this thesis, we present two self-stabilizing asynchronous distributed algorithms for the $k$-clustering problem. We assume that each member of $V$ is a process, and that processes $P$ and $Q$ can communicate if and only if $\{P, Q\} \in E$. We use the composite model of computation.

Throughout the report, we let $n$ be the number of processes in the network, and let $\text{diam}$ be its diameter. We also assume that each process has a unique ID, a positive integer, and that there is no designated leader process.

The two algorithms presented in this thesis are much improved versions of that in [3, 29]. So, we give a brief description of those two algorithms in the next two sections.

5.1 Amis et al. Algorithm

In [3], each process in the network initiates $2d$ rounds of flooding. Every process maintains a logged entry of the results of each flooding round. The first $d$ rounds implement a floodmax scheme to propagate the largest process ID's. Using the values that exist at each process after the first $d$ rounds, the second $d$ rounds constitute a floodmin scheme to propagate the smallest process ID's. After the completion of the second $d$ rounds, each
process looks at its logged entries, and sets its status using the following rules:

• *Rule 1*: Each node checks to see if it has received its own node ID in the second $d$ rounds of flooding. If it has, then it declares itself a clusterhead and skips the rest of the heuristic. Otherwise, proceed to Rule 2.

• *Rule 2*: Each process looks for process pairs. Once a process has identified all process pairs, it selects the minimum process pair to be the clusterhead. If a process pair does not exist, then proceed to Rule 3.

• *Rule 3*: Elect the maximum process ID at the end of the first $d$ rounds of flooding as the clusterhead for this process.

After clusterhead selection each process broadcasts its elected clusterhead to all of its neighbors. If there are neighbors with clusterhead selections that are different, then these processes are *gateway* processes. A gateway process then begins a convergecast to the clusterhead process sending its process ID, all neighboring gateway processes and their associated clusterheads.

• *Rule 4*: There are certain scenarios where this heuristic will generate a clusterhead that is on the path between a process and its elected clusterhead. In this case, during the convergecast the first clusterhead to receive the convergecast will adopt the process as one of its children. The clusterhead will immediately send a message to the process identifying itself as the new clusterhead.

### 5.2 Fernandes and Malkhi Algorithm

In the $k$-clustering algorithm in [29], two processes within a cluster can be at most $k$ hops from one another. This algorithm is based on three phases. In the first phase, a BFS
tree is constructed. The BFS tree construction is not included in the paper. Moreover, it is assumed that the root process is known. In other words, the leader election algorithm is also omitted. In the second phase, an MIS tree is computed on this BFS tree. This is done in a top down manner starting from the root following the levels of the BFS tree. Finally, in the last phase, subtrees of diameter at most $k$ are constructed from the MIS tree by going bottom-up the tree and detaching the subtree along the way.

5.3 Related Work

To the best of our knowledge, there exist only three asynchronous distributed solutions to the $k$-clustering problem in mobile ad hoc networks (MANETs), in the comparison based model, i.e., where the only operation allowed on IDs is comparison. Amis et al. [3] gave the first distributed solution this problem. The time and space complexities of their solution are $O(k)$. Spohn and Garcia-Luna-Aceves [63] gave a distributed solution to a more generalized version of the $k$-clustering problem. In this version, a parameter $m$ is given, and each process must be a member of $m$ different $k$-clusters. The usual $k$-clustering problem is then the case $m = 1$. The time and space complexities of the distributed algorithm in [63] are not given.

A set of nodes $D$ in a graph is called $k$-dominating if every node is within $k$ hops of some member of $D$. Any $k$-dominating set can be used to construct a $k$-clustering letting each member of the set be a clusterhead, and letting each process join the nearest clusterhead. Two synchronous distributed algorithm which compute $k$-dominating sets using a non-comparison based model are given in [45, 57].

A synchronous algorithm for $k$-clustering for wireless radio multi-hop networks is presented in [60].
Amis et al. give a non-self stabilizing message passing algorithm for the \(k\)-clustering problem which takes \(O(k)\) steps, and requires \((3k + O(1))\log_2 n\) bits of memory in each process [3]. Their algorithm has bad worst case behavior, and is not fully explained in their paper.

Fernandess and Malkhi [29] give a non-self stabilizing message passing algorithm for the problem that uses \(O(\log n)\) memory per process, takes \(O(n)\) steps, providing a BFS tree for the network is already given. In the special case that the network is a unit disk graph in the plane, their algorithm is \(8k\)-competitive, meaning that the number of clusters constructed by their algorithm is at most \((8k + K)\) times the minimum possible number of clusters in a \(k\)-clustering of the same network, where \(K\) is a constant that depends neither on the network nor on \(k\). (In [29], a \(k\)-cluster is defined to have diameter at most \(k\). They give competitiveness \(2k\), which is equivalent to competitiveness \(8k\) using the definition of \(k\)-cluster given in this paper.) Finding an optimal \(k\)-clustering, i.e., one which the minimum possible number of clusters, is known to be \(\mathcal{NP}\)-hard.

The proof of competitiveness given by Fernandess and Malkhi contains a flaw, although their result is correct; they incorrectly state that at most \(k^2\) disjoint disks of radius 1 can be placed in a \(2k \times 2k\) square in the plane.
CHAPTER 6

PRELIMINARIES

We will assume that we are given a connected undirected network of \( n \) processes, \( \mathcal{X} \), where \( n \geq 2 \), and an integer \( k \geq 1 \). Each process \( P \) has a unique ID, \( P.id \), a non-negative integer.

Let \( \mathcal{N}_P = \) all the neighbors of \( P \), and let \( \mathcal{U}_P = \mathcal{N}_P \cup \{P\} \). For each integer \( d \geq 0 \), let \( \mathcal{H}_d(P) \) be the \( d \)-hop neighborhood of \( P \), the set of all processes whose distance to \( P \) is at most \( d \). Thus \( \mathcal{U}_P = \mathcal{H}_1(P) \).

Let \( m_{opt}(\mathcal{X}, k) \) be the minimum number of clusters of any \( k \)-clustering of \( \mathcal{X} \). We say that a \( k \)-clustering algorithm \( A \) is \( C \)-competitive if \( m_A(\mathcal{X}, k) \leq C \cdot m_{opt}(\mathcal{X}, k) + O(1) \), where \( m_A(\mathcal{X}, k) \) is the number of clusters produced by \( A \), given inputs \( \mathcal{X} \) and \( k \).

6.1 Self-Stabilization

In this thesis, we will consider the shared memory model introduced in [21]. In this model, each process \( P \) maintains registers, such that \( P \) can write only to its own registers, and read its own registers and registers owned by its neighboring processes. However, if \( P \) and \( Q \) are neighbors, \( P \) is permitted to have a variable that can only be read by \( Q \).

The program of a process consists of a set of registers and a finite set of actions of the following form: \(< label > :: < guard > \rightarrow < statement > \). The guard of an action in the program of \( P \) is a Boolean expression involving the variables of \( P \) and its neighbors. The statement of an action of \( P \) updates one or more variables of \( P \). An action can be executed only if its guard evaluates to true. We assume an asynchronous network.
Actions are atomically executed, meaning, the evaluation of a guard and the execution of the corresponding statement of an action, if executed, are done in one atomic step.

The state of a process is defined by the values of its registers. The configuration of a system is the product of the states of all processes. A distributed algorithm \( \mathcal{A} \) is a relation on \( \mathcal{C} \), the set of possible configurations of the system, denoted by \( \rightarrow \). \( \mathcal{A} \) defines an oriented graph \( (\mathcal{C}, \rightarrow) \), called the transition graph of \( \mathcal{A} \). (By a slight abuse of notation, we refer to \( \mathcal{C} \) as the transition graph if \( \rightarrow \) is understood.) A sequence \( e = \gamma_0, \gamma_1, \ldots, \gamma_i, \gamma_{i+1}, \ldots \), where \( \gamma_i \in \mathcal{C} \), is called an execution of \( \mathcal{A} \) if \( \forall i \geq 0, \gamma_i \rightarrow \gamma_{i+1} \in \mathcal{C} \), and if either \( e \) is infinite or the last member of \( e \) is a sink of \( \mathcal{C} \).

An action \( a \) of a process \( P \) is said to be enabled in \( \gamma \in \mathcal{C} \) if the guard of \( a \) is true in \( \gamma \). A process \( P \) is said to be enabled in a configuration \( \gamma \) if some action of \( P \) is enabled in \( \gamma \).

We consider that any enabled process \( P \) is neutralized in the computation step \( \gamma_i \rightarrow \gamma_{i+1} \) if \( P \) is enabled in \( \gamma_i \) and not enabled in \( \gamma_{i+1} \), but does not execute any action between these two configurations. (The neutralization of a process represents the following situation: at least one neighbor of \( P \) changes its state between \( \gamma_i \) and \( \gamma_{i+1} \), and this change effectively makes the guard of all actions of \( P \) false.) We assume that each transition from a system configuration to another is driven by a scheduler, also called a daemon. The daemon is distributed, meaning that, during a computation step, if one or more processes are enabled, the daemon selects at least one (possibly more) of these enabled processes to execute an action, and the daemon is also unfair, meaning that, even if a process \( P \) is continuously enabled, \( P \) might never be selected by the daemon unless \( P \) is the only enabled process.

We use the notion of round, which captures the speed of the slowest process in an execution. Starting from any configuration \( \gamma \), a round is defined to be the minimal prefix of a computation containing, for each process \( P \) which is enabled at \( \gamma \), an execution of at
least one action of \( P \) or a neutralization of \( P \). The \textit{round complexity} of an algorithm is defined to be the maximum number of rounds before a correct output.

\textbf{Definition 6.1 (Self-stabilization)} We associate a protocol \( P \) with a legitimacy predicate, \( L_P \) that must have the following properties:

(i) Starting from a configuration \( \alpha \) satisfying \( L_P \), every reachable configuration in any execution of \( P \) satisfies \( L_P \) (the closure property).

(ii) Starting from an arbitrary configuration, any execution of \( P \) eventually reaches a configuration satisfying \( L_P \) (the convergence property).

\textbf{Prioritized Actions.} Each action is given a priority number. Each action's guard includes the condition given in the third column of the action tables, and also includes the condition that no action which has an earlier priority number is enabled. We say that an action becomes \textit{silent} if it will never again be enabled. We say that a module (or a program) \textit{converges} if all its actions become silent.

We say that a variable is \textit{consistent} if no action which could alter that variable, or which has a lower priority number, is enabled. We say that a variable is \textit{stable} if all actions which could change the value of that variable are silent. Thus, any stable variable is consistent. The converse does not hold, however. It is possible for a process \( P \) and all its neighbors to be initialized in such a way that \( P \) is not initially enabled to execute any action, which implies that all variables of \( P \) are initially consistent; and yet, in a later round, some variables of some neighbors of \( P \) could change in such a way that \( P \) is enabled to execute an action.
CHAPTER 7

THE ALGORITHM FLOOD

In this chapter, we present a silent, self-stabilizing algorithm FLOOD, which computes a \( k \)-clustering of a network. FLOOD uses \( O(k \log n) \) space per process, and self-stabilizes within \( O(k) \) rounds.

**Basic Idea of FLOOD.** The basic idea of FLOOD is that a process \( P \) is chosen to be a clusterhead if and only if, for some process \( Q \), \( P \) has the smallest ID of any process within \( k \) hops of \( Q \). It requires at most \( 2k \) rounds for each process to be informed that it is, or is not a clusterhead.

A clustering of the network is then obtained by every process joining a tree rooted at the nearest clusterhead; the processes of each tree become one cluster. Every process is within \( k \) hops of some clusterhead, and thus our clustering is a \( k \)-clustering.

**Implementation of FLOOD.** Each process \( P \) contains two arrays, \( P.minid[d] \) for \( 1 \leq d < k \), and \( P.maxminid[d] \) for \( 1 \leq d \leq k \). In addition, \( P \) has variables \( P.leader \) and \( P.parent \), both IDs, and \( P.dist \), a non-negative integer. Each of these variables has a stable value, namely that value that each will have when FLOOD stabilizes. In Lemma 7.2, we will prove the following:

- The stable value of \( P.minid[d] \) is the smallest ID of any process within \( d \) hops of \( P \). If \( P.minid[k] = P.id \), then \( P \) is a clusterhead, however, a process could be a clusterhead without being the smallest ID within \( k \) hops of itself. We thus need to
compute another array.

- The stable value of $P_{\text{maxminid}}[d]$ is the largest value of $Q_{\text{minid}}[k]$ for any process within within $d$ hops of $P$. $P$ is a clusterhead if and only if the stable value of $P_{\text{maxminid}}[k]$ is $P.id$.

- The stable value of $P_{\text{leader}}$ is the ID of $P$'s clusterhead, i.e., the clusterhead nearest to $P$.

- The stable value of $P_{\text{dist}}$ is the distance from $P$ to its clusterhead.

- If $P$ is not a clusterhead, the stable value of $P_{\text{parent}}$ is the ID of the neighbor of $P$ on the shortest path from $P$ to its clusterhead, i.e., the parent of $P$ in the BFS spanning tree of its cluster; that is, $P_{\text{leader}} = P_{\text{parent}}_{\text{leader}}$ and $P_{\text{dist}} = P_{\text{parent}}_{\text{dist}} + 1$.

- The stable values of $\text{leader}$ define a spanning forest in the network, where the clusterheads are the roots, and the trees are the $k$-clusters.

- All variables stabilize within $3k + 1$ rounds of arbitrary initialization.

$P$ acts by checking its variables against those of its neighbors. A variable of $P$ is consistent if it satisfies the appropriate rule in the list below.

- $P_{\text{minid}}[1] = \min \{Q.id : Q \in U_P \}$.

- For $d > 1$, $P_{\text{minid}}[d] = \min \{Q_{\text{minid}}[d - 1] : Q \in U_P \}$.

- $P_{\text{maxminid}}[1] = \max \{Q_{\text{minid}}[k] : Q \in U_P \}$.

- For $d > 1$, $P_{\text{maxminid}}[d] = \max \{Q_{\text{maxminid}}[d - 1] : Q \in U_P \}$.

- If $P_{\text{dist}} = 0$, then $P_{\text{parent}} = P.id$. 

39
• If $P.dist > 0$, then $P.parent = \min \{Q.id : (Q \in N_P) \land (Q.dist + 1 = P.dist)\}$.

• $P.leader = P.parent.leader$.

If all variables of $P$ are consistent, then $P$ is not enabled to execute any action. Otherwise, $P$ will identify the inconsistent variable of lowest priority number, and change its value to make it consistent; where $P.minid[d]$ has priority $d$, $P.maxmin[d]$ has priority $d + k$, and $P.dist$, $P.parent$ and $P.leader$ each have priority $2k + 1$. To save time, FLOOD changes those last three variables in a single action. When all variables of all processes are consistent, FLOOD is silent.

Resolving Ties. Ties, which occur when a process $P$ is equidistant to two nearest clusterheads, can be resolved arbitrarily. We choose to use the "lowest ID of neighbor" rule: if $Q_1$ and $Q_2$ are neighbors of $P$, where $Q_1.id < Q_2.id$, and if, stably, $P.dist = Q_1.dist + 1 = Q_2.dist + 1$, then the stable value of $P.parent$ might be $Q_1.id$, but cannot be $Q_2.id$.

7.1 Functions and Actions of FLOOD

We now give a formal definition of FLOOD. Each process $P$ has the following variables. Each variable is of ID type, except $P.dist$, which is a non-negative integer.

$P.id$.

$P.minid[d]$ for $d \leq 1 \leq k$.

$P.maxmin[d]$ for $d \leq 1 \leq k$.

$P.dist$.

$P.parent$.

$P.leader$. 

40
Each process $P$ can evaluate the following functions by reading its variables and those of its neighbors.

- **MinIdF**($P, d$) =
  \[
  \begin{cases} 
  P.id & \text{if } d = 0 \\
  \min \{Q.id : Q \in U_P\} & \text{if } d = 1 \\
  \min \{Q.minid[d - 1] : Q \in U_P\} & \text{if } 2 \leq d \leq k 
  \end{cases}
  \]

- **MaxMinIdF**($P, d$) =
  \[
  \begin{cases} 
  P.maxminid[k] & \text{if } d = 0 \\
  \max \{Q.minid[k] : Q \in U_P\} & \text{if } d = 1 \\
  \max \{Q.maxminid[d - 1] : Q \in U_P\} & \text{if } 2 \leq d \leq k 
  \end{cases}
  \]

- **IsClusterheadF**($P$) = $P.maxminid[k] = P.id$, of Boolean type.

- **DistF**($P$) =
  \[
  \begin{cases} 
  0 & \text{if } IsClusterheadF(P) \\
  \min \{Q.dist + 1 : Q \in N_P\} & \text{otherwise}
  \end{cases}
  \]

- **ParentF**($P$) =
  \[
  \begin{cases} 
  P.id & \text{if } IsClusterheadF(P) \\
  \min \{Q.id : (Q \in N_P) \land (Q.dist + 1 = DistF(P))\} & \text{otherwise}
  \end{cases}
  \]

- **LeaderF**($P$) =
  \[
  \begin{cases} 
  P.id & \text{if } IsClusterheadF(P) \\
  P.parent.leader & \text{otherwise}
  \end{cases}
  \]

The actions of FLOOD are given in Table 7.1.
### Table 1: Actions of FLOOD

<table>
<thead>
<tr>
<th>Action</th>
<th>Priority</th>
<th>Condition</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1(d)</td>
<td>1 ≤ d ≤ k</td>
<td>(P_{\text{minid}}[d] \neq \text{MinId}F(P))</td>
<td>(P_{\text{minid}}[d] \leftarrow \text{MinId}F(P, d))</td>
</tr>
<tr>
<td>A2(d)</td>
<td>1 ≤ d ≤ k</td>
<td>(P_{\text{maxminid}}[d] \neq \text{MaxMinId}F(P, d))</td>
<td>(P_{\text{maxminid}}[d] \leftarrow \text{MaxMinId}F(P, d))</td>
</tr>
</tbody>
</table>
| A3      | 2k + 1   | \(P_{\text{dist}} \neq \text{Dist}F(P)\) \lor (P_{\text{parent}} \neq \text{Parent}F(P)) \lor (P_{\text{leader}} \neq \text{Leader}F(P)) | \(P_{\text{dist}} \leftarrow \text{Dist}F(P)\)
|         |          |                                                 | \(P_{\text{parent}} \leftarrow \text{Parent}F(P)\)
|         |          |                                                 | \(P_{\text{leader}} \leftarrow \text{Leader}F(P)\) |

#### 7.2 An Example Computation

In Figure 7.1, we give a network, which we call the *standard graph* that we shall use for example calculations throughout this paper. In Figure 7.2, which consists of 12 subfigures, we illustrate the steps of a computation of FLOOD, where \(k = 4\).

Figures 7.2(a) through 7.2(d) show the stable values of \(\text{minid}[i], i = 1 \ldots 4\), which are computed by (Action A1). Figures 7.2(e) through 7.2(h) show the stable values of \(\text{maxminid}[i], i = 1 \ldots 4\) which are computed by (Action A2). Figure 7.2(h) also shows the final selection of clusterheads, namely processes 10, 13, and 14. Clusterheads are indicated by larger dots in Figures (h) through (l). Figures 7.2(i) though 7.2(l) demonstrate the growth of clusters around the three clusterheads. Note that the cluster subgraphs are BFS trees rooted at the clusterheads.

We used boxed numbers and dashed polygonal lines to identify the different zones created by the \(\text{minid}[i]\) (in Figures 7.2(a) through 7.2(d)) and \(\text{maxminid}[i]\)
(Figures 7.2(e) through 7.2(l)) values, where a zone is defined to be the set of processes whose value of $\text{minid}[i]$ or $\text{maxmin}[i]$, for a given $i$, is the same.

For example, in Figure 7.2(a), processes 18, 37, 66, 76, and 93 computed 18 as their $\text{minid}[1]$. In Figure 7.2(h), processes 13, 21, 32, 36, 39, 50, 63, 75, and 87 computed 13 as their $\text{maxmin}[4]$ and their final clusterhead. Note that in Figure 7.2(d), process 14, which will be chosen to be a clusterhead because it is the $\text{minid}$ of some processes, is not a member of its own zone.
Figure 7.2: Sequence of configurations illustrating FLOOD.
7.3 Proofs for FLOOD

To aid in our proofs, we define a number of functions. Unlike the functions introduced in Section 7.1, which can be computed by a process \( P \), these functions are defined abstractly.

\[
\begin{align*}
\text{Dist}_A(P, Q) &= \text{the distance, i.e., number of hops, from } P \text{ to } Q. \\
\text{MinId}_A(P, d) &= \min \{ Q.id : Q \in \mathcal{H}_d(P) \}. \\
\text{MaxMinId}_A(P, d) &= \max \{ \text{MinId}_A(Q, k) : Q \in \mathcal{H}_d(P) \}. \\
\text{IsClusterhead}_A(P) &\equiv \exists Q : \text{MinId}_A(Q, k) = P.id. \\
\text{Dist}_A(P) &= \min \{ \text{Dist}_A(P, Q) : \text{IsClusterhead}_A(Q) \}.
\end{align*}
\]

The similarity of the names of the abstract functions given above and the locally computable functions given in Section 7.1 is deliberate. For example, \( \text{MinId}_A(P, d) \) is unchangeable, and is not immediately knowable by \( P \), while \( \text{MinId}_F(P, d) \) is changeable, and is computable by \( P \) at any time. We shall show that, eventually, the computable value of \( \text{MinId}_F(P, d) \), as well as the variable \( P.minid[d] \), will be equal to \( \text{MinId}_A(P, d) \).

**Lemma 7.1** \( \text{IsClusterhead}_A(P) \) if and only if \( \text{MaxMinId}_A(P, k) = P.id \).

**Proof.** One direction is easy: if \( \text{MaxMinId}_A(P, k) = P.id \), then, by definition, \( \text{MinId}_A(Q, k) = P.id \) for some \( Q \in \mathcal{H}_k(P) \), i.e., \( \text{IsClusterhead}_A(P) \). We prove the converse by contradiction. Suppose that \( \text{IsClusterhead}_A(P) \). Pick \( Q \) such that \( \text{MinId}_A(Q, k) = P.id \). By definition, \( P \in \mathcal{H}_k(Q) \), which implies that \( Q \in \mathcal{H}_k(P) \).

Suppose \( \text{MaxMinId}_A(P, k) = \text{MinId}_A(R, k) = S.id \). If \( S.id > P.id \), then that contradicts the definition of \( \text{MinId}_A(R, k) \), since \( P.id \) would be a better choice. If \( S.id < P.id \), that contradicts the definition of \( \text{MaxMinId}_A(P, k) \), since \( P.id \) would be a better choice than \( S.id \). Thus, \( S = P \). \( \square \)
**Lemma 7.2** Let $P$ be a process.

(a) If at least $t$ rounds have elapsed, then $P.\text{minid}[d] = \text{MinIdA}(P,d)$ for all $1 \leq d \leq \max\{t,k\}$.

(b) After $k$ rounds have elapsed, Action $A1$ is silent.

(c) If at least $t$ rounds have elapsed, then $P.\text{maxminid}[d] = \text{MaxMinIdA}(P,d)$ for all $1 < d < \max\{t-k,k\}$.

(d) After $k$ rounds have elapsed, Actions $A1$ and $A2$ are silent.

(e) If at least $t$ rounds have elapsed, then $P.\text{dist} > \min\{t - 2k, \text{DistA}(P)\}$.

(f) If at least $t$ rounds have elapsed and $\text{Dist}(P) \leq t - 2k - 1$, then $P.\text{dist} = \text{DistA}(P)$.

(g) If at least $t$ rounds have elapsed, and if $\text{DistA}(P) < t - 2k - 1$, then $P.\text{leader} = \text{LeaderA}(P)$ and $P.\text{parent} = \text{ParentA}(P)$.

**Proof.** We prove Part (a) by induction on $t$. Action $A1$ is enabled to execute whenever its guard is true, since no action has a lower priority number.

Let $t = 1$. $\text{MinIdF}(P,1) = \text{MinIdA}(P,1)$ permanently, since both have the same definition.

$P$ is enabled to execute Action $A1(1)$ if $P.\text{minid}[1] \neq \text{MinIdF}(P,1)$. Thus, within one round, $P.\text{minid}[1] = \text{MinIdA}(P,1)$

Suppose $t \geq 2$. If $d < t$, we are done, by the inductive hypothesis. Let $d = t$. After $t - 1$ rounds, by the inductive hypothesis, $Q.\text{minid}[t - 1] = \text{MinIdA}(Q,t - 1)$ for all $Q$, and hence $\text{MinIdF}(P,t) = \text{MinIdA}(P,t)$; and by the inductive hypothesis, all actions of priority numbers $1 \ldots d - 1$ are silent, and thus $P$ is enabled to execute Action $A1(d)$ if its guard is true. Within one more round, $P.\text{minid}[t] = \text{MinIdA}(P,t)$.

Part (b) follows from (a), by letting $t = k$.

We prove Part (c) by induction on $t$. 

47
If \( t < k + 1 \), the statement is vacuous. Suppose \( t = k + 1 \). Then \( d = 1 \). By (a), \( Q.min[k] = MinIdA(Q, k) \) for all \( Q \in \mathcal{U}_P \) after \( k \) rounds have elapsed, and thus, by definition, \( MaxMinIdF(P, 1) = MaxMinIdA(P, 1) \). By (b), \( P \) is enabled to execute Action \( A2(1) \) if \( P.maxminid[1] \neq MaxMinIdF(P, 1) \). Thus, within one more round, 
\[ P.maxminid[1] = MaxMinIdA(P, 1) \]
Suppose \( t > k + 1 \). Without loss of generality, \( t \leq 2k \). If \( d < t - k \), we are done, by the inductive hypothesis. Let \( d = t - k \). By the inductive hypothesis, \( Q.maxmin[d - 1] = MinIdA(Q, d - 1) \) for all \( Q \in \mathcal{U}_P \) after \( t - 1 \) rounds have elapsed, and thus \( MaxMinIdF(P, d) = MaxMinIdA(P, d) \). By (b) and by the inductive hypothesis, \( P \) is enabled to execute Action \( A2(d) \) if \( P.maxminid[d] \neq MaxMinIdF(P, d) \). Thus, within one more round, 
\[ P.maxminid[d] = MaxMinIdA(P, d) \]
Part (d) follows from (c), by letting \( t = 2k \), and from Part (b).

We prove Part (e) by induction on \( t \).

If \( t \leq 2k \) or \( DistA(P) = 0 \), we are done, since \( P.dist \) cannot be negative.

Suppose \( t > 2k \) and \( d = Dist(P) > 0 \). If \( d < t - 2k \), we are done by the inductive hypothesis. Let \( d = t - 2k \). After \( t - 1 \) rounds have elapsed, \( Q.dist \geq d - 1 \) for all \( Q \in \mathcal{N}_P \), and hence \( DistF(P) \geq d \), by the inductive hypothesis. By (d), \( P \) is enabled to execute Action \( A3 \) if \( P.dist \neq DistF(P) \). Thus, after one more round, \( P.dist \geq d \).

We prove Part (f) by induction on \( t \). Let \( d = DistA(P) \). If \( t = 2k + 1 \) and \( d = 0 \), then, by (d), \( P \) is enabled to execute Action \( A3 \) if \( P.dist \neq DistF(P) \). Thus, after one more round, \( P.dist = 0 \).

Suppose \( t > 2k + 1 \). If \( d < t - 2k - 1 \), we are done by the inductive hypothesis. Let \( d = t - 2k - 1 \). Pick \( Q \in \mathcal{N}_P \) such that \( DistA(Q) = d - 1 \). By the inductive hypothesis,
after \( t - 1 \) rounds, \( Q.\text{dist} = d - 1 \), hence \( \text{DistF}(P) \leq d \); and \( P \) is enabled to execute Action A3 if \( P.\text{dist} \neq \text{DistF}(P) \). Thus, after one more round, \( P.\text{dist} \leq d \). By Part (e), \( P.\text{dist} = d \).

We prove Part (g) by induction on \( t \). Let \( d = \text{DistA}(P) \). If \( t = 2k + 1 \) and \( d = 0 \), then, by (c), after \( t - 1 \) rounds have elapsed, \( \text{LeaderF}(P) = \text{LeaderA}(P) = P.\text{id} \) and \( \text{ParentF}(P) = \text{ParentA}(P) = P.\text{id} \), and by (d), \( P \) is enabled to execute Action A3 if its guard is true. Thus, after one more round, \( P.\text{leader} = P.\text{parent} = P.\text{id} \).

Let \( t > 2k + 1 \). If \( d < t - 2k - 1 \), we are done by the inductive hypothesis. Let \( d = t - 2k - 1 \).

Pick \( Q \in \mathcal{N}_P \) such that \( \text{ParentA}(P) = Q.\text{id} \). By (f), \( Q.\text{dist} = d - 1 \) and, by the inductive hypothesis, \( Q.\text{leader} = \text{LeaderA}(Q) = \text{LeaderA}(P) \) after \( t - 1 \) rounds have elapsed. We need to show that \( \text{ParentF}(P) = Q.\text{id} \) and \( \text{LeaderF}(P) = Q.\text{leader} \) after \( t - 1 \) rounds have elapsed.

If \( R \in \mathcal{N}_P, R \neq Q \), then, by definition of \( \text{ParentA}(P) \), \( \text{DistA}(Q) \geq d - 1 \), and \( R.\text{id} > Q.\text{id} \) if \( \text{DistA}(R) = d - 1 \). By (e), if \( t - 1 \) rounds have elapsed, \( \text{DistF}(R) \geq d - 1 \) and \( \text{DistF}(R) > d - 1 \) if \( \text{DistA}(R) > d - 1 \), hence \( \text{ParentF}(P) \neq R \). The statement of Action 3 consists of three parts, executed in sequence. If that action is not enabled, we are done. Otherwise, after the first part has executed, \( P.\text{dist} = d \), by (f), and \( \text{ParentA}(P) = Q \).

After the second part has executed, \( P.\text{parent} = Q.\text{id} \) and \( \text{LeaderF}(P) = Q.\text{leader} \). After the third part has executed, \( P.\text{leader} = Q.\text{leader} \), and we are done. \( \square \)

**Theorem 7.1** FLOOD stabilizes within \( 3k + 1 \) rounds of initialization, and partitions the processes into \( k \)-clusters. The processes of each cluster form a BFS tree, of height at most \( k \), rooted at the clusterhead.

**Proof.** For each process \( P \), the process whose ID is \( \text{MinIdA}(P, k) \) is a clusterhead, and thus \( P \) is within \( k \) hops of some clusterhead. Thus, by Lemma 7.2, FLOOD is silent
after $3k + 1$ rounds. By the definitions of $\text{Dist} A$, $\text{Parent} A$, and $\text{Leader} A$, and by Lemma 7.2, $\{ P : P.\text{leader} = R.\text{id} \}$ is a $k$-cluster for any clusterhead $R$, and contains an internal BFS tree rooted at $R$ defined by the parent pointers. □

7.4 A Worst Case Example

In this section, we show that, in the worst case, FLOOD picks most processes to be clusterheads, even in the special case of a planar disk graph.

![Figure 7.3: The Line Graph $L_{23,3}$](image)

The Line Graph $L_{n,m}$. For any integers $1 < m < n$, define $L_{n,m}$ to be the network consisting of processes $P_1, \ldots P_n$, where $P_i.\text{id} = i$, and where $P_i$ is adjacent to $P_j$ if and only if $|i - j| \leq m$.

Figure 7.3 shows the graph $L_{23,3}$. Note that $L_{n,m}$ can be realized as a unit disk graph in the line (and hence the plane), by placing each $P_i$ at the point whose coordinate is $\frac{2i}{m}$.

**Lemma 7.3** Consider the $k$-clustering chosen by FLOOD on $L_{n,m}$, for any given $n$, $m$, and $k$. Then $P_i$ is a clusterhead if and only if $i \leq n - mk$.

**Proof.**

$$\text{MinId} A(P_j) = \begin{cases} j - mk & \text{if } j > mk \\ 1 & \text{otherwise} \end{cases}$$
Thus, if $mk \leq \frac{1}{2}n$, most of the processes of $\mathcal{L}_{n,m}$ will be chosen to be clusterheads by FLOOD. □

7.5 A Lower Bound for Comparison Based Clustering Algorithms

We now show that the worst case behavior illustrated in 7.4 is unavoidable for any fast algorithm that uses only comparison to distinguish IDs.

We define an algorithm for the $k$-clustering problem to be comparison based if the only operator permitted on IDs is comparison. For example, the algorithms FLOOD and BFS-MIS-CLSTR given in this paper are comparison based. In contrast, an algorithm that can do arithmetic on an ID, such as computing $P.id \mod 2$ or extracting a single bit from an $P.id$, is not comparison based.

**Theorem 7.2** There is no comparison based deterministic distributed algorithm for the $k$-clustering problem that takes $o(\text{diam})$ time, where diam is the diameter of the network, and selects fewer than half of all processes to be clusterheads. Furthermore, there is no function of $k$ which is an upper bound on the competitiveness of such an algorithm.

**Proof.** Let $k$ be given. Suppose that $A$ is a comparison based deterministic distributed algorithm for the $k$-clustering problem that takes at most $r \geq k$ rounds for any network. Pick $n = 4r + 2$, and let the network be $\mathcal{L}_{n,1}$.

We start $A$ in a configuration where all processes have the same values of their variables. Consider an adversary which selects all enabled processes at every step. All behavior of a $P_i$ during the first $t$ rounds is determined by the initial states of the processes in the “window” around $P_i$ of radius $t$, i.e., $\{P_j : |i - j| \leq t\}$. Because comparison is the only operator permitted for IDs, the windows of radius $r$ for all $P_i$ such that $r + 1 \leq i \leq n-r-1$ are indistinguishable to $A$. Thus, either all those processes will be chosen to be clusterheads,
or none will. Choosing none is impossible, since the middle processes would not be in any cluster. Thus, all the processes in that range, more than half the processes altogether, will be clusterheads.

To prove the second part, assume that $A$ has competitiveness $C_k$. Let $n = \frac{4rC_k}{k} + 2$, and let $m = \frac{C_k}{k}$. Let the network be $C_{n,C_k/k}$. Using essentially the same argument as above, we can show that $A$ must choose more than half the processes to be clusterheads; but the optimal $k$-clustering consists of only $\frac{n}{2C_k}$ clusters. □
CHAPTER 8

THE ALGORITHM BFS-MIS-CLSTR

In this chapter, we give a silent self-stabilizing asynchronous distributed algorithm for the $k$-clustering problem. Our algorithm, BFS-MIS-CLSTR, consists of three modules, BFS, MIS, and CLSTR, whose actions are given in Tables 8.1, 8.2, and 8.3, respectively.

The first module of BFS-MIS-CLSTR is BFS, which constructs a BFS spanning tree, $T_{BFS}$, rooted at $Root$, the process which has the smallest ID. Within $O(n)$ rounds of arbitrary initialization, BFS is silent, meaning that all the variables associated with BFS have achieved their silent values and will never change again, and no action of BFS can ever again be enabled.

The second module is MIS, which constructs a maximal independent set $S$ of processes, as well as what we call the MIS spanning tree, $T_{MIS}$. The MIS spanning tree is also rooted at $Root$, and a process $P$ is a member of the $S$ if and only if $P$ is at an even level in $T_{MIS}$. The height of $T_{MIS}$ is at most twice the height of $T_{BFS}$, hence at most twice the diameter of the network. Once BFS is silent, MIS becomes silent within $O(n)$ rounds.

The third module is CLSTR, which constructs a $k$-clustering of the network. Each cluster constructed by CLSTR is a connected subgraph of the MIS spanning tree, and hence a tree in its own right. The root of that tree is chosen to be the clusterhead. Each cluster but one (the one that contains $Root$) will contain at least $\left\lceil \frac{k}{2} \right\rceil$ members of the maximal independent set, $S$. It follows that there can be at most $\left\lceil \frac{2n-k}{k} \right\rceil + 1$ clusters altogether. In
the special case that the network is a unit disk graph in the plane, the number of clusters constructed is at most $7.2552k + O(1)$ times the minimum number possible. Once MIS is silent, CLSTR becomes silent within $O(diam)$ rounds, where $diam$ is the diameter of the network. Thus, the entire algorithm BFS-MIS-CLSTR becomes silent within $O(n)$ rounds of arbitrary initialization.

The actions of BFS have priority numbers 1, 2, and 3; the action of MIS has priority numbers 4; and the actions of CLSTR have priority numbers 5 or more. Thus, we can prove convergence of BFS without considering the actions of MIS and CLSTR; we can prove convergence of MIS assuming that all actions of BFS are silent, and without considering the actions of BFS and CLSTR; and we can prove convergence of CLSTR assuming that all actions of BFS and MIS are silent.

8.1 The Module BFS

The first module of BFS-MIS-CLSTR is BFS, which computes a BFS tree, $\mathcal{T}_{bfs}$. The three actions of BFS are given in Table 8.1. Initially, the network is in an arbitrary state. After $O(n)$ rounds, BFS has converged, meaning that its three actions are silent. At this point, the processes are organized into a BFS spanning tree $\mathcal{T}_{bfs}$, rooted at $Root$, the process with the smallest ID.

Each process $P$ has a variable $P.leader\_bfs$, which is ID of $P$'s current leader. When BFS terminates (or becomes silent), $P.leader\_bfs = Root.id$ for all $P$. In addition, $P$ has a variable $P.level\_bfs$, the level of $P$, which is $P$'s current estimate of its distance to its current leader. Upon termination of BFS, $P.level\_bfs$ is the distance from $P$ to $Root$.

The BFS.key of a Process. We define the BFS.key of a process $P$ to be the ordered pair $P.bfs\_key = (P.leader\_bfs, P.level\_bfs)$. Keys are ordered lexically. Let $P.self = (P.id, 0)$. 54
We define the successor of a BFS.key \((i, \ell)\) to be the BFS.key \((i, \ell + 1)\). We define \(\text{MinKeyNeighbor}(P)\) to be the minimum value of \(Q.\text{BFS.key}\) among all \(Q \in \mathcal{N}_P\). Ideally, each process \(P\) should satisfy the following conditions:

C1. \(P.\text{BFS.key} \leq P.\text{self}\).

C2. If \(P.\text{BFS.key} > \text{MinKeyNeighbor}(P)\),

\[
\text{then } P.\text{BFS.key} = \text{successor}(\text{MinKeyNeighbor}(P)),
\]

else \(P.\text{BFS.key} = P.\text{self}\).

If the above two conditions are satisfied, then BFS is done. It then seems to be a simple matter to define a distributed algorithm which converges to those conditions, as follows:

A1. If \((P.\text{BFS.key} > P.\text{self}) \lor (P.\text{BFS.key} \leq \text{MinKeyNeighbor}(P))\),

\[
\text{then } P.\text{BFS.key} \leftarrow P.\text{self}.
\]

A2. If \(\text{successor}(\text{MinKeyNeighbor}(P)) < P.\text{BFS.key} \leq P.\text{self}\),

\[
\text{then } P.\text{BFS.key} \leftarrow \text{successor}(\text{MinKeyNeighbor}(P)).
\]

If \(P.\text{leader.BFS}\) is always the ID of some process in the network, then the simple algorithm described by Actions A1 and A2 above converges within \(\text{diam} + 1\) rounds. \(\text{Root.self}\) is the smallest possible BFS key. After one round, \(\text{Root.BFS.key} = \text{Root.self}\), and after \(t + 1\) rounds, all processes within distance \(t\) of \(\text{Root}\) have stabilized.

However, because of arbitrary initialization, \(P.\text{leader.BFS}\) could be initialized to a value of ID type which is not the ID of any process; in this case we say that \(P\) has a fictitious leader. If some process has a fictitious leader which is less than the ID of any process in the network, the simple algorithm might never converge, because the network might never get rid of the fictitious ID.

Consider a 2-process network with processes, \(P_2\) and \(P_3\), where \(P_1.\text{id} = i\), and where
initially $P_2.BFS\_key = (1,0)$ and $P_3.BFS\_key = P_3.self = (3,0)$. If each process executes one action during each round, after one round, $P_2.BFS\_key = (2,0)$ and $P_3.BFS\_key = (1,1)$. After another round, $P_2.BFS\_key = (1,2)$ and $P_3.BFS\_key = (3,0)$. After a total of $2t$ rounds, $P_2.BFS\_key = (1,2t)$, and $P_3.BFS\_key = (3,0)$.

We can solve that problem by putting an upper bound of $D$ on the value of $P.level.BFS$, where we know in advance that $D > diam$. We replace A1 by A1', given below:

\[ A1'. \text{ If } (P.BFS\_key > P.self) \lor (P.BFS\_key < \text{MinKeyNeighbor}(P)) \lor (P.level.BFS \geq D), \]
\[ \text{then } P.BFS\_key \leftarrow P.self. \]

By induction, it can be shown that if $t$ rounds have elapsed since initialization, and if a process $P$ has a fictitious leader, then $P.level.BFS \geq t$. Thus, after $D + 1$ rounds have elapsed, there will be no fictitious leader in the network. After at most $diam$ additional rounds, the algorithm converges. This method is similar to the algorithm in [7].

We now introduce a method to deal with the problem of fictitious IDs that does not depend on knowledge of an upper bound on the diameter.

**Zero Processes and Frozen Processes.**

In the module BFS defined in this subsection, we solve the fictitious leader problem by introducing one more ID, which we call 0 in this paper, which is less than any other ID, and is known by all processes to be fictitious. If $P.leader.BFS = 0$, we say that $P$ is a zero process. If, in addition, $P.level.BFS = 0$, we say that $P$ is a zero root. Since 0 is less than any other ID, the set of zero processes will expand, as neighbors link to zero processes.

Eventually, this expansion will be halted, and the zero processes will reset, changing to self roots. In order to prevent those new self roots from being immediately recaptured by the zero processes, we make the rule that any self root that is adjacent to a zero process is
a frozen process, i.e., cannot execute any action.

As more zero processes reset, frozen processes again become able to execute. After all zero processes have reset, the BFS spanning tree is completed within $O(\text{diam})$ rounds.

**Parent Pointers.** In $T_{\text{BFS}}$, parents are implicitly defined. If $P \neq \text{Root}$, then the parent of $P$ is the neighbor $Q \in \mathcal{N}_P$ of smallest ID such that $P.\text{BFS.key} = \text{successor}(Q.\text{BFS.key})$.

In the figures illustrating our example computations, we show the implicit parent pointers, in order to improve the exposition. If, for an application, it is necessary to have explicit parent pointers, they can be easily computed by each $P$ in one additional round.

8.1.1 Variables, Functions, and Actions of BFS

Each process $P$ has the following variables.

- $P.id$, the ID of $P$, which is unchangeable and cannot be in error. We are guaranteed that no two different processes have the same ID.

- $P.\text{leader.BFS}$, of ID type. $P.\text{leader.BFS}$ might not initially be the ID of any process actually in the network; if not, it is called a fictitious ID. One value of that type, which must be the least value of that type, will be reserved to be fictitious. We use 0 for this value. Thus, $P.\text{leader.BFS} \geq 0$, but $P.id > 0$.

- $P.\text{level.BFS}$, of non-negative integer type. That is, even with arbitrary initialization, $P.\text{level.BFS}$ may not be negative.

- $P.\text{BFS.key} = (P.\text{leader.BFS}, P.\text{level.BFS})$. Actually a composite of two variables, it is useful to write it as a single variable.

We define the following predicates that can be computed by a process $P$.

\[ \text{Self Root}(P) \equiv P.\text{BFS.key} = P.self \]
ZeroRoot(P) ≡ P.BFS.key = (0,0)

Valid(P) ≡ Self Root(P) ∨ ZeroRoot(P) ∨ (P.BFS.key > MinKeyNeighbor(P))

IsLinked(P) ≡ P.BFS.key = successor(MinKeyNeighbor(P))

Frozen(P) ≡ Self Root(P) ∧ (∃Q ∈ N : Q.leader.BFS = 0)

ZeroLeaf(P) ≡ (P.leader.BFS = 0) ∧

((∀Q ∈ N : (Q.BFS.key ≤ P.BFS.key) ∨ Self Root(Q)))

| A1 | Correct | −Valid(P) | → if P.leader.BFS < P.id then
|    | Errors  |          | P.BFS.key ← (0,0)
|    |         |          | else
|    |         |          | P.BFS.key ← P.self

| A2 | Link | −IsLinked(P) ∧ −Frozen(P) | → P.BFS.key ←
priority 2 |                  | successor(MinKeyNeighbor(P))

| A3 | Reset | ZeroLeaf(P) | → P.BFS.key ← P.self
priority 3 |                |

Table 8.1: Actions of BFS

Figure 8.1: (a) An example graph.
(b) The corresponding BFS tree constructed by BFS.
8.1.2 Phases of BFS

BFS consists of four phases. These phases are not disjoint in a global sense, since different processes could be executing different phases at the same time.

The first phase is the error correction phase. This phase consists of all the executions of Action A1, and lasts at most one round. The corrected processes become self roots or zero roots.

The second phase is the zero tree growth phase. It consists of all the executions of Action A2, either by a zero process or by a process which becomes a zero process. We will show that the zero tree growth phase must end within \( n \) rounds of initialization.

Phase 3 is the reset phase, which consists of all the executions of A3. We will show that the reset phase is completed within \( 2n \) rounds of initialization.

It is possible for a process to execute Action A3 before the Phase 2 is completed on the whole network. In that case, it is possible for that process to once again become a zero process, in which case it must execute A3 later.

Phase 4, the floodmin phase, consists of all executions of Action A2 that are not part of the Phase 2.

Processes which execute Phase 4 actions before Phase 2 is completed on the whole network could possibly return to Phase 2, and then Phase 3.

The number of rounds between initialization and the end of the Phase 4 is at most \( 2n + \text{diam} - 1 \).

If a process starts in a clean configuration where all processes are self roots, the first three phases will not be executed; the processes will only execute the fourth phase.

Figure 8.1 shows an example network and the BFS tree of the graph using the algorithm BFS. Process 10 having the smallest ID becomes the root of the BFS tree.
8.1.3 Example Computation: Starting From A Good Configuration

In Figure 8.2, which consists of 9 subfigures, we show a computation of BFS, starting from a clean configuration, i.e., where every process is a self root. We will assume that, in each round, every enabled process executes exactly one action. Note that our protocol works under an unfair distributed daemon. We made the assumption of synchronous execution only to make the presentation simpler.

A self root is indicated by a large dot. Every other process is indicated by a small dot. The ID of each process is a number near the dot. As stated earlier, our protocol does not maintain the parent pointers. Let

\[
Preds(P) = \{Q \in \mathcal{N}_P : P.BFS\_key = successor(Q.BFS\_key)\}
\]

\[
ImplicitParent(P) = \begin{cases} 
\min\{Q.id : Q \in Preds(P)\} & \text{if } Preds(P) \neq \emptyset \\
\perp & \text{if } Preds(P) = \emptyset
\end{cases}
\]

In order to make the figures more readable, we indicate implicit parents as solid arrows. If a process \(P\) does not have an implicit parent, then \(P.BFS\_key\) is written as an ordered pair next to the dot representing \(P\). Except for Figure 8.2(a), dashed polygonal lines are used to separate zones. All the processes in a zone have the same leader-BFS. The leader-BFS values of the zones, are shown in square boxes, except for those zones consisting of a single selfroot.

As all processes are self roots in Figure 8.2, the first three phases will not execute in this example; only Phase 4, the floodmin phase, will execute. Figure 8.2(a) starts with 59 self roots. During the computation, the trees rooted at smaller ID's grow by absorbing the neighboring trees rooted at larger ID's. Eventually, the only tree rooted at the smallest ID (10 in this example) will survive, and will include all processes of the network, as shown
in Figure 8.2(i).

Starting from the configuration shown in (a), in one round, only 10 out of 59 processes remained as self roots, as shown in (b). Some self roots (10, 13, 14, 17, 18, and 19) were able to capture all of their neighbors inside their trees because they were the MinKeyNeighbor's of their neighbors. However, some other self roots (13, 25, 48, and 64) were unable to absorb any of their neighbors into their trees. For example, all neighbors of 64 found neighbors with smaller BFS_key than that of process 64.

Let us consider process 65, to explain how a process changes its virtual parent before choosing the final one. In (a), all neighbors of 65 are self roots, so have level_BFS equal to 0. Process 65 executes Action A2 and and sets its BFS_key to (26,1) as shown in (b). Process 65 would choose process 26 to be its virtual parent, but in the same step, 26 chose 25 as its leader, and thus cannot be the virtual parent of 65. Also during that first step, process 56, one of the neighbors of 65, chose 14 as its virtual parent. In the second round, from configuration (b) to configuration (c), 65 will choose its virtual parent to be 56. The reason is as follows: 56.BFS_key = (14,1) and 65.BFS_key = (26,1). So, 56 has a smaller BFS_key. After that step, processes 65 and 56 belong to the zone, with leader_BFS 14. , as shown in (d).

In the next (third) round 65 will change its parent pointer again and will point to 26 because now, 65.BFS_key = (14,2) and 26.BFS_key = (13,2). Process 65 is now part of the tree of 13, as shown in (d).

In the meantime, the tree of 10 is expanding. In (d), process 98, which is a neighbor of 26, became part of the tree of 10. So, in the next round, configuration (d) to (e), 65 chooses 98 as its virtual parent, switching from the zone of 13 to that of 10.

61
8.1.4 Another Example Computation: Starting from an Erroneous Configuration

In Figure 8.3, which consists of 20 subfigures, we show a computation of BFS, starting from an almost clean configuration. Of the 59 processes in the network, all but seven start as *self roots*. We will assume that, in each round, every enabled process executes exactly one action.

A *self root* is represented by a large dot, which is white if the process is a *frozen process*.
and black otherwise. A zero root is represented by a large hatched dot. All other processes are represented by small black dots. The ID of each process is a number near the dot. There is an arrow from \( P \) to \( Q \) if and only if \( Q \) is the virtual parent of \( P \). If a process \( P \) has no virtual parent and is not a self root, then \( P.BFS.key \) is written as an ordered pair next to the dot representing \( P \). All the processes in each zone have the same leader.BFS. Except for those zones consisting of just a selfroot, each zone is separated from other zones by dashed polygonal lines and labeled with a box containing the leader.BFS of that zone. If a process is not shown to be a member of a zone, then it is the sole member of a singleton zone, of which it is the leader; to reduce clutter in the figures, the boxes and dashed lines are not shown for those zones.

In Figure 8.3(a), we show an initial configuration which is not "clean," viz., there are seven processes which do not begin as self roots, chosen to illustrate how various situations are handled by BFS. Process 31 initially has a leader whose ID is larger than its own. It will immediately change itself to a self root by executing Action A1. Processes 75 and 91 have leaders whose IDs are smaller than their own, but have no virtual parents. They will immediately change to zero roots by executing Action A1. Processes 44 and 56 are initially valid, but their BFS.key values are incorrect; they will thus execute A2 in the first round,

Phase 2 actions begin at the second round, from Figure (b) to (c), in which processes 72, 14, 92, 37, 47, and 34 in the lower left portion of the diagram; and processes 63, 21, 32, and 87, in the upper right portion of the diagram, execute Action A2, and become zero processes. Phase 2 executes for 5 consecutive rounds, ending at the configuration shown in Figure (g). After that, the set of zero processes will shrink, as its members execute Action A3.

During Phase 2, a self root can become a frozen process because a neighbor joins the
set of zero processes. Examples of this include Process 13 at the second round, as shown in Figure (c), and Process 10 at the third round, as shown in Figure (d).

Phase 3 actions begin at the third round, from Figure (c) to (d), when Processes 34 and 47 execute Action A3. Phase 3 executes for ten consecutive rounds. Figure (m) shows the configuration after the last execution of Phase 3. Note that there are no zero processes and no frozen processes in Figure (m).

In this example, Phase 4 executes for five consecutive rounds, starting with the first round. Figure (f) shows the resulting configuration after those rounds. All the results of these executions, however, are "wiped out" by subsequent Phase 2 executions. After Phase 2 has been completed, Phase 4 begins again, starting with the configuration shown in Figure (i), and executing for the next eleven consecutive rounds, ending with the configuration shown in Figure (t).

We now look at some specific details of this computation. Processes 37 and 66 initially have the fictitious leader 5, but do not realize that this ID is fictitious, since they have virtual parents.

The zone of processes whose leader_BFS is 5 expands on one side, capturing processes with larger leader_BFS values, and contracts on the other side, as the tree of zero processes rooted at process 91 expands. After six rounds, that zone disappears.

The wall of frozen processes that protects processes which are not zero processes from being captured by the trees of zero processes during the rounds that Phases 3 and 4 are both executing can clearly be seen in Figures (i), (j), (k), and (l). For example, in Figure (k), a wall consisting of Nodes 64, 94, 56, 70, 23, 66, and 18, and a similar wall in the upper right portion of the figure, protect processes shown in the middle portion of the figure from being recaptured by the trees of zero processes. In that figure, the set of protected
processes is shown enclosed in a heavy dashed polygon.

After Phase 3 ends, as shown in Figures (m) and beyond, there is no further need for the walls of frozen processes, since there are no more zero processes.

During Phase 4, P.leader.BFS cannot be fictitious. Thus, the smallest value of leader.BFS in the network is the smallest ID of any process, which is 10 in our example. During that phase, each process attaches itself to the neighbor of lowest leader.BFS. Trees rooted at processes with low IDs grow at the expense of trees rooted at larger IDs, and in the end, the tree rooted at 10 captures all processes, as shown in Figure (t).

Phase 3 begins at Figure (g) ends at Figure (m). Phase 2 has been completed; the set of zero processes cannot grow any further. Phase 3 continues through Figure (m). Phase 4 runs concurrently with Phase 3; in Figures (j) through (l), the “wall” of frozen processes, which protects the processes executing Phase 4 from the retreating zero zones, can clearly be seen. Phase 4 ends when all processes have chosen their leader.BFS to be 10, the ID of Root, as shown in Figure (t).
Figure 8.3: Sequence of Configurations
Illustrating BFS. Start Configuration has Errors.
8.1.5 Proofs for BFS

Lemma 8.1 If at least one round has elapsed, all processes are valid.

Proof. No action can cause any valid process to become invalid. Any invalid process is enabled to execute Action A1, which causes it to become valid, and must, by the definition of round, execute that action during the first round. □

We define a process $P$ to be zero active if

1. $P$ is a zero process,

2. For some $Q \in N_P$, $Q$ is not a frozen process, and either $Q.leader > 0$ or $Q.level.BFS > P.level.BFS + 1$.

Lemma 8.2 If at least one round has elapsed, and if $P$ is zero active, then $P$ must have executed during the last complete round.

Proof. Let $t'$ be the current time, and $t$ the time at the beginning of the last complete round. Suppose $P$ did not execute during the last complete round. Then there exists $Q \in N_P$ such that $Q.level.BFS > P.level.BFS + 1$, and $Q$ is not a frozen process.

$Q$ cannot have been a frozen process at time $t$, since it would then still be a frozen process at time $t'$. If $Q$ executed during the last complete round, it must have executed A2, which implies that $Q.level.BFS \leq P.level.BFS + 1$, contradiction. If $Q$ did not execute during the last complete round, then $P$ must have been active during the entire last round, which implies that $P$ must have executed during that round, contradiction. □

Lemma 8.3 If at least $t+1$ rounds have elapsed, and if $P$ is zero active, then $P.level.BFS \geq t$.

Proof. By induction on $t$. If $t = 0$, we are done, since $P.level.BFS \geq 0$.

Suppose $t > 0$. By Lemma 8.2, $P$ must have executed during the last round. The last
execution of $P$ must be A2, causing $P$ to link to a neighbor $Q$. This implies that, at some
time after at least $t$ rounds had executed, $Q$ was zero active. By the inductive hypothesis,
$Q_.level.BFS \geq t - 1$ at the point of that execution, which implies that $P_.level.BFS \geq t$. □

**Lemma 8.4** If at least $t+1$ rounds have elapsed, and if $P$ is a zero process and $P_.level.BFS \leq t$, then, for each $0 \leq i < P_.level.BFS$, there is some zero process $Q$ such that $Q_.level.BFS = i$.

**Proof.** By double induction, on increasing $P_.level.BFS$ and decreasing $i$.

If $P_.level.BFS = 0$, the statement is vacuous.

Suppose $P_.level.BFS > 0$ and $i = P_.level.BFS - 1$. Pick some zero process $Q \in N_P$ such that $Q_.level.BFS < P_.level.BFS$. (If no such $Q$ exists, then $P$ is invalid, which contradicts Lemma 8.1.) If $Q_.level.BFS < i$, then $Q$ is zero active, which contradicts Lemma 8.3, and thus we are done.

Suppose $0 \leq i < P_.level.BFS - 1$. By the inductive hypothesis, there exists a zero process whose level is $P_.level.BFS - 1$. Again by the inductive hypothesis, there exists a zero process whose level is $i$, and we are done. □

**Lemma 8.5** If at least $n$ rounds have elapsed, then

(a) there is no active zero process;

(b) no process which is not already a zero process may become a zero process;

(c) every zero process has level less than $n$.

**Proof.** We prove (a) by contradiction. Suppose that at least $n$ rounds have elapsed.

If $P$ is an active zero process, then $P_.level.BFS \geq n - 1$, by Lemma 8.3. By Lemma 8.4, there are at least $n - 1$ distinct processes whose BFS.key is less than $P_.BFS.key$. By the definition of zero active, there must be at least one process whose BFS.key is greater than $P_.BFS.key$. There are thus at least $n + 1$ processes in the network, contradiction. Part (b)
follows immediately.

We prove (c) by contradiction. Let \( i \) be the smallest non-negative integer such that there is no zero process of level \( i + 1 \). Since the there are \( n \) processes altogether, \( i < n \). Since there must be a zero process of level at least \( n \), we can pick \( j \) to be the smallest integer \( j > i + 1 \) such that there is a zero process \( P \) of level \( j \). Again, since there are at most \( n \) processes, \( i < n - 1 \). By Lemma 8.1, there exists a zero process \( Q \in \mathcal{N}_P \) such that \( Q.\text{level} < j \). Thus, \( Q \) is zero active, and \( Q.\text{level} \leq i < n - 1 \), which contradicts Lemma 8.3. □

**Lemma 8.6**

(a) If \( t \geq 0 \), and if at least \( n + t \) rounds have elapsed, then \( P.\text{level.BFS} < n - t \) for any zero process \( P \).

(b) If at least \( 2n \) rounds have elapsed, there is no zero process.

(c) If at least \( 2n \) rounds have elapsed, \( P.\text{leader.BFS} \geq \text{Root.id} \) for any process \( P \).

(d) If at least \( 2n \) rounds have elapsed, and if the distance from \( P \) to \( \text{Root} \) is at least \( i \), then \( P.\text{BFS.key} \geq (\text{Root.id}, i) \).

(e) If at least \( 2n \) rounds have elapsed, \( \text{Root.BFS.key} = \text{Root.self} \).

(f) If at least \( 2n + i \) rounds have elapsed, and if the distance from \( P \) to \( \text{Root} \) is \( i \), then \( P.\text{BFS.key} = (\text{Root.id}, i) \).

**Proof.** Note that (b) follows immediately from (a); we prove (a) by induction on \( t \). The case \( t = 0 \) is Lemma 8.5(c). Suppose \( t > 0 \). After \( n + t - 1 \) rounds, the largest possible level of any zero process is \( n - t \), by the inductive hypothesis. Thus, by Lemma 8.3, any process which is a zero process when \( n + t - 1 \) are completed, will execute Action Aact: reset sle during the next round, and we are done.

We prove (c) by contradiction. Let \( P \) be the process of smallest \( \text{BFS.key} \). By (b),
$P.leader.BFS > 0$. If $P.leader.BFS < P.id$, then $P$ is invalid, contradicting Lemma 8.1.

Thus, $P.leader.BFS \geq P.id \geq Root.id$.

We prove (d) by induction on $i$. By (c), $Root.self$ is the minimum possible $BFS.key$. If $i = 0$, the result follows.

Let $i > 0$. By the inductive hypothesis, $Q.BFS.key \geq (Root.id, i - 1)$ for all $Q \in N_P$. If $P.leader.BFS = Root.id$ and $P.level.BFS < i$, then $P$ is invalid, contradicting Lemma 8.1.

We prove (e) by contradiction. If $Root.leader.BFS > Root.id$, then $Root$ is invalid, contradicting Lemma 8.1. Thus, by (c), $Root.leader.BFS = Root.id$. If $Root.level.BFS > 0$, then there must exist $Q \in N_P$ such that $Q.leader.BFS = P.id$ and $Q.level.BFS < P.level.BFS$.

Let $R$ be the process of minimum level such that $R.leader.BFS = Root.id$. Since $R \neq Root$, $R$ must be invalid, contradicting Lemma 8.1.

We prove (f) by induction on $i$. By (d), we need only prove $P.BFS.key \leq (Root.id, i)$.

The case $i = 0$ is simply (e). Let $i > 0$. By (d), we need only prove $P.BFS.key \leq (Root.id, i)$.

Pick $Q \in N_P$ such that the distance from $Q$ to $Root$ is $i - 1$. By the inductive hypothesis, $Q.BFS.key = (Root.id, i - 1)$ after $2n + i - 1$ rounds have elapsed, and by Lemma 8.6, $P$ cannot be a frozen process. Thus, either $P.BFS.key \leq (Root.id, i)$ after $2n + i - 1$ rounds, or $P$ is enabled to execute Action $A2$ during the $(2n + i)^{th}$ round by linking to that neighbor of smallest $BFS.key$, which can be at most $Q.BFS.key$. We are done. □

**Theorem 8.1** Within $2n + \text{diam}$ rounds, the output specification of BFS will be satisfied.

**Proof.** By Lemma 8.6(f), the values of $P.BFS.key$ are correct for all $P$. If the distance from $P$ to $Root$ is $i > 0$, then $P.parent.BFS$ is correct, since, if $Q$ is the correct parent of $P$, $Q.BFS.key$ was correct after $2n + i - 1$ rounds, □
8.1.6 An $\Omega(n)$-Round Example

In our second example, BFS converged in 19 steps for a network where $diam = 9$ and $n = 59$. One might then wonder whether Theorem 8.1 can be improved to match the trivial lower bound for the problem, i.e., whether the number of rounds is actually $O(diam)$. The answer is no. We give an example of a network where $diam = 2$ and $n$ is arbitrarily large, and a computation of BFS on that network that takes $2n - 1$ rounds.

Let the network consists of processes $P_1, \ldots, P_n$, where $P_i.id = i$. There are $2n - 3$ edges, namely $\{P_i, P_n\}$ for all $1 \leq i \leq n - 1$, $\{P_i, P_{i+1}\}$ for all $2 \leq i \leq n - 2$, and $\{P_1, P_{n-1}\}$. We choose the initial configuration as follows:

$P_i.leader.BFS = 0$ and $P_i.level.BFS = 0$.

$P_i.leader.BFS = 2$ and $P_i.level.BFS = i - 2$ if $2 \leq i \leq n - 1$.

$P_n.leader.BFS = n$ and $P_n.level.BFS = 0$.

Figure 8.4(a) shows the network and the initial configuration in the case that $n = 6$.

We choose the computation where every process executes every round if enabled. $P_n$ is a frozen process for the first $2n - 3$ configurations since it a self root and is adjacent to the zero root $P_1$. Therefore, for the first $2n - 3$ rounds, BFS executes only on the subnetwork consisting of all processes except $P_n$. This subnetwork is a chain consisting of $n - 1$ processes, and thus has diameter $n - 2$. Phases 2 must traverse the chain from $P_{n-1}$ to $P_2$, after which Phase 3 must traverse the chain in the other direction, from $P_2$ to $P_1$.

Phase 1 is vacuous, since there are no initially invalid processes. Phase 2 takes $n - 2$ rounds to complete. Figure 8.4(e) shows the resulting configuration. Phase 3 takes an additional $n - 1$ rounds to complete, after which $P_n$ is able to execute for the first time. At that point, since $P_n$ is able to participate, Phase 4 is then completed in two additional
Figure 8.4: An $\Omega(n)$-Time Example.
rounds. The final configuration is:

\[ P_i.\text{leader}\_\text{BFS} = 1 \text{ for all } i. \]

\[ P_i.\text{level}\_\text{BFS} = 0. \]

\[ P_i.\text{level}\_\text{BFS} = 2 \text{ if } 2 \leq i \leq n - 1. \]

\[ P_{n-1}.\text{level}\_\text{BFS} = P_n.\text{level}\_\text{BFS} = 1. \]

Figure 8.4 shows the computation in the case \( n = 6 \). We use the same legend as in Figure 8.3. Figure 8.4(a) shows the initial configuration. Figure 8.4(e) shows the configuration after Phase 2 is complete. Figure 8.4(j) shows the configuration after Phase 3 is complete, and Figure 8.4(l) shows the final configuration.

8.2 The Module MIS

The module MIS constructs a maximal independent set (MIS) of the network, namely as set of processes, \( S \), which has the following properties:

1. \( S \) is independent, i.e., no two members of \( S \) are adjacent.

2. \( S \) is maximal, i.e., \( S \) is not the proper subset of any other independent set of processes.

MIS also constructs an MIS spanning tree \( T_{\text{MIS}} \), which has the following properties:

3. \( T \) is rooted at \( \text{Root} \), the root of the BFS tree, and the process of smallest ID.

4. \( T \) is 2-colored, i.e., every process has color 0 or 1, and no two processes of the same color are adjacent in \( T_{\text{MIS}} \). The members of \( S \) have color 0, and the other processes have color 1. Note that two processes of color 1 can be adjacent in the network, but they are not adjacent in \( T_{\text{MIS}} \).

5. The height of \( T_{\text{MIS}} \) is at most twice the diameter of the network.
8.2.1 Overview of MIS

We describe a sequential construction of $S$ which emulates MIS. Let $P_1, P_2, \ldots, P_n$ be the processes of the network, ordered lexically by $LevelKey(P) = (P.level_{BFS}, P.id)$. Define:

$$S_0 = \emptyset$$

For $i$ from 1 to $n$: 

$$S_i = \begin{cases} 
S_{i-1} \cup \{P_i\} & \text{if } S_{i-1} \cap N_{P_i} = \emptyset \\
S_{i-1} & \text{otherwise}
\end{cases}$$

Let $S = S_n$. Note that $Root = P_1 \in S$. In Lemma 8.7, we will show that $S$ is a maximal independent set.

If the BFS module became silent, i.e., if at least $2n + \text{diam}$ rounds have been completed after the initialization, the guard of Action 1 for $P_i$ depends only on the values of variables of $P_j$ only for $j < i$. Thus, within $i$ rounds after completion of BFS, the values of $P_i.color$ and $P_i.parent_{\text{MIS}}$ must be stable.

At the conclusion of MIS, at most $n$ rounds after BFS is done, the MIS spanning tree $T_{\text{MIS}}$, which has the same root as $T_{\text{BFS}}$, has been built. This tree is 2-colored; $Root.color = 0$, and any child of any process has the opposite color.

8.2.2 Variables, Functions, and Action

Each process $P$ has the following variables.

$P.color \in \{0, 1\}$. $P.color$ will be set to 0 if and only if $P$ is chosen to be a member of $S$.

$P.parent_{\text{MIS}}$ of ID type, which will be set to the parent of $P$ in $T_{\text{MIS}}$.

We define the following functions that can be computed by a process $P$. 

77
LevelKey(P) = (P.levelBFS, P.id), the level key of P. We order level keys lexically.

\[ A(P) = \{ Q \in N_P : \text{LevelKey}(Q) < \text{LevelKey}(P) \} \]

Color(P) = \[
\begin{cases}
1 & \text{if } \exists Q \in A(P) : Q.\text{color} = 0 \\
0 & \text{otherwise}
\end{cases}
\]

MIS-Key(P) = (P.color, P.levelBFS, P.id), the MIS key of P. We order MIS keys lexically.

Parent_Mis(P) = \[
\begin{cases}
P & \text{if } A_P = \emptyset \\
Q \in A_P \text{ of minimum MIS-Key} & \text{otherwise}
\end{cases}
\]

---

<table>
<thead>
<tr>
<th>B1 Update Priority 4 Color and MIS Parent</th>
<th>P.color \neq Color(P) \lor P.\text{parent Mis} \neq Parent_Mis(P)</th>
<th>\rightarrow P.color \leftarrow Color(P) P.\text{parent Mis} \leftarrow Parent_Mis(P)</th>
</tr>
</thead>
</table>

---

8.2.3 An Example Computation

Consider our standard graph. Figure 8.5(a) shows the BFS spanning tree \( T_{\text{BFS}} \), where levels are indicated by dashed polygonal lines. Figure 8.5(b) shows the MIS spanning tree, \( T_{\text{MIS}} \). The dashed polygonal lines still represent levels in \( T_{\text{BFS}} \).

Figure 8.5(c) shows six partial configurations, restricted to the subgraph consisting of those processes whose BFS levels are at most 2. Two names are shown for each process, the IDs and the designations \( P_1, \ldots P_n \), using LevelKey order; e.g., \( P_4 \) is process 54.

The first diagram (i) in Figure 8.5(c) shows the subgraph if we assume that the BFS
has stabilized, and the variables of MIS have their "clean" values, i.e., \( P.color = 1 \) and \( P.parent_{\text{MIS}} = P \) for all \( P \). After five rounds, under the same synchrony assumption we made for BFS, the variables of MIS have stabilized in that subgraph. This partial \( T_{\text{MIS}} \) is shown in Figure 8.5(c)(vi). In Figure 8.5(c)(i), all processes are enabled to execute Action B1. As all of them are in color 1, they all become MIS processes by executing Action B1 as shown in Figure 8.5(c)(ii). Although all processes have executed in the first round, a subset of processes (\( P_1, P_6, P_7, P_9 \), and \( P_{10} \)) have reached their stable values.

Note that the module MIS guarantees the stability of the values of \( P_1 \) only at the end of the first round. In subsequent rounds, the processes that are enabled for Action B1 correct their variables.

MIS stabilizes for the entire graph (Figure 8.5(a)) in 14 rounds under the same assumptions. If the standard graph is initialized to a clean configuration, as in Figure 8.2, and the MIS variables are initialized to their clean values, and if BFS and MIS will be computed concurrently, according to the priority rules given in Tables 8.1 and 8.2, then, under the synchrony assumption, BFS will stabilize in eight rounds, as stated in Section 8.1.3, and MIS will stabilize in five additional rounds, i.e., 13 rounds after initialization.

Similarly, if the standard graph is initialized as in Figure 8.3, and other assumptions are the same, BFS will stabilize in 19 rounds, as stated in Section 8.1.3, and MIS will stabilize in five additional rounds, i.e., 24 rounds after initialization.

8.2.4 Proofs of Correctness for MIS

Let \( \text{Level}(P) \) to be the distance from a process \( P \) to \( \text{Root} \). From Theorem 8.1:

**Remark 8.1** If BFS is silent, then \( P.\text{level}_{\text{BFS}} = \text{Level}(P) \) for any process \( P \),
Figure 8.5: (a) shows a BFS tree of an example graph. (b) shows the MIS tree constructed by MIS, where members of the MIS are circled. The BFS levels are separated by dashed polygonal lines. (c) shows steps of the MIS computation starting from a clean state after BFS levels are stable.
Let \( P_1, \ldots, P_n \) be the ordering of the processes by \( LevelKey \), and let \( S_i \subseteq \{P_1, \ldots, P_i\} \) be as defined in Subsection 8.2.1, and \( S = S_n \). Note that \( S_i = S \cap \{P_1, \ldots, P_i\} \).

**Lemma 8.7** \( S \) is a maximal independent set of processes.

**Proof.** Suppose \( P_i \) and \( P_j \) are adjacent, for \( i \neq j \), and \( P_i, P_j \in S \). Without loss of generality, \( i < j \). Then \( P_i \in S_i \subseteq S_{j-1} \), which implies that \( P_j \notin S_j \), contradiction. Thus, \( S \) is independent.

Suppose \( S \) is not maximal. Then \( S \cup \{P_k\} \) is independent, for some \( P_k \notin S \). By definition, \( P_k \) is adjacent to some \( P_i \in S \) such that \( i < k \), contradiction. \( \square \)

**Lemma 8.8** Let \( 1 \leq i \leq n \).

(a) If at least \( 2n + \text{diam} + i \) rounds have elapsed, then \( \text{Color}(P) = 0 \) if and only if \( P_i \in S \).

(b) If at least \( 2n + \text{diam} + i + 1 \) rounds have elapsed, then \( P_i.\text{color} = 0 \) if and only if \( P_i \in S \).

**Proof.** By induction on \( i \). \( P_1 = \text{Root} \), and \( \text{Color}(\text{Root}) = 0 \) provided BFS has stabilized. Within one more round, \( \text{Root}.\text{color} = 0 \), by the definition of Action B1.

Suppose \( i > 0 \). We first prove (a). By the definition of \( S_i \) and inductive hypothesis, \( P_j.\text{color} \) has stabilized for all \( j < i \), and thus \( \text{Color}(P_i) = 0 \) if and only if \( P_i \in S \). By the definition of Action B1, (b) follows. \( \square \)

**Lemma 8.9** Let \( 1 \leq i \leq n \).

(a) If at least \( 2n + \text{diam} + i \) rounds have elapsed, then \( \text{Parent}.\text{MIS}(P_i) \) has stabilized.

(b) If at least \( 2n + \text{diam} + i + 1 \) rounds have elapsed, then \( P_i.\text{parent}.\text{MIS} \) has stabilized, and \( P_i.\text{parent}.\text{MIS} = \text{Parent}.\text{MIS}(P_i) \).

81
Proof. Part (a) follows from the fact that Lemma 8.8(b) holds for all \( j < i \). Part (b) follows immediately from the definition of Action B1. 

By Theorem 8.1 and Lemmas 8.7 and 8.8, we immediately have:

**Theorem 8.2** Within \( 3n + \text{diam} + 1 \) rounds of initialization

(a) the module MIS is silent;

(b) the pointers \( \{P.\text{parent}_\text{MIS}\} \) define a spanning tree, \( T_{\text{MIS}} \), of the network;

(c) \( P \in S \) if and only if \( P \) is at an even level of \( T_{\text{MIS}} \);

(d) \( S \) is a minimal dominating set of the network;

(e) any path in \( T_{\text{MIS}} \) of length \( \ell \) contains at least \( \left\lceil \frac{\ell}{2} \right\rceil \) members of \( S \);

Proof. Part (a) follows from Lemmas 8.8 and 8.9. Part (b) follows from the fact that for any \( i > 1 \), \( \text{Parent}_\text{MIS}(P_i) = P_j \) for some \( j < i \). Part (c) is by induction on the level of \( P \) in \( T_{\text{MIS}} \). \( \text{Root} \in S \) is the only process at level 0. If \( P \) is at level \( \ell > 0 \), the \( P.\text{parent}_\text{MIS} = Q \) for some \( Q \) at level \( \ell - 1 \). By Lemma 8.9 and the definition of \( \text{Parent}_\text{MIS} \), \( P.\text{color} \neq Q.\text{color} \). Since (c) holds for \( Q \) by the inductive hypothesis, it holds for \( P \). Part (d) follows from Lemma 8.7, since a maximal independent set of any graph is also a minimal dominating set. Part (e) follows from (c). 

8.3 The Module CLSTR

The final module of BFS-MIS-CLSTR is CLSTR, which constructs a \( k \)-clustering of the network, given that the MIS tree \( T_{\text{MIS}} \) has been constructed. CLSTR consists of two phases. The first phase chooses a set of clusterheads, while the second phase builds \( k \)-clusters around those clusterheads.

During the first phase, CLSTR computes \( k \)-dominating set of the MIS spanning tree,
\( \mathcal{T}_{\text{MIS}} \), which is then necessarily a \( k \)-dominating set of the network. The members of that set will be the clusterheads.

8.3.1 Overview of CLSTR

**First Phase and Computation of \( \alpha \).** Our method is to first define an abstract function \( \alpha(x) \) on all nodes of any rooted tree, \( \mathcal{T} \). This function takes integral values in the range \( 0 \ldots 2k \). The values of \( \alpha \) are defined recursively, in bottom-up fashion. Using \( \alpha \), we then define \( \text{OptDom}(\mathcal{T}) \), an optimum \( k \)-dominating set of \( \mathcal{T} \).

The recursive definition of \( \alpha \) given below seems somewhat non-intuitive; we shall give the intuition later.

1. If \( x \) is a leaf, \( \alpha(x) = 0 \).

2. For any node \( x \), we say that \( x \) is short if \( \alpha(x) < k \), and tall if \( \alpha(x) \geq k \).

3. If \( x \) is not a leaf, define \( \text{Maxshort}(x) \) to be the maximum value of \( \alpha(y) \) for all short children \( y \) of \( x \). If \( x \) has no short children, we define \( \text{Maxshort}(x) = -1 \).

4. If \( x \) is not a leaf, define \( \text{Mintall}(x) \) to be the minimum value of \( \alpha(y) \) for all tall children \( y \) of \( x \). If \( x \) has no tall children, we define \( \text{Mintall}(x) = \infty \).

5. If \( \text{Maxshort}(x) + \text{Mintall}(x) \leq 2k - 2 \), let \( \alpha(x) = \text{Mintall}(x) + 1 \).

6. If \( \text{Maxshort}(x) + \text{Mintall}(x) > 2k - 2 \), let \( \alpha(x) = \text{Maxshort}(x) + 1 \).

\( \text{OptDom}(\mathcal{T}) \) is defined to be the set of all nodes \( x \) such that either \( \alpha(x) = k \), or \( \alpha(x) < k \) and \( x = \text{root} \). We will prove, in Theorem 8.3, that \( \text{OptDom}(\mathcal{T}) \) is a \( k \)-dominating set of \( \mathcal{T} \), and is optimum, i.e., has the smallest cardinality of any \( k \)-dominating set of \( \mathcal{T} \). Figure 8.6 shows the computation of \( \alpha \) for a rooted tree.
We now give the intuition behind the definition of the function $\alpha(x)$. The construction of $OptDom(T)$, given above, is a lazy algorithm. As we move up the tree, we only choose a node $x$ to be a member of $OptDom$ if we are absolutely forced to do so, i.e., given the choices we have made in the subtree $T_x$ so far, inclusion of $x$ is mandatory for $OptDom$ to be a $k$-dominating set. In making that decision, we know whether $x$ is the root, and we know the subtree history, namely the choices we have made for the proper descendants of $x$, but we do not know choices in other parts of $T$.

The function $\alpha(x)$ is a finite "fingerprint" of the subtree history; and it carries all the information about that history that the algorithm needs to make its decision, as well as to pass on to $\text{parent}(x)$.

The recursive definition of $\alpha(x)$ is, in fact, a form of dynamic programming. Those parts of the subtree histories of the children of $x$ which are no longer needed are discarded; $\alpha(x)$ holds all the information that will be needed at $x$ and above.

The values of $\alpha$ can be interpreted naturally as follows. Let $P_x = T_x - \{x\}$, the set of proper descendants of $x$. If $\alpha(x) > k$, every member of $T_x$ is within $k$ hops of some member of $T_x \cap OptDom$, and the nearest member of $P_x \cap OptDom$ to $x$ is $\alpha(x) - k$ levels below $x$. If $\alpha(x) \leq k$, every member of $T_x$ which is more than $\alpha(x)$ levels below $x$ is within $k$ hops of some member of $T_x \cap OptDom$, and there is some $y \in T_x$ which is $\alpha(x)$ levels below $x$ and which is not within $k$ hops of any member of $P_x \cap OptDom$.

Thus, if $\alpha(x) = k$, we must place $x$ into $OptDom$. If $\alpha(x) > k$, we do not place $x$ into $OptDom$, since there is no need. If $\alpha(x) < k$, we must ensure that there is a path of length $k - \alpha(x)$ from $x$ to some member of $OptDom$, and no such member exists within $P_x$. Thus, if $\alpha(x) < k$, and $x$ is the root, $x$ must be placed in $OptDom$, while if $x$ is not the root, we push the responsibility for ensuring existence of that path onto the parent of $x$. 

84
$x$, by requiring that $\alpha(x) + 1 \leq \alpha(\text{parent}(x)) \leq 2k - \alpha(x) - 1$.

The set of clusterheads computed by CLSTR, using Action C1, is $\text{OptDom}(T_{\text{MIS}})$, where $T_{\text{MIS}}$ is as computed in module MIS. It is important to note that this set is not necessarily an optimum $k$-dominating set of the original network.

**Theorem 8.3**

(a) $\text{OptDom}$ is a $k$-dominating set of $T$,

(b) $\text{OptDom}$ has the smallest cardinality of any $k$-dominating set of $T$.

**Proof.** (a): We will prove, by induction on $\beta(x)$, that any node $x$ of $T$ is within $\beta(x)$ of some member of $\text{OptDom}$. Part (a) follows.

If $\beta(x) = 0$, then $x \in \text{OptDom}$. Suppose $\beta(x) > 0$. We will prove that either $x \in \text{OptDom}$, or there is some $y \in N_x$ such that $\beta(y) \leq \beta(x) - 1$, completing the inductive step.

Case: $\alpha(x) > k$. Then, $\alpha(x) = \alpha(y) + 1$ for some $y$ which is a child of $x$. Thus, $\beta(y) = \beta(x) - 1$.

Case: $\alpha(x) < k$. If $x$ is the root of $T$, then we are done. Otherwise, let $y$ be the parent of $x$. and let $u = \text{Maxshort}(x)$ and $v = \text{Mintall}(x)$. Then $\alpha(y) \in \{u + 1, v + 1\}$.

If $\alpha(y) = u + 1$, then $\beta(y) = k - u - 1 \leq k - \alpha(x) - 1 = \beta(x) - 1$.

If $\alpha(y) = v + 1$, then $u + v \leq 2k - 2$. Thus

$$\beta(y) = v - k + 1 \leq k - u - 1 = k - \alpha(x) - 1 = \beta(x) - 1$$

Part (b) follows from Lemma 8.11. □
Three Versions of the Second Phase. The second phase of CLSTR partitions the processes of the network into clusters, each of which contains one clusterhead. Each cluster contains a cluster spanning tree, a tree containing all the processes of that cluster, which is rooted at the clusterhead. Furthermore, the height of the cluster spanning tree is at most $k$.

We present three versions of the second phase of CLSTR. The first version uses Actions C1 and C2. This version simply uses the optimum clustering of $\mathcal{T}_{\text{MIS}}$. Thus, each cluster spanning tree is a subgraph of $\mathcal{T}_{\text{MIS}}$, possibly with the directions of some edges reversed.

The second version uses Actions C1, C2, and C3. The processes in the clusters are exactly the same as in the first version, but the cluster spanning trees are optimized, i.e., made into BFS trees, to improve communication. Note that both the second and third versions maintain BFS trees inside each cluster. However, in the second version, while constructing the BFS trees, only the subgraphs of $\mathcal{T}_{\text{MIS}}$ are used. In the third version, the BFS trees are constructed on the subgraphs of the whole (original) graph. Thus, the heights of the cluster trees are expected to be the smallest in the third version.
The third version uses Actions C1 and C4. The same clusterheads are used, but the clusters are chosen differently. To further improve communication, each process joins the nearest clusterhead, and the cluster spanning trees are BFS trees.

8.3.2 Variables, Functions, and Actions of CLSTR

We now list the variables of each process that are used by the various versions of the module CLSTR.

- \( P.a \), an integer in the range 0...2k. This variable is used by all three versions.
- \( P.level\_CLR-II \), a non-negative integer. This variable is used by CLSTR-II only.
- \( P.level\_CLR-III \), a non-negative integer. This variable is used by CLSTR-III only.
- \( P.parent\_CLR-I \) of ID type, the parent of \( P \) in the cluster spanning tree of the clusterhead of the cluster \( P \) belongs to. This variable is used by CLSTR-I and CLSTR-II.
- \( P.parent\_CLR-II \) of ID type. This variable is used by CLSTR-II only.
- \( P.parent\_CLR-III \) of ID type. This variable is used by CLSTR-III only.
- \( P.leader\_CLR-I \) of ID type, the leader of \( P \) in the cluster spanning tree of the clusterhead of the cluster \( P \) belongs to. This variable is used by CLSTR-I and CLSTR-II.
- \( P.leader\_CLR-III \) of ID type. This variable is used by CLSTR-III only.

Each of the following functions can be computed locally, i.e., by the process \( P \) using only its own and its neighbors' variables.

- \( P.isshort \equiv P.a < k \), of Boolean type.
- \( P.istall \equiv P.a \geq k \), of Boolean type.
- \( P.isclstrhd \equiv (P.isshort \land (P.parent\_MIS = P.id)) \lor (P.a = k) \), of Boolean type.
- \( Shrtchldrn(P) = \{ Q : (Q.parent\_MIS = P) \land Q.isshort \} \)
\[ T_{allchild}(P) = \{ Q : (Q.\text{parent}_i = P) \land Q.\text{istall} \} \]
\[
Maxshort(P) = \begin{cases} 
\max \{ Q.\alpha : Q \in Shrtchildrn(P) \} & \text{if } Shrtchildrn(P) \neq \emptyset \\
-1 & \text{otherwise}
\end{cases}
\]
\[
Mintall(P) = \begin{cases} 
\min \{ Q.\alpha : Q \in Tallchildrn(P) \} & \text{if } Tallchildrn(P) \neq \emptyset \\
\infty & \text{otherwise}
\end{cases}
\]
\[
MintallID(P) = \begin{cases} 
\min \{ Q.\alpha : \left( Q \in Tallchildrn(P) \right) \land (Q.\alpha = \text{Mintall}(P)) \} & \text{if } Tallchildrn(P) \neq \emptyset \\
\bot & \text{otherwise}
\end{cases}
\]
\[
\text{Alpha}(P) = \begin{cases} 
\text{Mintall}(P) + 1 & \text{if } \text{Mintall}(P) + \text{Maxshort}(P) \leq 2k - 2 \\
\text{Maxshort}(P) + 1 & \text{otherwise}
\end{cases}
\]
\[
\text{Parent}_\text{CLR-I}(P) = \begin{cases} 
P.\text{parent}_i & \text{if } P.\alpha < k \\
P.\text{id} & \text{if } P.\alpha = k \\
\text{MintallID}(P) & \text{if } P.\alpha > k
\end{cases}
\]
\[
\text{Leader}_\text{CLR-I}(P) = \begin{cases} 
P.\text{id} & \text{if } P.\text{isclstrhd} \\
P.\text{parent}_i.\text{leader}_\text{CLR-I} & \text{otherwise}
\end{cases}
\]
\[
\text{Level}_\text{CLR-II}(P) = \begin{cases} 
0 & \text{if } P.\text{isclstrhd} \\
1 + \min \{ Q.\text{level}_\text{CLR-II} : (Q \in N_P) \land (Q.\text{leader}_\text{CLR-I} = P.\text{leader}_\text{CLR-I}) \} & \text{otherwise}
\end{cases}
\]
\[
\text{Parent}_\text{CLR-II}(P) = \begin{cases} 
P.\text{id} & \text{if } P.\text{isclstrhd} \\
\min \{ Q.\text{id} : (Q \in N_P) \land (Q.\text{level}_\text{CLR-II} + 1 = P.\text{level}_\text{CLR-II}) \land (Q.\text{leader}_\text{CLR-I} = P.\text{leader}_\text{CLR-I}) \} & \text{otherwise}
\end{cases}
\]
\[
\text{Level}_{\text{CL}R-\text{III}}(P) = \begin{cases} 
0 & \text{if } P\text{-isclstrhd} \\
1 + \min \{ Q.\text{level}_{\text{CL}R-\text{III}} : (Q \in N_P) \} & \text{otherwise}
\end{cases}
\]

\[
\text{Parent}_{\text{CL}R-\text{III}}(P) = \begin{cases} 
P.id & \text{if } P\text{-isclstrhd} \\
\min \{ Q.id : (Q \in N_P) \land (Q.\text{level}_{\text{CL}R-\text{III}} + 1 = P.\text{level}_{\text{CL}R-\text{III}}) \} & \text{otherwise}
\end{cases}
\]

\[
\text{Leader}_{\text{CL}R-\text{III}}(P) = \begin{cases} 
P.id & \text{if } P\text{-isclstrhd} \\
P.\text{parent}_{\text{CL}R-\text{III}}.\text{leader}_{\text{CL}R-\text{III}} & \text{otherwise}
\end{cases}
\]

We give the actions of CLSTR in Table 8.3.

<table>
<thead>
<tr>
<th>C1</th>
<th>Priority</th>
<th>Update</th>
<th>Action</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2</td>
<td>Priority 6</td>
<td>Update</td>
<td>Cluster-I</td>
<td>(P.parent_{\text{CL}R-I} \neq Parent_{\text{CL}R-I}(P)) \lor (P.\text{leader}<em>{\text{CL}R-I} \neq Leader</em>{\text{CL}R-I}(P))</td>
</tr>
<tr>
<td>C3</td>
<td>Priority 7</td>
<td>Update</td>
<td>Cluster-II</td>
<td>(P.\text{level}<em>{\text{CL}R-II} \neq \text{Level}</em>{\text{CL}R-II}(P)) \lor (P.\text{parent}<em>{\text{CL}R-II} \neq Parent</em>{\text{CL}R-II}(P))</td>
</tr>
<tr>
<td>C4</td>
<td>Priority 6</td>
<td>Update</td>
<td>Cluster-III</td>
<td>(P.\text{level}<em>{\text{CL}R-III} \neq \text{Level}</em>{\text{CL}R-III}(P)) \lor (P.\text{parent}<em>{\text{CL}R-III} \neq Parent</em>{\text{CL}R-III}(P)) \lor (P.\text{leader}<em>{\text{CL}R-III} \neq Leader</em>{\text{CL}R-III}(P))</td>
</tr>
</tbody>
</table>

89
Figure 8.7: Various Steps of BFS-MIS-CLSTR in an Example where $k = 4$
In Figure 8.7, we show all three versions of the second phase of CLSTR. Figure 8.7(a) shows $T_{\text{MIS}}$, the output of MIS, where members of the maximal independent set are circled. The dashed polygonal lines separate BFS levels. Figure 8.7(b) shows $T_{\text{MIS}}$, where the members of MIS are circled, together with the values of $\alpha$ computed by Action C1, where $\alpha(P)$ is indicated by a number enclosed in an oval. Note that, if $P$ is a leaf of the MIS tree, $\alpha(P) = 0$.

Figure 8.7(c) shows the clusterheads and clusters computed by Action C2. The clusterheads are shown as large dots. The clusters are separated by dashed lines. Each cluster spanning tree is a subgraph of $T_{\text{MIS}}$. Note that each cluster contains at least $\lceil \frac{n}{2} \rceil = 2$ members of the MIS.

Figure 8.7(d) shows the clusters computed by Action C3. The clusterheads and cluster boundaries are the same as in (c), but each cluster spanning tree is a BFS tree for its cluster, minimizing the distance from each process to its clusterhead.

For example, in 8.7(c), process 76 is three hops from its cluster head, process 31. In (d), this distance has been reduced to two.

Figure 8.7(e) shows the clusterheads and clusters computed by Action C4. The cluster spanning trees are again BFS trees. The clusterheads are the same as in (c) and (d), but each process links to the nearest clusterhead, further reducing the distances from processes to their clusterheads. For example, in 8.7(d), process 81 is three hops from its cluster head, process 10, while in 8.7(e), it joins the cluster headed by process 32, which is only one hop away.

For comparison, Figure 8.7(f) shows an optimal 4-clustering of the graph, which uses only two clusters.
8.3.3 Proofs of Optimality of $OptDom$

In this subsection, we let $T$ be any rooted tree, let $\alpha(x)$ be the recursively defined function and $OptDom$ the set of nodes of $T$ defined in 8.3.1. Recall that $x \in OptDom$ if and only if $(\alpha(x) = k) \lor ((\alpha(x) < k) \land (x = \text{root}))$. We also define $\beta(x) = |\alpha(x) - k|$ and

$$\delta(x) = \begin{cases} 
  k - \alpha(x) + 1 & \text{if } \alpha(x) \leq k \\
  \alpha(x) - k & \text{if } \alpha(x) > k
\end{cases}$$

for any node $x$. Intuitively, $\beta(x)$ is an upper bound on the distance from $x$ to the nearest member of $OptDom$, while $\delta(x)$ is a lower bound on the distance from $x$ to its nearest proper descendant (in $T$) which is a member of $OptDom$.

**Lemma 8.10** For any node $x$ of $T$:

(a) If $x$ is the parent of $y$, then $\delta(y) \geq \delta(x) - 1$.

(b) If $z \in (T_x \cap OptDom) - \{x\}$, then $level(z) - level(x) \geq \delta(x)$.

**Proof.** (a): Let $u = \text{Maxshort}(x)$ and $v = \text{Mintall}(x)$. Then $\alpha(y) \leq u$ or $\alpha(y) \geq v$.

Case: $u + v \leq 2k - 2$. Then $\alpha(x) = v + 1$, by Part 5 of the recursive definition of $\alpha(x)$, and $\delta(x) = \alpha(x) - k = v - k + 1$.

If $\alpha(y) \geq v$, then $\delta(y) \geq \alpha(y) - k \geq v - k = \delta(x) - 1$.

If $\alpha(y) \leq u$, then $\delta(y) \geq k - \alpha(y) + 1 \geq k - u + 1 \geq v - k + 3 > \delta(x) - 1$.

Case: $u + v > 2k - 2$. Then $\alpha(x) = u + 1$, by Part 6 of the recursive definition of $\alpha(x)$, and $\delta(x) = k - \alpha(x) + 1 = k - u$.

If $\alpha(y) \geq v$, then $\delta(y) \geq \alpha(y) - k \geq v - k \geq k - u - 1 = \delta(x) - 1$.

If $\alpha(y) \leq u$, then $\delta(y) \geq k - \alpha(y) + 1 \geq k - u + 1 > \delta(x) - 1$.

(b): By induction on $level(x) - level(y)$. If $level(x) - level(x) = 1$, then $\alpha(x) = k$ This implies that $Mintall(x) = k$. Let $u = \text{Maxshort}(x)$. If $u \leq k - 2$, then $\alpha(x) = Mintall(x) + 1 = k + 1$,
while if $u = k - 1$, then $\alpha(x) = \text{Mazshort}(x) + 1 = k$. In either case, $\delta(x) = 1$, and we are done.

Suppose $\text{level}(z) - \text{level}(x) > 1$. Pick $y$ to be the unique child of $x$ such that $z \in T_y$. By (a) and the inductive hypothesis:

\[
\begin{align*}
\text{level}(y) - \text{level}(x) &= 1 \\
\text{level}(z) - \text{level}(y) &\geq \delta(y) \\
\delta(y) &\geq \delta(x) - 1
\end{align*}
\]

Combining the above, $\text{level}(z) - \text{level}(x) \geq \delta(x)$. \hfill \Box

**Lemma 8.11** Let $\mathcal{B}$ be any $k$-dominating set of $\mathcal{T}$. Then $|\mathcal{B} \cap T_x| \geq |\text{OptDom} \cap T_x|$ for any node $x$.

**Proof.** By bottom-up induction on $x$.

**Case:** $x$ is a leaf.

If $\mathcal{T}$ consists of just one node, then $\text{OptDom} = \mathcal{B} = \{x\}$, and we are done. Otherwise, $\alpha(x) = 0$, and thus $\text{OptDom} \cap T_x = \emptyset$, and we are done.

**Case:** $x \not\in \text{OptDom}$, and $x$ is not a leaf.

Let $y_1, \ldots, y_m$ be the children of $x$. By the inductive hypothesis, $|\text{OptDom} \cap T_{y_i}| \leq |\mathcal{B} \cap T_{y_i}|$ for all $i$. Thus

\[
|\text{OptDom} \cap T_x| = \sum_{i=1}^{m} |\text{OptDom} \cap T_{y_i}| \leq \sum_{i=1}^{m} |\mathcal{B} \cap T_{y_i}| \leq |\mathcal{B} \cap T_x|
\]

**Case:** $x \in \text{OptDom}$, and $x$ is not a leaf.

93
Let $a = \alpha(x) \leq k$. Let $x_0, \ldots x_a = x$ be nodes such that $\alpha(x_i) = i$ and $x_{i+1}$ is the parent of $x_i$ for all $i$.

Let $\mathcal{D}$ be the set of nodes which are within $k$ hops of $x_0$. If $z \in \mathcal{D}$ and $z \neq x$, then $z \in \mathcal{T}_{x_i}$ for some $i$, and $z$ is at most $k - i$ levels below $x_i$. By Lemma 8.10, $z \notin \text{OptDom}$, since $\delta(x_i) = k - i + 1$. Thus, $\text{OptDom} \cap \mathcal{D} = \{x\}$.

Let $w_1, \ldots w_m$ be the nodes of $\mathcal{T}_x$ which are exactly $k + 1$ hops from $x_0$. Then $\mathcal{T}_x$ is the disjoint union of all the $\mathcal{T}_{w_i}$ and $\mathcal{D}$.

![Diagram](image.png)

Figure 8.8: Proof of Lemma 8.11. A Case where $k = 3$ is illustrated. The value of $\alpha(x)$ is shown for each Process $x$. The values of $\beta(x)$ and $\delta(x)$ are computed in the table. The dashed polygon encloses $\mathcal{D}$, the $k$-hop neighborhood of $x_0$. 

94
By the inductive hypothesis, $|\text{OptDom} \cap \mathcal{T}_{w_i}| \leq |\mathcal{B} \cap \mathcal{T}_{w_i}|$ for all $i$. Since $\mathcal{B}$ is a $k$-dominating set, $|\mathcal{B} \cap \mathcal{D}| \geq 1$, since otherwise $\mathcal{B}$ would not contain a node within $k$ hops of $x_0$. Thus

$$|\text{OptDom} \cap \mathcal{T}_x| = \sum_{i=1}^{m} |\text{OptDom} \cap \mathcal{T}_{w_i}| + 1 \leq \sum_{i=1}^{m} |\mathcal{B} \cap \mathcal{T}_{w_i}| + |\mathcal{B} \cap \mathcal{D}| = |\mathcal{B} \cap \mathcal{T}_x|$$

and we are done. $\square$

Figure 8.8 illustrates an example for the second case of the inductive step of the proof of Lemma 8.11, where $m = k = 3$. Values of $\alpha(x)$ are shown for each node $x$. The dashed polygon encloses $\mathcal{D}$.

### 8.3.4 Proofs for CLSTR

Throughout this section, we assume that BFS and MIS are silent. In particular, the MIS spanning tree, $\mathcal{T}_{\text{MIS}}$, has been constructed, and will not change. For any process $\mathcal{P}$, let $\text{Level}_{\text{BFS}}(\mathcal{P})$ be the level of $\mathcal{P}$ in $\mathcal{T}_{\text{BFS}}$, and let $\text{Level}_{\text{MIS}}(\mathcal{P})$ be the level of $\mathcal{P}$ in $\mathcal{T}_{\text{MIS}}$.

**Lemma 8.12** The height of $\mathcal{T}_{\text{MIS}}$ is at most $2\text{diam}$.

**Proof.** The height of $\mathcal{T}_{\text{BFS}}$ is at most $\text{diam}$, since it is a BFS spanning tree of the network. We show by induction on $\text{Level}_{\text{MIS}}(\mathcal{P})$ that $\text{Level}_{\text{BFS}}(\mathcal{P}) \geq \lfloor \text{Level}_{\text{MIS}}(\mathcal{P})/2 \rfloor$ for any $\mathcal{P}$.

If $\text{Level}_{\text{MIS}}(\mathcal{P}) = 0$, then $\mathcal{P} = \text{Root}_{\text{BFS}}$, and we are done. Suppose $\text{Level}_{\text{MIS}}(\mathcal{P}) = \ell > 0$.

If $\ell$ is odd, then we are done, since $\lfloor \ell/2 \rfloor = (\ell - 1)/2 = \lfloor (\ell - 1)/2 \rfloor$. If $\ell$ is even, then $\ell$ has color 0, and $\text{Parent}_{\text{MIS}}(\mathcal{P}) = \text{Parent}_{\text{BFS}}(\mathcal{P}) = Q.id$. Thus, $\text{Level}_{\text{BFS}}(\mathcal{P}) = 1 + \text{Level}_{\text{BFS}}(Q) \geq 1 + \lfloor (\ell - 1)/2 \rfloor = \ell/2 = \lfloor \ell/2 \rfloor$. $\square$
Lemma 8.13 Let $\text{Level.MIS}(P) = \ell$. If at least $2\text{diam} - \ell + 1$ rounds have elapsed, then $P.\alpha$ has stabilized.

Proof. By backwards induction on $\ell$. The case $\ell = 2\text{diam} + 1$ is vacuous, by Lemma 8.12. Let $\ell \leq 2\text{diam}$. The value of $\text{AlphaF}(P)$ depends only on the values of $Q.\alpha$ for all $Q$ such that $\text{Level.MIS}(Q) = \text{Level.MIS}(P) + 1$. By the inductive hypothesis, all those values stabilize within the first $2\text{diam} - \ell$ rounds. Thus, either $P.\alpha$ stabilizes within $2\text{diam} - \ell$ rounds, or Action C1 executes during the $(2\text{diam} - \ell + 1)^{th}$ round, after which $P.\alpha$ is silent.

Lemma 8.14 Let $\text{Level.MIS}(P) = \ell$. If at least $2\text{diam} - \ell + 2$ rounds have elapsed, then $P.\text{parent.CLR-1}$ has stabilized.

Proof. By the inductive hypothesis, Lemma 8.13, and the definitions of $\text{Parent.CLR-1}(P)$ and Action C2.

Lemma 8.15 Let $\text{Level.MIS}(P) = \ell$. If at least $2\text{diam} - \ell + 2\beta(P) + 3$ rounds have elapsed, then $P.\text{leader.CLR-1}$ has stabilized.

Proof. By induction on $\beta(P)$. If $\beta(P) = 0$, then $P$ is a clusterhead. By Lemma 8.14, all actions with priority numbers less than 6 have stabilized by $2\text{diam} - \ell + 2$ rounds. Within one more round, Action C2 will have executed, and we are done.

Suppose $\beta(P) > 0$. Pick $Q \in N_P$ such that $\beta(Q) < \beta(P)$ and $Q$ is either a parent or a child of $P$ in $T_{\text{bfs}}$. By the inductive hypothesis, $Q.\text{leader.CLR-1}$ stabilizes within $2\text{diam} - \ell + 2\beta(P) + 2$ rounds. Within one more round, Action C2 will have executed, and we are done.

Lemma 8.16 If at least $2\text{diam} + 3 + k$ rounds have elapsed, then Action C3 is silent for
all processes

Proof. By Lemma 8.15, Action C2 is silent for all processes after $2 \text{diam} + k$ rounds. Action C3 is simple flooding starting from the clusterheads, and the radius of every cluster is at most $k$. □

Lemma 8.17 If at least $2 \text{diam} + 4 + 2k$ rounds have elapsed, then Action C4 is silent for all processes

Proof. By Lemma 8.16, Action C3 is silent for all processes after $2 \text{diam} + 2k + 1$ rounds. Action C4 is simple flooding starting from the clusterheads, and the radius of every cluster is at most $k$. □

Theorem 8.4

(a) Within $O(\text{diam})$ rounds after MIS has stabilized, clusterheads will be chosen.

(b) Within $O(\text{diam})$ rounds after clusterheads have been chosen, clusters will be constructed.
COMPETITIVENESS OF BFS-MIS-CLSTR

We now consider the special case, also considered by Fernandess and Malkhi, that our network is a unit disk graph in the plane. That is, the processes are placed at points in the Euclidean plane, and two processes can communicate if and only if their distance is at most 1.

**Lemma 9.1** Suppose that the network is a connected unit disk graph in the plane. \( M_{BMC} \) be the number of clusters in the \( k \)-clustering constructed by BFS-MIS-CLSTR, and let \( M_{OPT} \) be the number of clusters of the optimum \( k \)-clustering. Then, there is a constant \( K \), not dependent on \( k \) and not dependent on the network, such that \( M_{BMC} \leq (7.25552k + K)M_{OPT} \).

In order to prove Lemma 9.1, we make use of a result by Folkman and Graham [30]. If \( X \) is a bounded subset of the plane, define \( p(X) \) to be the maximum cardinality of any set \( S \subseteq X \) such that the distance between any two distinct points of \( S \) is at least 1. Let \( f(r) = p(D_r) \), where \( D_r \) is a disk of radius \( r \).

**Theorem 9.1 (Folkman and Graham)** If \( X \) is a compact convex region in the plane and \( S \subseteq X \), and if the distance between any two distinct members of \( S \) is at least 1, then the cardinality of \( S \) is at most \( \left\lfloor \frac{2}{\sqrt{3}} A(X) + \frac{1}{2} P(X) + 1 \right\rfloor \), where \( A(X) \) and \( P(X) \) are the area and perimeter of \( X \), respectively.

**Corollary 1** \( f(r) \leq \frac{2\pi r^2}{\sqrt{3}} + \pi r + 1 \).
Precise values of $f(r)$ for small $r$ can be found in [35]. For large values of $r$, the estimate given in Corollary 1 is very close, viz., the error is $O(r)$.

**Proof.** (Lemma 9.1)

Since $S$ is an independent set, the distance between any two members of $S$ in the plane must be larger than 1. Let $C$ be any one of the clusters constructed by the optimum $k$-clustering algorithm, and let $x$ be the point in the plane where the clusterhead of $C$ is located. All members of $C$ are located within a disc of radius $k$ centered at $x$. Thus, by Corollary 1, no more than $2\pi k^2 / \sqrt{3} + \pi k + 1$ members of $S$ can be in $C$. It follows that the cardinality of $S$ is at most $M_{OPT}\left(\frac{2\pi k^2}{\sqrt{3}} + \pi k + 1\right)$.

Let $C_1, \ldots, C_m$ be the $k$-clusters constructed by BFS-MIS-CLSTR, using CLSTR $- I$, where $m = M_{BMC}$. By Theorem 8.2(e), each $C_i$ either $C$ contains Root or $C$ contains at least $\lfloor k \rfloor 2$ members of $S$. It follows that the cardinality of $S$ must be at least $1 + \frac{k}{2}(M_{BMC} - 1)$. Through routine calculation, we obtain $M_{BMC} \leq 1 + \left(\frac{4\pi}{\sqrt{3}} k + 2\pi + \frac{2}{k}\right) M_{OPT}$, and $\frac{4\pi}{\sqrt{3}} \approx 7.2552$. If a different version of CLSTR is used, the number of clusters constructed is the same. □

We point out that the bound given in Lemma 9.1 could be improved by giving a more accurate estimate of $f(k)$.

### 9.0.5 Approximate Disk Graphs

More generally, if $V$ is a set of points in a metric space $M$, and $\lambda \geq 1$, we say that $G = (V, E)$ is an *approximate disk graph in $M$ with approximation ratio* $\lambda$ if there is a constant $c > 0$, the *lower distance*, such that any two points that are less than $c$ apart are adjacent in $G$ and any two points that are more than $\lambda c$ are not adjacent in $G$. That is,
for any $u, v \in V$,

$$||u, v|| < c \implies \{u, v\} \in E \implies ||u, v|| \leq \lambda c$$

For example, consider a scattered set of communication devices in rough terrain. The ability of two devices to communicate depends not only on distance, but also on nearby obstructions. Suppose two devices can always communicate if they are within $c_0$ of each other for some distance $c_0$, and can never communicate if they are farther than $c_1$ from each other for some larger distance $c_1$; while in the intermediate range of distances, they may or may not be able to communicate, depending on other factors. The resulting communication graph is an approximate disk graph in the plane with approximation ratio $\lambda = c_1/c_0$.

**Lemma 9.2** If $G$ is an approximate disk graph in the plane, with approximation ratio $\lambda$, then

$$M_{\text{BMC}} \leq 1 + \left( \frac{4\pi}{\sqrt{3}} \lambda^2 k + 2\pi \lambda + \frac{2}{k} \right) M_{\text{OPT}}$$

**Proof.** By Lemma 1, the number of members of $S$ in any $k$-cluster is no more than $f(\lambda k) \leq \frac{2\pi \lambda^2 k^2}{\sqrt{3}} + \pi \lambda r + 1$. The remainder of the proof is similar to that of Lemma 9.1. □

9.0.6 Bounded Independence Graphs

Given a graph $G$ and a function $f$, we say that $G$ is independence bounded by $f$ if, for any node $x$ of $G$ and any integer $k \geq 0$, the cardinality of the maximum independent subset of $U_k(x)$ is at most $f(k)$. If $G$ is a class of graphs, we say that $G$ is a $d$ power law class if every member of $G$ is independence bounded by some function $f$, where $f(k) = O(k^d)$.

**Lemma 9.3** If the network is a bounded independence graph, bounded by a function $f$, then

$$M_{\text{BMC}} \leq 1 + \frac{2f(k)}{k} M_{\text{OPT}}$$
Comparison with Fernandess-Malkhi. In [29], Theorem 3.4 claims that their algorithm is $8k + O(1)$-competitive, for the $k$-clustering problem if the network is a unit disk graph in the plane.¹

We also note that, although their result is correct, their proof contains a flaw. At one point, they state that a $2k$ by $2k$ square in the plane can hold at most $k^2$ unit disks. But by using a hexagonal packing, that square can hold $\frac{2\sqrt{3}}{3} \cdot k^2 - O(k) \approx 1.1547k^2$ disks. Their proof can be easily repaired by using curves of constant diameter instead of squares.

¹In that paper, they use the notation $k$ to refer to the diameter of a cluster instead of its radius.
For $0 < r < d < 2r$, we define a $(d,r)$-cluster to be a connected graph of diameter at most $d$ and of radius at most $r$. A $(d,r)$-clustering of a graph $G = (V,E)$ is a collection of subgraphs of $G$, each of which is a $(d,r)$-cluster, such that each vertex of $G$ belongs to exactly one of those clusters. We say that a $(d,r)$-clustering of $G$ is optimal if it contains the minimum number of clusters possible for any $(d,r)$-clustering of $G$. It is known that, for any positive $r$ and $d$, finding an optimal $(d,r)$-clustering for a given graph is $NP$-hard.

If $A$ is an algorithm that computes a $(d,r)$-clustering for a class $G$ of graphs, we say that $A$ is $C$-competitive for some $C > 1$ if there is a constant $K$ such that, for any graph $G \in G$, $A$ constructs a $(d,r)$-clustering with at most $C \cdot m + K$ $(d,r)$-clusters, where $m$ is the number of clusters in the optimal $(d,r)$-clustering of $G$.

10.1 The Hierarchy of Clustering Problems

Trivially, any $(d_1,r_1)$-clustering is a $(d_2,r_2)$-clustering, if $d_2 \leq d_1$ and $r_2 \leq r_1$.

Lemma 10.1.1

(a) If $r \leq d \leq 2r - 2$, then the optimal $(d,r-1)$-clustering algorithm is not competitive for the $(d,r)$-clustering problem on the class of all graphs.

(b) If $r < d \leq 2r$, then the optimal $(d-1,r)$-clustering algorithm is not competitive for the $(d,r)$-clustering problem on the class of all graphs.
Proof. We prove (a) and (b) simultaneously, since their proofs are almost identical. Suppose that, for some constants $C$ and $K$, the number of clusters in some $(d - 1, r)$-clustering or $(d, r - 1)$-clustering of any graph is at most $K$, plus $C$ times the number of clusters in the optimal $(d, r)$-clustering of that graph. We will reach a contradiction by giving a graph $G$ which is itself a single $(d, r)$-cluster, but which cannot be covered by $C + K (d - 1, r)$-clusters or $(d, r - 1)$-clusters.

We define the standard $(d, r)$-cluster, $G_{d,r}$, as follows. If $r = \lceil \frac{d}{2} \rceil$, then $G_{d,r}$ is a simple chain of length $d$. Otherwise, $G_{d,r}$ consists of a ring of $4r - 2d$ nodes, which we call the inner ring, together with a chain of length $d - r$ attached to each node in the inner ring. Figure 10.1 shows $G_{7,5}$, for example. We observe that $G_{2r,r}$ has 2 leaves, and that $G_{d,r}$ has $4r - 2d$ leaves if $d < 2r$. Let $L_{d,r}$ be the set of leaves of $G_{d,r}$.

Let $G = (G_{d,r})^m$, the $m$-fold strong product. Two nodes of $G$, which are $m$-tuples $u = (u_1, \ldots, u_m)$ and $v = (v_1, \ldots, v_m)$, are adjacent if and only if $u_i = v_i$ or $u_i$ is adjacent to $v_i$ for all $i$. Note that $G$ is a $(d, r)$-cluster. Let $L = (L_{d,r})^m \subseteq G$, which has cardinality $2^m$ if $d = 2r$, and $(4r - 2d)^m$ otherwise.

Note that no $(d, r-1)$-cluster or $(d-1, r)$-cluster of $G_{d,r}$ can contain all its leaves. Thus, if $d < 2r$, any $(d, r-1)$-cluster or $(d-1, r)$-cluster of $G$ can contain at most $(4r - 2d - 1)^m$
nodes of $L$. In the special case that $d = 2r$, a $(2r - 1, r)$-cluster of $G_{2r,r}$ can contain at most one member of $L$.

Finally, the number of $(d - 1, r)$-clusters or $(d, r - 1)$-clusters needed to cover $G$ is at least the ratio between the cardinality of $L$ and the maximum number of members of $L$ covered by a given $(d - 1, r)$-cluster or $(d, r - 1)$-cluster, respectively, which grows without bound as $m$ increases, and will thus eventually be larger than $C + K$. \hfill \Box

10.2 $k$-Clustering

There is more than one definition of $k$-clustering in the literature. In Amis et al. [3], and in Chapters 1 through 8 of this thesis, the term $k$-clustering refers to $(2k, k)$-clustering, i.e., where every cluster has radius at most $k$. In Fernandess and Malkhi [29], the term $k$-clustering refers to $(k, k)$-clustering, i.e., where every cluster has diameter at most $k$. However, close examination of the algorithms given by Fernandess and Malkhi reveals that they are actually solving the $(k, \lceil k/2 \rceil)$-clustering problem.

As a practical matter, it is desirable that a cluster contains an internal communication network, such as a spanning tree rooted at a clusterhead. In this case, the radius of the cluster is much more relevant than its diameter. We thus suggest that, for practical reasons, the most important case of the $(d, r)$-clustering problem is the case that $d = 2r$, which is the case addressed by the algorithms in this paper.
CHAPTER 11

CONCLUSION AND FUTURE RESEARCH

We present two self-stabilizing asynchronous distributed algorithms for construction of a BFS spanning tree, a maximal independent set, and a $k$-clustering for a given $k$, for any network with unique IDs. Algorithm FLOOD, given in Section 7, uses $O(k \log n)$ space per process and takes $O(k)$ rounds. In the worst case, FLOOD does not perform well, and in fact, constructs $k$-clusters with average size less than 2.

Our second algorithm, BFS-MIS-CLSTR, uses $O(\log n)$ space per process, takes $O(n)$ rounds, and computes $O(\frac{n}{k})$ clusters. In the special case that the network is a unit disk graph in the plane, BFS-MIS-CLSTR is $O(k)$-competitive. BFS-MIS-CLSTR also elects the process of smallest ID to be a leader, and constructs a BFS tree. It also constructs a maximal independent set.

We also give a lower bound tradeoff between the time complexity and competitiveness of any distributed algorithm for the $k$-clustering problem that uses only comparison to distinguish IDs. Any such algorithm that is $C$-competitive for any $C < \frac{n}{2}$ must take $\Omega(\text{diam})$ rounds in the worst case.

This research can be extended in various ways. We have constructed clusters using four schemes — FLOOD and three clustering schemes in BFS-MIS-CLSTR. We can study different networks (by varying both size and topology) to compare these clustering methods in various ways. Some interesting parameters to investigate would be the average height...
of the cluster trees, the average degree of the processes, and the average distance between two non-cluster processes.

The main cost of BFS-MIS-CLSTR in terms of time is due to the BFS module which includes a leader election algorithm; the BFS module takes $O(n)$ rounds. The leader election can be solved in less than $O(diam)$ rounds, but only in non-self-stabilizing environment. The challenging task is to answer the following question: Is it possible to design a self-stabilizing leader election (and hence, a self-stabilizing BFS algorithm) in $(diam)$ time, or something less than $O(n)$ time using $O(\log n)$ space?

We can also move beyond the comparison-based model of computation, permitting comparison of the individual bits of the IDs. Allowing these comparisons could break the lower bound barrier we proved for the comparison model. We are currently investigating that approach.

The clusters formed by both FLOOD and BFS-MIS-CLSTR can be used to design intra-cluster routing protocols. However, we need to select some gateway processes to design inter-cluster routing schemes.
BIBLIOGRAPHY


VITA
Graduate College
University of Nevada, Las Vegas

Priyanka Vemula

Home Address:
4247 Grove Circle, Apt #2
Las Vegas NV 89119

Degrees:
Bachelor of Science, Computer Science, 2006
Nagarjuna University, India

Thesis Title: Self-Stabilizing K-Clustering in Mobile Ad Hoc Networks

Thesis Examination Committee:
Chairperson, Dr. Ajoy K. Datta, Ph.D.
Committee Member, Dr. Yoohwan Kim, Ph.D.
Committee Member, Dr. Lawrence L. Larmore, Ph.D.
Graduate Faculty Representative, Dr. Venkatesan Muthukumar, Ph.D.