PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
Ministry of Higher Education and Scientific Research
Batna 2 - University
Faculty of Mathematics and Computer Science
Computer Science department

Doctoral Thesis

To obtain the degree of Doctor of Science
Sector: Computer science
Specialty: Networks and Distributed Systems

Self-stabilization in dynamic distributed systems

Presented by
Saadi Leila

President	Faiza TITOUNA	Professor	U-Batna 2
Supervisor	Hamouma MOUMEN	Professor	U-Batna 2
Examiner	Sonia Sabrina BENDIB	MCA	U-Batna 2
Examiner	Aldjia BOUCETTA	MCA	U-Batna 1
Examiner	Abderrahmane BAADACHE	Professor	U-Alger 1
Examiner	Leila BOUSSAAD	MCA	U-Batna 1

○ ○ ○

”Strong people are the ones who’ve come through the toughest times.” .....

○ ○ ○

Dedicate to

○ ○ ○

To the memory of my mother, only the person I need in my life
To my father, only the person I respect and I am indeted
To my husband
To my daughters Soujoud et Safwa my hope in this life
To all my sisters and brothers and their families
To my father-in-law, only my best friend.

○ ○ ○

Acknowledgments

My gratitude goes to everyone whose assistance was vital in accomplishing this thesis.

First and foremost, my sincerest thanks go to my supervisor, Hamouma Moumen, for his constant unwavering guidance, diligent advice, and infinite support. Thank you Chief.

I would like to thank Dr.Baderddine Benreguia for being a source of knowledge on the main principles of this thesis.

Special thanks go to Pr. Faiza Titouna, Dr. Sonia Sabrina Bendib, Dr. Aldjia Boucetta, Dr. Abderrahmane BAADACHE and Dr. Leila Boussaad who agreed to be part of my final defense committee. I would also like to extend my thanks to all colleagues and friends in the Computer Science Department who contributed one way or another to this thesis.

Leila

Abstract

Dynamic distributed systems are used widely in everyday life by dealing with exploitation and treatment on information, documents, services, medias, and all entertainment means. Many companies creating software and systems compete to offer users powerful and tolerant services.

This dynamic DS are obliged to response the user demands in time and with trustworthy information, they offer all levels of security; authenticity of all services and personnel information. This kind of systems appears in our life in all areas using different means of sensors that collect all type of information and requests where the challenge is to ensure a permanent service availability since failures occurs.

Many algorithms are used here to fault tolerant and exceed those situations, among the self-stabilizing algorithms that take the system to a legitimate state even the presence of failures and errors.

This thesis deals with the problem of finding dominating set using self-stabilizing paradigm in dynamic distributed systems. Usually, members of a dominating set are selected to be as cluster heads in Wireless Sensor Networks (WSN) in order to ensure a permanent service availability. Since failures occurs frequently inside WSN due to limited battery energy, self-stabilizing algorithm allows recomputing the dominating set, and hence the network returns to its ordinary running.

Existing works have introduced many variants of self-stabilizing algorithms that compute minimal dominating set S where each node out of S has neighbours in S more than it has out S. In this thesis, we introduce a generalized self-stabilizing algorithm called minimal (α, β)-dominating set. An α-dominating set is a subset of nodes S such that for any node v out of S, the rate of neighbours of v inside S must be greater than α, where $0 < \alpha \leq 1$. In the same way, an (α, β)-dominating set is a subset of nodes S such that: S is α-dominating set and for each node v in S, the rate of neighbours of v inside S is greater than β, where $0 \leq \beta \leq 1$. Mathematical proofs and simulation tests show the correctness and the efficiency of the proposed algorithm.

Through our proposed variant (α, β)-domination, we prove rigorously the conjecture of Carrier et. al. (Self-stabilizing (f, g)-alliances with safe convergence) who have proposed a self-stabilizing algorithm for a domination variant called (f, g)-alliance set only when $f \geq g$. We prove the correctness of the case $f < g$.

Key words: *self-stabilizing algorithm; minimal dominating set; α-domination; dynamic distributed systems.*

Contents

List of Figures

List of Tables

General Introduction

General Introduction

Distributed computing occurs when one has to solve a problem in terms of physically distinct entities (usually called nodes, processors, processes, agents, sensors, etc.) such that each entity has only a partial knowledge of the many parameters involved in the problem.

Understanding and designing distributed applications is not an easy task [AW04], [CaR11]. This is because, due to the very nature of distributed computing, no node can capture instantaneously the global state of the application it is part of. This comes from the fact that, as nodes are geographically localized at distinct places; distributed applications have to cope with the uncertainty state created by failures.

Self-stabilization is a fault tolerance approach for distributed systems that has been introduced for the first time by Dijkstra [Dij74]. A self-stabilizing distributed system is able to achieve a global correct configuration (without any external intervention), in a finite time, starting from an initial illegitimate configuration. Various self-stabilizing distributed algorithms have been proposed in the literature using graph theory such as leader election, nodes coloring, domination problems, independent set identification, spanning tree construction. The reader can refer to the survey [GK10] for more details of self-stabilizing algorithms.

Domination has been extensively studied in literature [HHS98] and adopted in many real-life applications. It has been utilized for address routing, power management and clustering issues in ad-hoc networks [LLY09, AWF03, BDTC05]. Recently, particular parameters of domination have been used to influence (and change) the opinion of the users in the social networks [FEK09, AKL18]. A dominating set is a subset S of the graph nodes where every node is either in S or is a neighbor of at least one node of S. The dominating set S is minimal if there is no proper subset in S that could be a dominating set. Before 2003, no self-stabilizing algorithms have been introduced to solve (minimal) dominating sets problem. At that time, algorithms are in general based on greedy, exact and heuristic methods. The first *self-stabilizing* algorithm for (minimal) dominating set was proposed by Hedetniemi *et al.* [HHJS03]. After that, many variants of self-stabilizing algorithms have been proposed imposing additional parameters of domination like total domination [BYK14, GHJS03b], efficient domination [Tur13, HHJ$^+$12], connected dominating set [BBP13, DWS16], influence domination [WWTZ13, DWS14b], distance-k domination [DDL18]. Each parameter

has its benefits according to the used application. For example, connected dominating sets are generally used as backbone (infrastructure) in ad-hoc and sensor networks.

The α-domination concept (without using self-stabilization concept) has been studied for the first time by [DHLM00]. Other results on α-domination are given in [DRV04]. Let $G = (V, E)$ be a connected graph where V is the set of nodes and E is the set of edges. We say that $S \subseteq V$ is α-dominating if for all $v \in V - S$, $\frac{|N(v) \cap S|}{|N(v)|} \geq \alpha$, where $0 < \alpha \leq 1$ and $N(v)$ is the set of v neighbors i.e. $N(v) = \{u | vu \in E\}$. Besides the particular cases of α-domination introduced in the literature, we present a new parametric called (α, β)-dominating set. An (α, β)-dominating set is a subset of nodes S such that: S is α-dominating set and for each node v in S, the rate of neighbours of v inside S is greater than β, where $0 \leq \beta \leq 1$.

Contribution

Previous self-stabilizing algorithms have used only $\alpha = 1/2$, where every node out of S must be adjacent to at least 50% of nodes inside the dominating set. The threshold 50% can be a limited choice, whereas other values like 20% ($\alpha = 0.2$) or 70% ($\alpha = 0.7$) may be more useful. Therefore, using general parameters of α-domination and β-domination will be more interesting for many practical cases.

Carrier $et.\ al.$ [CDD$^+$15] have introduced a variant similar to α and β domination, called (f, g)-alliance set. However, authors have proved the self-stabilizing algorithm only when $f \geq g$. Hence, the case $f < g$ still as a conjecture. Through our proposed variant (α, β)-domination, we prove rigorously their conjecture.

The thesis has also the following threefold contributions.

(i) We propose a self-stabilizing algorithm to find a *minimal α-dominating set* called α-MDS.

(ii) a new parameter of domination (α, β)-$domination$ is introduced.

(iii) Finally, a self-stabilizing algorithm is presented for computing *minimal $(\alpha - \beta)$-dominating set*.

The later proposed algorithm proves the conjecture of Carrier $et.\ al.$ [CDD$^+$15] that there is a self-stabilizing algorithm for (f, g)-alliance problem when $f < g$.

Thesis organization

This thesis is organized as follows: Chapter 1 provides a presentation of distributed systems and fault tolerance paradigm. First, we have summarized the notions of distributed systems, their definition, objectives, types; then we moved on to existing communication models. Synchronization is also an important concept in DS, which is why we have focused on synchronous and asynchronous systems.

studying distributed systems requires the study and knowledge of the fault tolerance paradigm which is described in the second part of the chapter.

In the second chapter we introduced and described the self-stabilizing algorithms used to overcome faults in a distributed system. we have seen how to designate a self-stabilizing algorithm and the main elements influencing their stabilization such as daemons, complexity measures and transformers. We ended the chapter with an important state of the art of some self-stabilizing algorithms in different graphs.

Since this thesis is interested in the minimal dominating set paradigm and how to apply a self-stabilizing algorithm to find it, the third chapter introduces the concept of minimal dominating set in general and the varieties of algorithms used to find it.

The fourth chapter details the use of the self-stabilizing paradigm to find the dominating set. After presenting some existing self-stabilizing algorithms (like the first self-stabilizing algorithm for minimal dominating set proposed by Hedetniemi et al. and other many variants of self-stabilizing algorithms proposed imposing additional parameters like total domination, efficient domination, ...), the chapter must discuses the main contribution : Self-stabilizing algorithm for minimal (α, β)-dominating set.

Finally, we conclude the work with an important conclusion and future perspectives.

Chapter 1

Distributed systems and faults tolerance

CHAPTER 1

DISTRIBUTED SYSTEMS AND FAULT TOLERANCE

Contents

1.1 Introduction

The study of this thesis takes place around fault tolerance in distributed systems exactly the proposal of new algorithms allowing systems to self-stabilize, which gave the need to make a study on distributed systems and fault tolerance.

This Chapter is divided en tow parts by presenting important knowledge on distributed systems and fault tolerance. Firstly,we quote the fundamentals of a distributed system; its objects, architectures, models and a brief presentation of an asynchronous distributed system. Generally, since the important goal in distributed systems design is to construct the system in such a way that it can automatically recover from partial failures without seriously affecting the overall performance, we will discuss in the second part the importance of fault tolerance. an important state of the art will be presented in this chapter to highlight the advantages and disadvantages of fault tolerance algorithms in distributed systems and networks.

1.2 Distributed systems

1.2.1 Definition

Computer architectures have evolved significantly since their appearance side by side of operating systems and different software, which has given rise to distributed systems defined as an interconnected, multiple processors. Various definitions of distributed systems have been given in the literature, none of them satisfactory, and none of them in agreement with any of the others.

[TS06] define the distributed system as a collection of independent computers that appears to its users as a single coherent system. Otherwise; this definition refers to two important characteristics of distributed systems. The first is that a distributed system is a set of computer elements that can each behave independently of each other called nodes. A second element is that users (whether people or applications) think they are dealing with a single system. [TS16] A general definition has been given by [KS08] that is a collection of independent entities that cooperate to solve a complex problem that cannot be individually solved. This definition refers all system where there is a communication among mobile intelligent agents in the real world like flock of birds, ecosystems of microorganisms. the distributed system became a reality in our life due to the emergence of the internet and the huge data flowing through the web.

A distributed system is composed of a set of independent computers in memory, processor, operating system and software/hardware resources; interconnected via a specific com-

munication network. each computer acts as an independent entity locally; globally all units can participate in distributed tasks/calculations like showing in Figure 1.1.

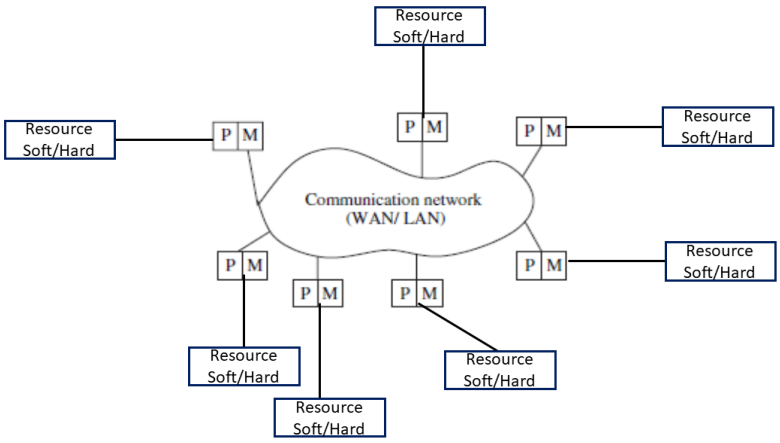

Figure 1.1: A distributed system connects processors and systems by a communication network. [KS08]

In order to support the heterogeneity of computers in the network while providing a single system view, distributed systems are often organized by means of a software layer, i.e. logically placed between a level layer top layer composed of users and applications, and an underlying layer, called middleware as shown in Figure 1.2.

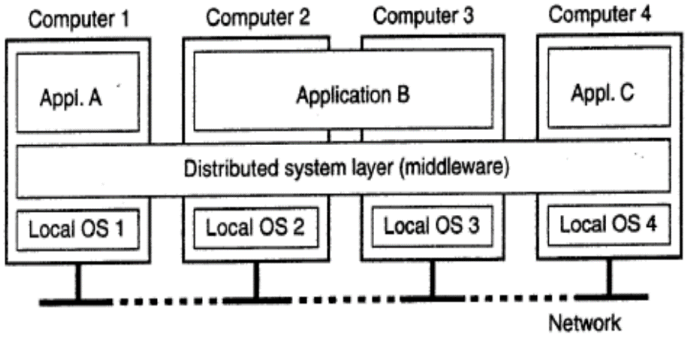

Figure 1.2: A distributed system organized as middleware. [TS06]

1.2.2 Distributed systems goals

- *Making resource accessible:* the most important reason to highlight a distributed system is to share different remote resources placed on different sites either hardware (printers, disks, computers, ...) or software (data, files, web pages, ...). The desired objective is to minimize the cost of using resources economically.

- *Distribution and transparency:* another important goal of a distributed system is to hide from clients everything about the operation of the system and how resources are physically shared among users.

  transparency type

  [TS06] resume the types of transparency in the table:

Access	hide differences in data representation and how a resource is accessed
Location	hide where a resource is located
Migration	hide that the resource may move to another location
Relocation	hide that a resource may be moved to another location while in use
Replication	hide that the resource is replicated
Concurrency	hide that the resource may be shared by several competitive users
Failure	hide the failure and recovery of resource

Table 1.1: Different types of transparency in a distributed system [ISO95].

  Degree of transparency

  In many cases, increasing the transparency of distribution costs the loss of performance in distributed systems. We have a compromise between having transparency and keeping the performance in such a way many Internet applications repeatedly try to contact a server before finally giving up. Consequently, attempting to mask a transient server failure before trying another one may slow down the system as a whole. [KS08] puts the point in such a case showing it may have been better to give up earlier, or at least let the user cancel the attempts to make contact. In other way, aiming for distribution transparency may be a nice goal when designing and implementing distributed systems, but that it should be considered together with other issues such as performance and comprehensibility. The price for not being able to achieve full transparency may be surprisingly high.

- *Openness:* an open distributed system is a system that offers services according to standard rules that describe the syntax and semantics of those services. [KS08] [Cro96] describe an open distributed system with a set of components that may be obtained

from a number of different sources, which together work as a single distributed system. In 1988, the International Standard Organization (ISO) began work on preparing standards for open distributed system (ODP) to define the interfaces and protocols to be used in the various components in an ODP. Generally, an open distributed system should also be extensible; that means we can easily add components or parts that run in different operating systems or replace an entire one.

- *Scalability:* to know if a distributed system is scalable, we have to measure the evolution of its size on the user side, resources, geographical mobility in the sense of distance, and administrative evolution so that the system can be managed even over time. To realize this idea, the designers face some problems dealing with a centralized server, huge amount of data and information and different types of networks from a localization point of view.

- *Pitfalls:* distributed systems differ from traditional software because components are dispersed across a network. Not taking this dispersion into account during design time is what makes so many systems needlessly complex and results in mistakes that need to be patched later on. This mistakes are formulated as the following false assumptions that everyone makes when developing a distributed application for the first time: [KS08]

  1. The network is reliable.
  2. The network is secure.
  3. The network is homogeneous.
  4. The topology does not change.
  5. Latency is zero.
  6. Bandwidth is infinite.
  7. Transport cost is zero.
  8. There is one administrator.

1.2.3 Types of distributed systems

Distributed systems are growing quickly and touche an important number of fields, in fact, we can distinguish between different types of them:

- *Distributed computing systems:* this class of distributed systems is used to perform high performance computing tasks; we can distinguish tow groups:

1. *Cluster computing systems:* this subgroup is used for the parallel programming in which a single program is run in parallel environment on multiple machines. the computers in a cluster are largely the same, they all have the same operating system, and are all connected through the same network.

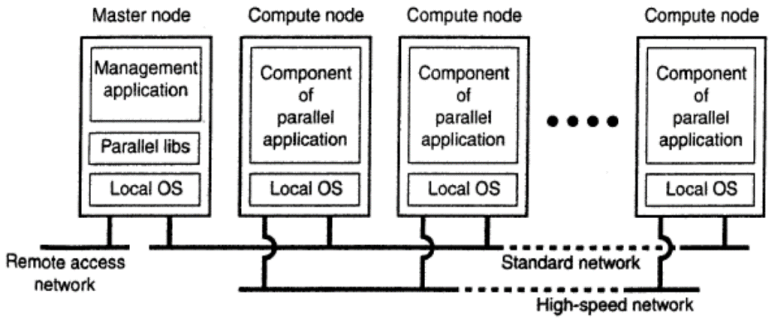

Figure 1.3: An example of a cluster computing system. [KS08]

2. *Grid Computing systems:* the key issue of a grid computing that we can use a large number of resources from different organizations and locations with different system to collaborate in a task belonging to q group of people using virtual organization.

3. *Cloud Computing system:* following Vaquero et al. [VMCL09], cloud computing is characterized by an easily usable and accessible pool of virtualized resources. Which and how resources are used can be configured dynamically, providing the basis for scalability: if more work needs to be done, a customer can simply acquire more resources. The link to utility computing is formed by the fact that cloud computing is generally based on a pay-per-use model in which guarantees are offered by means of customized service level agreements (SLAs).

   Cloud computing is organized to four layers as:

   Hardware processors, routers, but also power and cooling systems.

   Infrastructure It deploys virtualization techniques to provide customers an infrastructure consisting of virtual storage and computing resources.

   Platform Also like operating systems, the platform layer provides higher-level abstractions for storage and such.

   Application Actual applications run in this layer and are offered to users for further customization.

- *Distributed information systems:* the second class of distributed systems is found in organizations that were confronted with a multiple of networked applications, but for which interoperability turned out to be a painful experience. Many of the existing middleware solutions are the result of working with an infrastructure in which it was easier to integrate applications into an enterprise-wide information system. Two forms of this class are listed:

  1. *Transaction processing systems*
  2. *Enterprise application integration*

- *Distributed pervasive systems:* in this class, we refer to a distributed pervasive systems, they are often characterized by being small, battery-powered, mobile, and having only a wireless connection, although not all these characteristics apply to all devices. Moreover, these characteristics need not necessarily be interpreted as restrictive, as is illustrated by the possibilities of modem smart phones. [RMM05]. This systems are implemented in different aspects of life: in homes systems, buses, buildings, farms, electronic health care systems, sensor networks... to give in the end the notion of internet of things.

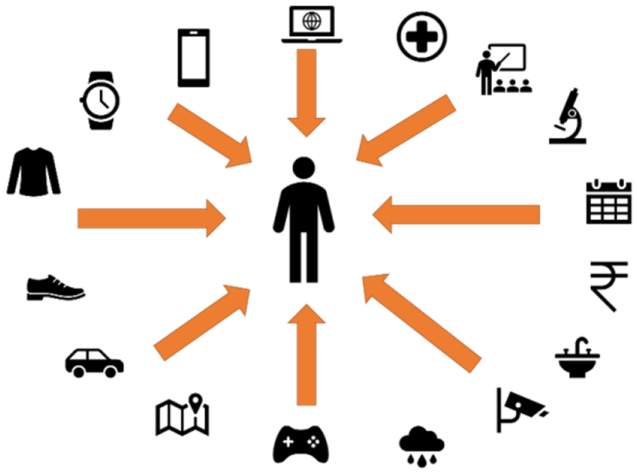

Figure 1.4: Distributed pervasive systems.

1.3 Models of communication in distributed systems

A distributed system involves a number of nodes communicating with one another. In order to develop a distributed algorithm, details of inter-nodes communication can be quite important. In general, there are many parameters of variability in distributed systems like

the model of communication. Models are simple abstractions that help understand the way nodes communicate.

1.3.1 Message passing communication

This model of communication correspond to systems where processes communicate by sending messages through a network. In synchronous message-passing, every process sends out messages at time t that are delivered at time t + 1, at which point more messages are sent out that are delivered at time t + 2; the whole system runs in lockstep, marching forward in perfect synchrony. [Asp22]

In asynchronous systems, messages are only delivered after unknown delay. we can found also semi-synchronous models.

1.3.2 Shared memory communication

Every two neighbors share a common memory. Therefore, using the common memory, a node will be able to read its local state and states of its neighbor nodes. Two sub-models can be distinguished in this case:

1.3.2.1 State model (or composite model):

In this model, a node executes the following three operations as an atomic step.

(1) reading states (registers) of all its neighbors,

(2) making internal computations and then

(3) updating its own state (register).

This model has been introduced by Dijkstra in order to be used for the concept of self-stabilization [Dij74].

1.3.2.2 read-write atomicity:

This model makes separation between operations. We assume that each operation is an atomic step i.e. reading is atomic step and writing is another atomic step. [DIM90].

1.4 Asynchronous distributed systems

The distributed systems are also classified on synchronous/asynchronous systems, the synchronisation can be processor synchrony/asynchrony, synchronous/asynchronous communication primitives and synchronous/asynchronous executions.

1.4.1 Synchronous distributed system

A synchronous distributed system comes with strong guarantees about properties and nature of the system. Because the system makes strong guarantees, it usually comes with strong assumptions and certain constraints. Synchronous nature by itself is multi-faceted:

- *Upper Bound on Message Delivery* There is a known upper bound on message transmission delay from one process to another process OR one machine/node to another machine/node. Messages are not expected to be delayed for arbitrary time periods between any given set of participating nodes.
- *Ordered Message Delivery* The communication ways between two machines are expected to deliver the messages in FIFO order. It means that the network will never deliver messages in an arbitrary or random order that can't be predicted by the participating processes.
- *Notion of Globally Synchronized Clocks* Each node has a local clock, and the clocks of all nodes are always in sync with each other. This makes it trivial to establish a global real time ordering of events not only on a particular node, but also across the nodes.
- *Lock Step Based Execution* The participating processes execute in lock-step. An example will make it more clear. Consider a distributed system having a coordinator node that dispatches a message to other follower nodes, and each follower node is expected to process the message once the message is received. It cannot be the case that different follower nodes process the input message independently at different times and thus generate output state at different times. This is why we say processes execute in lock step synchrony a la lock step marching.

The problem with synchronous distributed systems is that they are not really practical. Any software system based on strong assumptions tends to be less robust in real world settings and begins to break in practical/common workloads. For example, relying on the network that it is definitely going to deliver the message in a fixed amount of time is not really a practical assumption. In real world, software system is subjected to multiple kinds of failure. It is practically difficult to build a completely synchronous system, and have the messages delivered within a bounded time. [KS08]

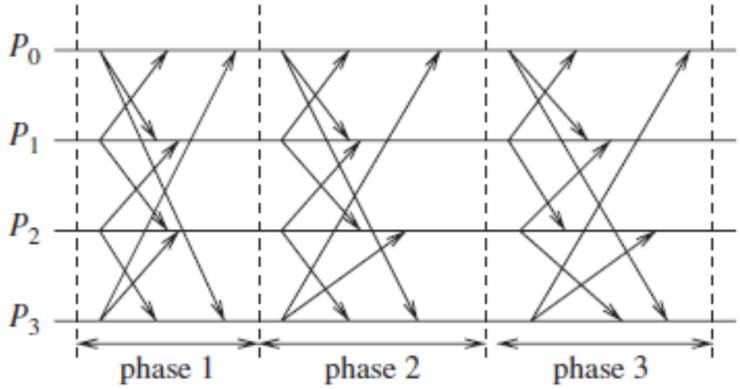

Figure 1.5: An example of a synchronous execution in a message-passing system. All the messages sent in a round are received within that same round. [KS08]

1.4.2 Asynchronous distributed system

The most important thing about an asynchronous distributed system is that it is more suitable for real world scenarios since it does not make any strong assumptions about time and order of events in a distributed system.

- *Clock may not be accurate, clocks can be out of synchronisation* Clocks of different nodes in a distributed system can drift apart. Thus it is not at all trivial to reason about the global real time ordering of events across all the machines in the cluster. Machine local timestamps will no longer help here since the clocks are no longer assumed to be always in synchronisation.
- *Messages can be delayed for arbitrary period of times* Unlike synchronous distributed system, there is no known upper limit on message transmission delay between nodes.

Asynchronous distributed system is tough to understand since it is not based on strong assumptions and does not really impose any constraints on time and ordering of events. It is also tough to design and implement such a system since the algorithms should tolerate different kinds of failures. An obvious example of such an asynchronous system is the Internet.

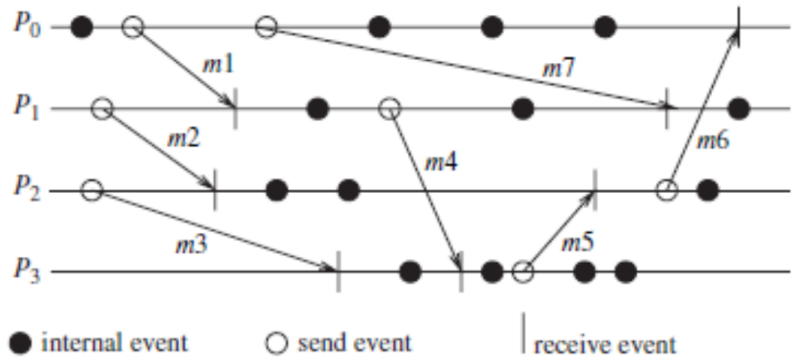

Figure 1.6: An example of an asynchronous execution in a message-passing system. A timing diagram is used to illustrate the execution. [KS08]

1.5 Fault tolerance

More close to everyday life, are the telecommunications switching systems and the bank transaction systems, this systems are more close to be attacked or get failures in reliability and availability. To achieve the needed reliability and availability, we need fault-tolerant computers which they have the ability to tolerate faults by detecting failures, and isolate defect modules so that the rest of the system can operate correctly.

1.5.1 Definition

[HW92] describes the situation of faults like that a system is said to have failure if the service it delivers to users deviates from compliance with the system specification for a specified period of time.

A failure is defined as a deviation of a service delivered by a system from the essential specification of this system to eliminate errors or faults.

Fault tolerance refers to the ability of a system (computer, network, cloud cluster, etc.) to continue operating without interruption when one or more of its components fail. To say that a system tolerates faults, we must guarantee the concepts:

- **availability:** refers to the probability that the system is operating correctly at any given moment and is available to perform its functions on behalf of its users.
- **Reliability:** presents the response of a system for a long time to the maximum and it performs its tasks continuously without failure and errors.

- **Safety:** refers to the situation that when a system temporarily fails to operate correctly, nothing catastrophic happens.
- **maintainability:** means that how a failed system can be repaired; that shows also a high degree of availability.

A system is said to fail when it cannot meet its promises. In particular, if a distributed system is designed to provide its users with a number of services, the system has failed when one or more of those services cannot be (completely) provided. [TS06] We can define the cause of an error as a fault, it's important to know the origin of an error to create a fault tolerance system. faults are classified as transient, intermittent, or permanent.

1.5.2 Failure models

[TS06] resume the failure models in the table 1.2:

Type of failure	**Description**
Crash failure	A server halts, but is working correctly until it halts
Omission failure Receive omission Send omission	A server fails to respond to incoming requests A server fails to receive to incoming messages A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure Value failure State transition failure	A server's response is incorrect The value of the response is wrong The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary time

Table 1.2: Different types of failure.

1.5.3 Fault tolerance approaches

1.5.3.1 Fault taxonomy in distributed systems

In [Tix10], Tixeuil describes faults in distributed systems using two criteria: time and nature. Considering the time occurrence of faults, three types are distinguished:

1. *transient faults:* faults that are arbitrary in nature can strike the system, but there is a point in the execution beyond which these faults no longer occur;
2. *permanent faults:* faults that are arbitrary in nature can strike the system, but there is a point in the execution beyond which these faults always occur;
3. *intermittent faults:* faults that are arbitrary in nature can strike the system, at any moment in the execution.

A second criterion is the nature of the faults. We can distinguish the following faults depending on whether they involve the state or the code of the element:

1. *State related faults:* changes in an element's variables may be caused by disturbances in the environment (electromagnetic waves, for example), attacks (buffer overflow, for example) or simply faults on the part of the equipment used.
2. *code-related faults:* an arbitrary change in an element's code is most often the result of an attack (the replacement, for example, of an element by a malicious opponent), but certain, less serious types correspond to bugs or a difficulty in handling the load.
    - Crashes
    - Omissions
    - Duplications
    - Desequencing
    - Byzantine faults

1.5.3.2 Fault-tolerant algorithm categories

Two major categories for fault-tolerant algorithms can be distinguished:

1. *Robust Algorithms:* these use redundancy on several levels of information, of communications, or of the system's nodes, in order to overlap to the extent that the rest of the code can safely be executed.
2. *Self-stabilizing algorithms:* these rely on the hypothesis that the faults are transient (in other words, limited in time), but do not set constraints regarding the extent of the faults (which may involve all of the system's elements). Typically, self-stabilizing algorithms are non-masking, because between the moment when the faults cease and the moment when the system has stabilized to an appropriate behavior, the execution may turn out to be somewhat erratic. [Tix10]

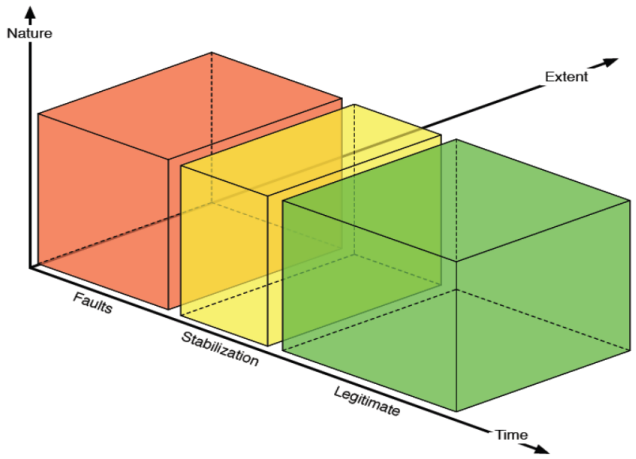

Figure 1.7: Self-stabilization algorithms.

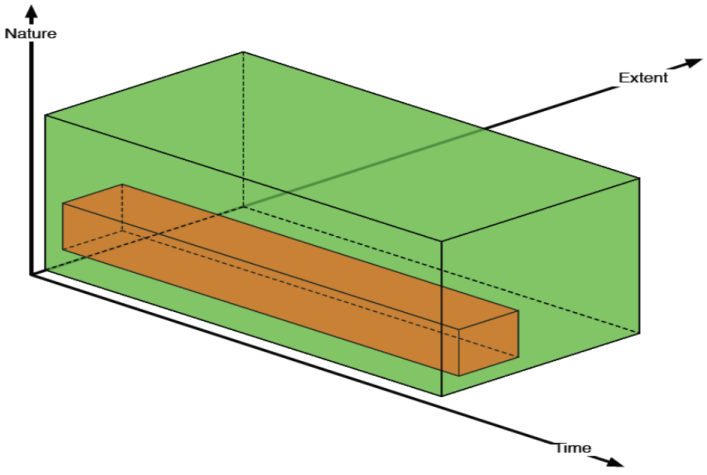

Figure 1.8: Robust algorithms.

1.5.4 Fault tolerance mechanism in distributed system

1.5.4.1 Replication Based Fault Tolerance Technique

The replication based fault tolerance technique is one of the most popular method. This technique actually replicate the data on different other system. In the replication techniques, a request can be sent to one replica system in the midst of the other replica system. In this way if a particular or more than one node fails to function, it will not cause the whole system to stop functioning as in the Figure 1.7.

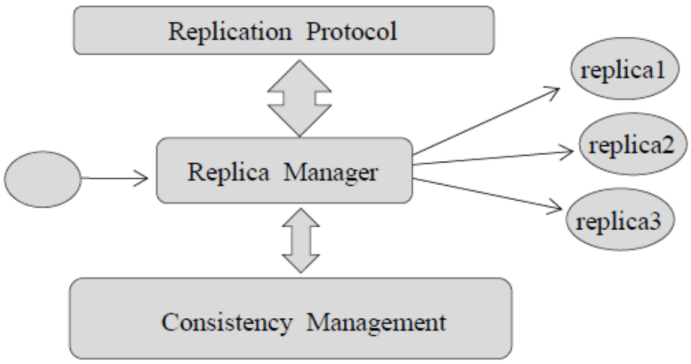

Figure 1.9: Replication based technique in distributed system. [SA15]

Consistency: This is a vital issue in replication technique. Several copies of the same entity create problem of consistency because of update that can be done by any of the user. The consistency of data is ensured by some criteria such as linearizability, sequential consistency and casual consistency, etc. sequential and linearizability consistency ensures strong consistency unlike casual consistency which defines a weak consistency criterion.

For example a primary backup replication technique guarantee consistency by linerarizability, likewise active replication technique.[SA15]

*Degree or Number of Replica:*the replication techniques utilises some protocols in replication of data or an object, such protocol are: Primary backup replication, voting and primary-per partition replication.

1.5.4.2 Process level redundancy technique

This fault tolerance technique is often used for the transient (described in 1.4.3) faults that disappears without anything been done to remedy the situation; the problem with transient

faults is that they are hard to handle and diagnose but they are less severe in nature. In handling of transient fault, software based fault tolerance technique such as Process-Level Redundancy (PLR) is used because hardware based fault tolerance technique is more expensive to deploy.

1.5.4.3 Fusion based technique

Replication is the most widely used method or technique in fault tolerance. The main disadvantage is that there are a lot of backups involved. Since backups increase with failures and management costs are very high, fusion-based technologies solve this problem. Convergence-based technology offers an alternative because it requires fewer backup machines than replication-based technique. [SA15]

1.5.4.4 Checking Point/RollBack Technique

Checkpointing is a technique for inserting fault tolerance into computing systems. It basically consists of taking a snapshot of the current application state, storing it on some memory area and later on using it for restarting the execution from that particular point in case of failure. It is a fault tolerant technique in which normal processing of a process is interrupted specifically to preserve the status information necessary and then to allow resumption of processing at a later time. Computation may be restarted from the current checkpoint instead of repeating it from the beginning if a failure occurs. Checkpoint based rollback recovery is being used as a technique in various areas like scientific computing, mobile computing, distributed database, telecommunication and critical applications in distributed and mobile ad hoc networks. [PT14] Categories of checking point/rollback technique are:

- uncoordinated checkpointing
- coordinated checkpointing (blocking and non-blocking)
- communication-induced check pointing

We resume in the table 1.3 the differences between the techniques of fault tolerance in distributed system:

1.6 Conclusion

To clarify the field of study of this thesis, reference has been made to a first chapter comprising a brief state of the art on distributed systems. In addition, we have shown through what

Major Factors	**Replication Technique**	**Checking Point/Roll Back Technique**	**Fusion Based Technique**	**Process Level Redundancy Technique**
Working	Redirected to replica	State saved on stables to rage used for recovery	Back up machine	A set of redundant process
Consistency	Some criterion; linearizability.	Avoiding orphan messages	Among backup machines	Not a major issues
Multiple Faults Handling	Depend upon number of replica.	Depend upon Check pointing scheduling	Depends upon number of back machine	Depends upon set of redundant process
Performance	Decreases as number of replica increases.	Decrease with frequency and size of checkpoint	Decrease in case of faults as recovery cost is high	Decrease as faults appears disappear
N-Faults	N replicas ensure n-1 faults	Uncoordinated Pessimistic and N level disk less used for N-1 Faults	In order handle Extra N faults N backups machine are required	Scaling the number of process and Majority voting
Multiple Failure Detector	Reliable, Accurate, Adaptive	Reliable, Accurate, Adaptive	Reliable, Accurate, Adaptive	Reliable, Accurate, Adaptive

Table 1.3: the differences between the techniques of fault tolerance in distributed system. [SA15]

we read as a document what a distributed system is, its objectives, its types and the means of communication between its entities.

And to clearly specify what type of distributed system is in this study, we spent a moment on the synchronous and asynchronous distributed systems. Finally, we have oriented our state-of-the-art brief towards fault tolerance in distributed systems by emphasizing definitions, approaches and algorithms.

Among the algorithms cited for tolerating faults in a distributed system is self-stabilizing which gives the subject studied in the next chapter.

Chapter 2

Self-stabilizing in distributed systems

CHAPTER 2

SELF-STABILIZING IN DISTRIBUTED SYSTEMS

Contents

2.1 Introduction

In the previous chapter, we discussed the algorithms used to tolerate faults in a system which are robust algorithms and self-stabilizing algorithms. these allow a system to move from one state to another in order to stabilize and continue its operation in the presence of failures. Self-stabilizing methods prove their effectiveness in overcoming transient type faults. To clarify the concept of self-stabilizing algorithms, this chapter focuses on their general and formal definition, how to designate a self-stabilizing algorithm, complexity measures and examples of their use with a state of the art of works known in the field.

2.2 Definition

The self-stabilizing paradigm was introduced by Edsger W. Dijkstra in 1973. A self-stabilizing system is a system that can automatically recover following the occurrence of (transient) faults [Dol00]. The idea is to design systems that can be started in an arbitrary state and still converge to a desired behavior. We call the system "self-stabilizing" if and only if, regardless of the initial state and regardless of the privilege selected each time for the next move, at least one privilege will always be present and the system is guaranteed to find itself in a legitimate state after a finite number of moves. [Dij74]

This makes self-stabilization an elegant approach for transient fault-tolerance [Dol00]. Figure 2.1 illustrates the behavior of self-stabilizing system. Note that self-stabilizing system may not reach a legitimate configuration (or desired configuration) if faults occur frequently during the convergence. For this reason, most publications assume that all faults are transient, i.e. no further faults occur during the stabilization of the system.

Most of the studies of self-stabilizing have focused on theoretical level using simple undirected graphs. Only few of works have used experimental tests on random graphs known as Erdos-Renyi graphs [NGHK15]. In fact, it has been proved that most of real networks cannot be represented by the model Erdos-Renyi. Barabasi and Albert have proposed a model known as scale-free network or Barabasi-Albert model [BA99].

This concept did not gain any attention in the beginning until 1984, when Lamport referred to Dijkstra's work as an important approach for fault-tolerance. he Wrote in his paper [Lam84]: I regard this as Dijkstra's most brilliant work–at least, his most brilliant published paper. It's almost completely unknown. I regard it to be a milestone in work on fault tolerance. The terms "fault tolerance" and "reliability" never appear in this paper. In fact, one reason why it's not better known might be Dijkstra's approach, illustrated in [Dij74]. He said: I regard self-stabilization to be a very important concept in fault tolerance, and to be a very fertile field for research.

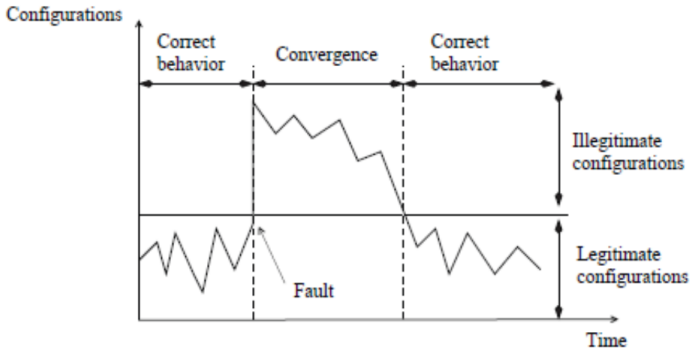

Figure 2.1: Self-stabilization system's behavior. [Neg15]

2.2.1 Formal definition

Let A be a distributed algorithm, G be a graph representing some topology, and D be a daemon. Let SP be a specification, i.e., a predicate over executions. We recall that SP is a formal definition of the problem to be solved, in terms of liveness and safety properties.

Definition 2.1: A is *self-stabilizing for SP* in G under D if there exists a non-empty subset of configurations $L \subseteq C$, called the legitimate configurations (conversely, $C\ L$ is called the set of illegitimate configurations) [ADDP19], such as defined by [AG93]:

- *Closure*: This property will be preserved once the system reaches a legal configuration; i.e. the set of legitimate configurations is closed. L is *closed* (by A in G under D), i.e., $\forall \gamma \in L$ and $\forall \gamma' \in C$, if $\gamma \mapsto \gamma'$, then $\gamma' \in L$.
- *Convergence*: The system always reaches a legitimate configuration after a finite time if no further fault occurs during the stabilization. A *converges under* D to L in G, i.e., $\forall e \in \varepsilon$, $\exists \gamma \in e$ such as $\gamma \in L$.
- *Correctness*: Every final configuration is legitimate, i.e. the algorithm actually computes for which it was originally developed. Under D, SP is satisfied from L, i.e., $\forall e \in \varepsilon(L)$, $SP(e)$ holds, where $\varepsilon(L)$ in the subset of executions of ε that starts from a configuration of L.

Self-stabilization and silence:

Many self-stabilizing algorithms for building distributed structures, such as spanning trees, actually implement an additional property called silence [DGS99]. This term was introduced into the register model, but their definitions apply independently of each model. A

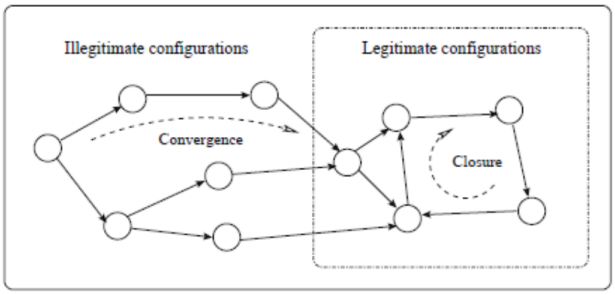

Figure 2.2: Self-stabilization properties, closure and convergence. [Neg15]

self-stabilizing algorithm is silent if it converges in a finite time to a configuration in which the values of the communication variables used by the algorithm remain fixed. Communication variables are buffer variables whose values are transmitted and/or retrieved from neighbors using communication primitives (usually send/receive or read/write operations). Silence is a desirable quality. As described in [DGS99], the silent property usually means simpler algorithm design. Furthermore, the silent algorithm can use less communication operations and communication bandwidth.

2.2.2 Self-stabilization advantages and disadvantages

Self-stabilization presents many advantages:

- *Self-recovering:* The system always reverts to correct behavior without external intervention or global initialization. Therefore, self-stabilization is very useful for scale-free systems where manual intervention is not possible.
- *No initialization:* The system always converges indeed if it starts from illegitimate configuration. Therefore, the self-stabilizing algorithms don't need any correct initialization.
- *Dynamic topology adaptation:* If an algorithm has a correct behavior which depends on the system topology, such as spanning tree construction, network decompositions and the algorithm may have an incorrect behavior when topology changes, then the self-stabilizing algorithm is suitable in this case. Since topology changes can be

seen as transient faults that affect the correct behavior of the algorithm, then the self-stabilizing algorithm returns automatically to the correct topology in finite time.

we can also list disadvantages to self-stabilization:

- *High complexity:* The performance of self-stabilizing algorithms are often lower than their equivalent non-self-stabilizing algorithms in case where no transient faults.
- *No termination detection:* The nodes of the system have no way of detecting the termination of the algorithm or aware if a legitimate configuration is reached or not.

2.2.3 Self-stabilization algorithm design

The distributed system is represented by an undirected graph $G = (V, E)$; where V is a set of nodes corresponding to the process and E is a set of edges representing the links between nodes. Let $n = |V|$ and $m = |E|$.

- Two nodes v and u are neighbors if and only if $(v, u) \in E$.
- The set of neighbors of a node v is denoted by $N(v)$, i.e. $N(v) = u \in V|(v, u) \in E$.
- The closed neighborhood of a node v is denoted by $N[v] = N(v) \cup v$.
- We denote by $d(v)$ the degree of a node v (i.e. $d(v) = |N(v)|$) and Δ the maximum node degree in the graph.
- The maximum length of the shortest path between any nodes is called diameter of G and it is denoted by D.
- In the system, every node v has a set of variables whose contents specify the state s_v of the node v.
- The union of the states of all nodes defines the system's global state (or configuration).

Definition 2.1 (Configuration): A configuration c of the graph G is defined as the n-tuple of all node's states: $c = (s_{v1}, ..., s_{vn})$. The set of all configuration is denoted by C_G. Each node has only a partial view of the system [Neg15].

Based on its state and that of its neighbors (distance-one model), a node can make a move which results the changing of at least one value of one or more of its variables. Note that in distance-two model (resp. distance-k model), a node can read its state and the state of nodes of distance at most two (resp. at most k). Therefore, self-stabilizing algorithms are given as a set of rules of the form :

[Rule's label] : [If $p(v)$ *then* M]

The predicate $p(v)$ is defined over $v's$ partial view. The statement M denotes a move that changes only state of the node v. A rule is called *enabled* if its predicate evaluates to true. A node v is also called *enabled* (or privileged) if at least one of its rules is *enabled*. Otherwise, the node v is *disabled*, i.e. all of its rules are disabled. The nodes cooperate to solve a specific problem. This problem is defined by a predicate P.

Definition 2.2 (Legitimate configuration): A configuration c is called *legitimate* (or desired) with respect to P if c satisfies P.

Let $L_P \subseteq C_G$ be the set of all legitimate configurations with respect to a predicate P. An

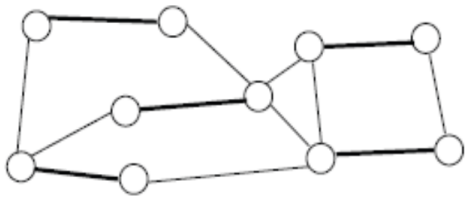

Figure 2.3: A legitimate configuration for matching problem. [Neg15]

example is illustrated in the Figure 2.3 for the problem of matching in graphs; predicate P is evaluated to true if any node in the graph G is matched (married) with only one neighbor. Then any configuration that satisfies P is called legitimate configuration. Otherwise, the configuration is illegitimate.

Definition 2.3 (Execution): An execution x of an algorithm is a maximal sequence of configurations $c_1, c_2, ..., c_i, ..., c_k$ such that each configuration c_{i+1} is the next configuration of c_i using one unit of time. The execution of self-stabilizing algorithms are encapsulated under the notion of daemon. An enabled node v makes a move if and only if v is selected by the daemon i.e. the node v brought into a new state that is a function of its old state and the states of its neighbors [Dij74].

Thus, several daemons have been proposed for designing self-stabilizing algorithms. The following section describes the most common daemons used in the literature.

2.2.4 Daemons

Daemons are one of the most central yet less understood concepts in self-stabilization where the execution of the algorithm is captured by them [Dij74]. Intuitively, the daemon is a mechanism for selecting the enabled (privileged) nodes to execute their moves. This mechanism plays the role of both *scheduler* and adversary against the stabilization of the algorithm. This can be done by scheduling the worst possible cases for algorithm's execution. Thus, the choice of daemon is important in designing of self-stabilizing algorithm in terms of convergence and complexity analysis. Indeed, many types of daemons are assumed in the literature of self-stabilizing algorithms. Dubois presents a good taxonomy of existing daemons in [DT11]. The three most common daemons are the following:

1. *Central daemons:* At each step, the central daemon selects exactly one enabled node to make a move.
2. *Distributed daemons:* The distributed daemon selects in each step a non-empty subset of the enabled nodes to make their moves simultaneously.
3. *Synchronous daemons:* This type of daemon can be considered as a special kind of distributed daemon where in each step all enabled nodes make their move simultaneously.

Daemons are also associated with the notion of *fairness*. A daemon can be fair (weakly), or unfair (adversarial). A daemon is fair if every continuously enabled node is eventually selected. The unfair daemon on the other hand may delay the move of an enabled node as long as there are other enabled nodes. Self-stabilizing algorithms designed for a specific daemon may not operate under a different daemon. However, an algorithm designed for an unfair distributed daemon works with all other daemons.

2.2.5 Complexity measures

The complexity measures are used to evaluate the performance of a self-stabilizing algorithm. These measures include time, memory or the number of messages sent. There are different measures for time complexity of self-stabilizing algorithms. These measures do not consider the local resource demand of the nodes. This is due to the assumption that the time needed for local computation (local resource demand) is negligible (smaller) compared to the time needed for nodes communications. [Neg15]

The measures used to describe the self-stabilization algorithm complexity are:

1. *Move:* A move of a node υ is one transition from state s_υ to a new state $\grave{s}_\upsilon$ after the execution of the statement of an enabled rule in the algorithm.

2. *Step:* step is a tuple $(c, \grave{c})$, where c and $\grave{c}$ are configurations, such that some enabled nodes, in configuration c, make moves during this step, and $\grave{c}$ is the configuration reached after such nodes made their moves simultaneously.

3. *Round:* A Round is a minimal sequence of steps in which every node that was enabled at the beginning of the round, gets the chance to be selected for making a move if it has not become disabled by a move of its neighbors.

4. *Time complexity:* The time complexity of self-stabilizing algorithms is the maximum number of moves, steps or rounds needed for reaching a legitimate configuration, regardless of starting configuration.

[Neg15] has to note that under central daemon, the steps complexity is equivalent to moves complexity, since the daemon selects only one enabled node per step. Moreover, for synchronous daemon, the rounds complexity is equivalent to steps complexity, since under this daemon, a round contains only one step. Since moves complexity is an upper bound of steps and rounds complexities within any daemon, then it would be more interesting to analyze self-stabilizing algorithms using moves instead of steps or rounds.

2.2.6 Transformers

There are several different distributed models assumed in the literature and therefore we need to design different algorithms to solve a problem in each model. A common approach to avoid this is to design general methods that permit to transform an algorithm from one model to other. Thus, many methods, called Transformers have been proposed with self-stabilization. A Transformer converts a self-stabilizing algorithm A that runs under a given model to a new self-stabilizing algorithm A' such that A' runs under another model. Note that both of algorithms (A,A') share the same set of legitimate configurations. Usually, these transformers cause overhead complexity in terms of time or space memory. In general, these transformers can be classified into three types:

1. *Communication model transformers:* In literature, we have three common communication models used in distributed systems: state model, shared-register model and message-passing model. Thus, the design of self-stabilizing algorithms depends heavily on the communication model used in the system and an algorithm under specific model cannot run under another communication model. For this reason, many transformers have been proposed in literature for converting a distributed algorithm and

preserving self-stabilization property such that transformer from shared memory to message passing proposed in [Dol00] and the transformer from message passing to shared memory presented in [Ioa02]. More transformers and details can be found in [Dol00].

2. *Distance-knowledge transformers:* Using model of computation of distributed algorithms, a node can read only its variables and those of its neighbors in the case of distance-one model. Thus, it is easier to design a self-stabilizing algorithm for certain problems assuming that a node can read the variables of nodes that are in distance two or more. The idea in this category of transformation is as follows: each node maintains its variables and copies of variables of its neighbors. Thus, in order to maintain these variables up-to-date, a node can execute a move if and only if all neighbors have given their permission.

3. *Daemon transformers:* In addition to communication model used by a system, the design of self-stabilizing algorithms also depends on the assumption presented in this chapter; Usually, the algorithms designed under central daemon do not stabilize under synchronous or distributed daemon. Indeed, designing self-stabilizing algorithms under central daemon is often convenient. However, the central daemon does not consider concurrent executions of two neighbors and therefore it is not directly practicable in real distributed systems. For this reason, several transformers have been proposed for converting an algorithm designed for central daemon into an algorithm that stabilizes under the distributed daemon. Since the distributed daemon is more general than others daemons, then transformation from the distributed daemon to the central daemon is not required.

In addition, another kind of transformers can also be found in the literature. These transformers allow to convert a distributed algorithm (non self-stabilizing) into a self-stabilizing algorithm [AS88], [KP93]. However, these transformers usually sacrifice either convergence time complexity or memory requirements.

2.2.7 Proof techniques

Most self-stabilizing algorithms for graph problems are silent, then proving their *correctness* is usually not difficult task; i.e. it is sufficient to prove that in configuration where no node is enabled, the configuration of the system is legitimate. However, proving the convergence of these algorithms is a challenging task. For proving *the convergence* (second property) of a self-stabilizing algorithm, several techniques has been proposed in the literature. We cite:

- *Variant Function:* [Kes88] proposed an approach for the first time by using a Variant Function to prove the convergence of self-stabilizing algorithms. This technique measures the progress and the evolution of an algorithm during its execution. The basic idea is to use a function over the configuration set whose value is bounded, to prove that this function monotonically increases (or decreases) when nodes execute any rule. This can be done for example by counting nodes with certain properties.

- *Attractor:* The technique of attractor is used to prove the convergence of a self-stabilizing algorithm when it is difficult to find variant function. The idea is to define a sequence of predicates $p_1, ..., p_k$ over the configuration set, where all legitimate configurations satisfy the predicate p_k. Moreover, each predicate p_{i+1} is a refinement of p_i where $1 \subseteq i \subseteq k$. The predicate p_{i+1} refines the predicate p_i if p_i pi holds whenever p_{i+1} holds. The term attractor is often used for each pi predicate [Dol00]. Then, the goal is to prove that a system in which p_i holds reaches a configuration satisfies p_{i+1}.

- *Global State Analysis:* A single node has not knowledge about the configuration of the whole system. However, this global view can be used for proving the termination of an algorithm. For instance, it may be possible to prove that there is no configuration that can occur twice. This proves that the number of possible configurations is finite due to the fixed number of nodes and their local states. Usually, most algorithms define several local states for each node, this causes an exponential number of possible configurations CG. Hence, this technique may not be a good decision when the goal is to prove the performance of an algorithm. [Neg15]

- *Analysis of Local States and Sequences:* Contrary to the global state analysis, this technique considers only the analysis of the state of a single node and its neighbors. Some systems have the property that nodes become disabled after executing certain moves. The basic idea is to show that any node in the system has a bounded number of moves or bounded number of state sequence. This technique is used in [Tur07] and [SX08] for proving the convergence of self-stabilizing algorithms for dominating set problems.

- *Graph Reduction and Induction:* Recently in [TH11], Turau and Hauck developed a new technique to prove the stabilization under central and distributed daemon. The basic idea of this technique is to create a mapping from the algorithm's execution sequence of a graph to that of a reduced graph. This allows to use complete induction proofs [TH11]. The authors used this technique for finding the worst-case complexity of self-stabilizing algorithms for finding the maximum weight matching with approximation ratio 2.

- *Neighborhood Resemblance:* This technique is used to prove lower bounds of memory

to solve a given problem within self-stabilizing paradigm. In fact, using this technique, we obtain some impossibility results, i.e. it is impossible to find a self-stabilizing algorithm for a given problem with less than a certain amount of memory.

2.3 Examples of self-stabilization algorithms for some graph problems

To study different problems that present different areas (communication networks, scheduling, distributed system, ...), the importance of graph theory appears, several self-stabilizing algorithms for classic graph parameters have been developed in this direction, such as self-stabilizing algorithms for finding minimal dominating sets, coloring, maximal matching, maximal packing, spanning tree. Herman [Her02] presents a list of self-stabilizing algorithms according to several categories such as topology or proof techniques. Gartner [Gar03] surveys self-stabilizing algorithms for spanning trees. Later, Guellati and Kheddouci [GK10] present a survey of self-stabilizing algorithms for independence, domination, coloring, and matching problems. In this part, some references on matching, domination, coloring and independence problems are summarized and more recent works are presented.

2.3.1 Matching

We can find in the literature a considerable number of self-stabilizing algorithms for the matching problem. A matching (Figure 2.4) in a graph $(G.V)$ is a set M of independent (pairwise nonadjacent) edges. A matching M is maximal if no proper superset of M is also a matching [GK10]. A matching of a graph G is called maximum matching if it has largest cardinality among all possible matchings in G. A node that is incident with an edge of a matching M is called matched, and nodes not incident with any edge of M are called unmatched.

A set $M \subseteq E$ of edges is called a generalized matching (or a b-matching) of G if, for all nodes $i \in V$, $|E_i \cap M| \leq b$, where E_i is the set of edges incident with i. An alternating path in a graph $G(VE)$, with respect to a matching M, is a path whose edges are alternately in M and in $(E - M)$. An alternating path is an augmenting path if the first and the last edges are in $(E - M)$.

Matching problem has many applications in fields as diverse as transversal theory, assignment problems [BNBJ$^+$08], network flows [HSM$^+$07], scheduling in switches [WS05]. since it is associated with marriage-like problems where the goal is to form maximum couples while optimizing specific criteria. For example, in networks, each client communicates

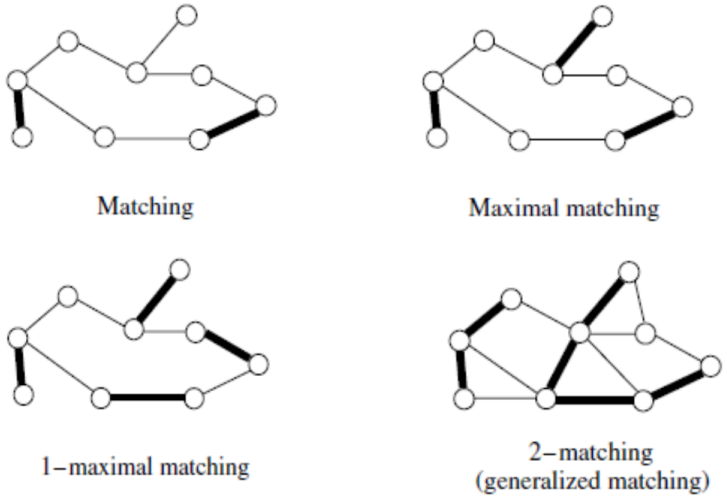

Figure 2.4: Examples of matchings (the shaded edges denote the elements of the matchings). [GK10]

with only one server. More details on applications of matching can be found in [Gib85].

The first self-stabilizing algorithm for computing a maximal matching was proposed by Hsu and Haung in [HH92]. The algorithm is uniform and works in anonymous system. It assumes a central daemon. The proposed algorithm maintains a variable for each node v in the system that contains a pointer. This pointer may be null or may point at a v's neighbor. Two nodes v and u are matched (i.e. married) if and only if they point at each other. Then, in final configuration, each pairs of matched nodes form a maximal matching.

The basic idea of the algorithm is as follows: each node v that points $null$ will point at an arbitrary neighbor u such that u points at v (which means v accepts to be matched with u). If a node v that points to null and any of its neighbors points at v then v points an arbitrary neighbor u such that u points to null (which means that v invites/proposes a node u to be matched). Then, a node v that points at neighbor u and the latter points at another node ω, so v will change its pointer to null (which means that v withdraws its proposition/invitation).

Hsu and Huang proved that the time complexity is $O(n^3)$ steps. The complexity of the same algorithm was improved to $O(n^2)$ steps by Tel in [Tel94] and later it was improved to $O(m)$ steps by Hedetniemi et al. in [HJS01].

Chattopadhyay et al. proposed in [CHS02], two algorithms for the same problem with read/write atomicity. The first algorithm that stabilizes in $O(n)$ rounds assumes that each node has a distinct local identifier. The idea that each node tries to be matched with its neighbor that has the minimum identifier. The authors extend this version by proposing the second algorithm for anonymous system with (n^2) rounds under central daemon. However, the second algorithm assumes that each node knows Δ (maximum node degree in the system)

and G is a bipartite graph.

In [GHJS03b], Goddard et al. proposed a synchronous version of the algorithm Hsu and Haung that stabilizes in $O(n)$ rounds in mobile ad-hoc networks. However, the authors assumed distinct local identifiers for nodes and communication is ensured through message exchanges between nodes. Goddard et al. in [GHS06] proposed a uniform version for finding 1-maximal matching in trees assuming central daemon. A 1-maximal matching is maximal matching and its cardinality cannot be increased by removing an edge and adding two edges. Their algorithm is based on the algorithm of Hsu and Haung and by adding a mechanism for exchanging an edge of the matching by two when it was possible. The proposed algorithm needs $O(n^4)$ steps.

In [MMPT07], Manne et al. proposed an algorithm for maximal matching that stabilizes in $O(m)$ moves under distributed daemon. The authors assumed distinct local identifiers within distance two. In the algorithm, each node maintains two variables, one variable for pointer (the same as used by Hsu and Haung in [HH92]) and one boolean variable for informing neighbors whether the node is matched or not.

We have to note that there is no self-stabilizing algorithm for finding a maximum matching of general graphs in the literature. However, there are some algorithms for certain classes of graph such tree [KS00] or bipartite graphs [CHS02]. Considering weighted graphs, Manne et al. proposed a self-stabilizing for maximum weighted matching in general graph [MM07]. The authors give upper bounds of $O(2^n)$ moves under the central daemon and $O(3^n)$ for the distributed daemon. Recently, Turau and Hauck improve this complexity in [TH11]. The authors present a new analysis of the algorithm proposed by Manne and they proved that the same algorithm stabilizes in $O(mn)$ moves under the central daemon. Moreover, the authors give a modified version that stabilizes within $O(mn)$ moves within the distributed daemon.

2.3.2 Dominating set

Domination in graphs has been extensively studied in graph theory. In a graph $G = (V, E)$, a set of nodes $D \subseteq V$ is called a dominating set (DS) if every node of V is either in D or has a neighbor in D, i.e.

$\forall v \in V - D : N(v) \cap D \neq \emptyset$.

A dominating set is minimal (MDS) if no proper subset of D is a dominating set.

In the literature, there are several self-stabilizing algorithms for finding different variants of dominating sets such as total dominating set, k-dominating set, connected dominating set and weakly connected dominating set.

Reference	Result	Topology	Anon.	Daemon	Complexity
[HH92]	Maximal	Arbitrary	Yes	Central	$O(m)$ moves
[CHS02]	Maximal	Arbitrary	No	Distributed	$O(n)$ rounds
[GHJS03b]	Maximal	Arbitrary	No	Synchronous	$O(n)$ rounds
[GHJS03c]	Generalized	Arbitrary	Yes	Central	$O(m)$ moves
[GHS06]	1-Maximal	Tree	Yes	Central	$O(n^4)$ moves
[MMPT07]	Maximal	Arbitrary	No	Distributed	$O(m)$ moves
[KS00]	Maximum	Tree	Yes	Central	$O(n^4)$ moves
[CHS02]	Maximum	Bipartite	Yes	Central	$O(n^2)$ rounds
[MM07]	1/2 ap. max. wei.	Arbitrary	No	Distributed	$O(3^n)$ moves
[TH11]	1/2 ap. max. wei.	Arbitrary	No	Distributed	$O(mn)$ moves

Table 2.1: Self-stabilizing algorithms for maximal matchings and its variants.

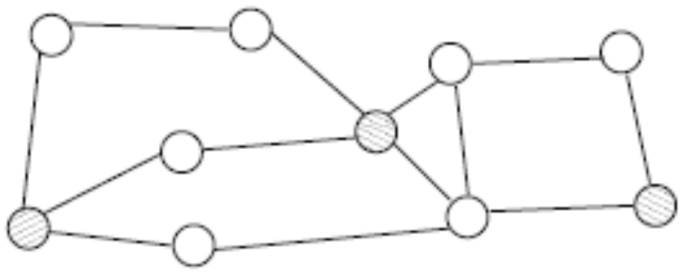

Figure 2.5: Minimal Dominating Set D of a graph G. (The members of D are hatched). [Neg15]

A set D is called total dominating set (TDS) if every node of the graph has a neighbor in D, i.e. $\forall v \in V : N(v) \cap D = \emptyset$. The set D is called k-dominating set (KDS) if every node outside of D has at least k neighbors inside D. A dominating set D is said connected dominating set (CDS) if D is connected and it is called weakly connected (WCDS) if the subgraph weakly induced by D, i.e. $(N[D], E \cap (D \times N[D]))$ is connected, where $N[S] = \bigcup_{v \in S} N[v]$.

The structure of dominating sets can be used as virtual overlays in a distributed system. These structures are often used for designing efficient protocols in wireless and ad-hoc networks[GHJS08], [UT11], [GHJS03b], [BDTC05]. Minimal dominating set can be used for locating some nodes to be servers; thus clients must be closely with the servers [GHJS08].

Connected dominating sets and weakly connected dominating set are often used to represent virtual backbone in wireless networks [BDTC05].

We can cite 4 variants of dominating set algorithms such as summarized in the table 2.2:

- **Simple domination variant**
- **Multiple domination**
- **Connected Dominating Set**
- **Independent Strong Dominating Set**

The contribution of this thesis is to propose a new self stabilization algorithm for finding dominating sets. This concepts will be detailed in the third chapter.

2.3.3 Independent Set

As defined above, a set of nodes is an Independent Set if no two nodes are adjacent. A maximal independent set *(MIS)* is an independent set that is not properly contained in any other independent set with bigger cardinality. The definition of MIS implies that for any graph $G = (V, E)$, if a node is not in the MIS, then it must be adjacent to at least one node in the MIS. Therefore, an MIS of a graph G is also a minimal dominating set, however an MDS is not necessary an MIS. The dominance property of the MIS and the sparseness of its nodes make it an important structure for many applications, such as clustering in wireless ad hoc networks [AWF03]. Since MDS and MIS are strongly related, many self-stabilizing algorithms were also proposed for finding MIS, Other new works are presented in [MPR19], [Gha15]; [Gha19]. The rest of algorithms presented for the independent set problems are summarized in Table 2.3.

2.3.4 Coloring graph

Another graph theoretic problem that was studied in the context of self-stabilization is the coloring of graphs [Die05]. A node coloring (or simply a coloring) of a graph $G(V, E)$ is a function $c \rightarrow N$ such that $c(i) \neq c(j)$ whenever i and j are adjacent. The elements of N are called the available colors. If a graph G may be colored using k colors, we say that it is *k-colorable*.

The smallest integer k such that G is k-colorable is the chromatic number of G, denoted by $X^{(G)}$. A coloring that uses $k \geq X^{(G)}$ colors is called a *proper coloring*. A coloring that uses $X^{(G)}$ colors is called *minimum coloring*. A *Grundy* coloring of a graph is a proper

coloring such that every node i has a neighbor of color r for all r, $1 \leq r \leq c(i)$. The proposed self-stabilization algorithms for the coloring graph problem are in four fields:

- *Coloring bipartite graphs*: in this algorithm, a specific node is designated as a root, and the other nodes determine their distance from the root in a breadth-first manner. Then, nodes at even distance from the root color themselves black and nodes at odd distance from the root color themselves white. This coloring corresponds to a proper 2-coloring of the graph. [SS93].

- *Coloring planar graphs*: Ghosh and Karaata [GK93] presented a uniform algorithm for coloring planar graphs with at most six colors. In fact, the solution described here is a combination of two algorithms, where the first generates a directed acyclic version of the planar graph, and the second colors the generated graph. In the coloring part, each node takes a color not taken by its successors in the directed acyclic graph. The authors firstly supposed composite atomicity and a central daemon. Then, a new version of the algorithm that works under a distributed daemon and a finer level of atomicity has been proposed.

- *Coloring chains and rings*: Shukla et al. [SRR94] studied the problem of developing uniform self-stabilizing algorithms for coloring anonymous chains and rings. They first proposed a uniform algorithm for the 2-coloring of a chain. The algorithm works as follows: each node i maintains an integer variable X_i and keeps it at the value zero if i is at the extremity of the chain, or keeps it at the minimum value X_j of its neighbors plus one. Then, the color of a node i is $C_i = X_i \; mod \; 2$.

  This algorithm produces a 2-coloring of any odd chain under a distributed daemon. For even chains, however, the algorithm does not produce a proper 2-coloring under a central daemon, and moreover there is no uniform deterministic self-stabilizing algorithm that works under a distributed daemon for this problem.

  The second algorithm presented by Shukla et al. is uniform and produces a 3-coloring for any oriented ring under a central daemon. The idea of the algorithm is as follows: each node maintains a variable that represents its color. If a node has the same color as its two neighbors, it will take the minimum of the available colors not taken by its neighbors. And, if a node has the same color as its left neighbor, it will take any other available color.

  Here, also, there is no uniform deterministic self-stabilizing algorithm that works under a distributed daemon for the 3-coloring for an oriented ring. However, a randomized solution can be designed for this problem.

- *Coloring arbitrary graphs*: Gradinariu and Tixeuil [GT00] presented a uniform algorithm for coloring any graph with at most $(\Delta + 1)$ colors. The algorithm works under

a central daemon and assumes that the system is anonymous. It is also assumed that each node knows Δ, the maximum degree of the system's communication graph. In this algorithm the nodes always try to take the maximum color from the set $0, ..., \Delta$ which is not taken by a neighbor. So, after stabilization they obtain a proper coloring of the system's communication graph. Gradinariu and Tixeuil modified the previous algorithm so that it may work under a distributed daemon in two ways.

The most works realized for self-stabilization in coloring graph problem are summarized in Table 2.4:

2.4 Conclusion

Self-stabilizing algorithms are used in most graph problems; therefore graphs are the best method to present and study the behavior of systems, networks and the change of their states. the graphs are proven their importance and effectiveness in the study of: communication in networks, scheduling of processes and tasks, distributed systems, social networks, ..., which are exposed to faults, breakdowns and change of topologies. Such a situation requires a self-stabilizing algorithm to overcome it and bring the system to a stable and legitimate state.

In this chapter, we have tried to summarize these problems and how a self-stabilizing algorithm is used; we started by defining such algorithms and these basic concepts, then we cited important works from the literature.

Among the fields of application of self-stabilizing algorithms is the search for the dominating set which is the subject of the 3rd chapter of this thesis.

Reference	Result	Topology	Anon.	Daemon	Complexity
[HHJS03]	DS	Arbitrary	Yes	Central	$O(n)$ moves
[HHJS03]	MDS	Arbitrary	Yes	Central	$O(n^2)$ moves
[XHGS03]	MDS	Arbitrary	No	Synchronous	$O(n)$ rounds
[GHJS08]	MDS	Arbitrary	No	Distributed	$O(n)$ moves
[Tur07]	MDS	Arbitrary	No	Distributed	$O(n)$ moves
[CCT14]	MDS	Arbitrary	No	Distributed	$O(n)$ moves
ISDS	MDS/MIS	Arbitrary	No	Distributed	$O(n)$ rounds
[GHJS03a]	MTDS	Arbitrary	No	Central	Exponential moves
[BYK14]	MTDS	Arbitrary	No	Distributed	$O(mn)$ moves
[KK03]	MKDS	Tree	Yes	Central	$O(n^2)$ moves
[KK03]	MKDS	Tree	No	Distributed	$O(n^2)$ moves
[HCW08]	M2DS	Arbitrary	Yes	Central	$O(mn)$ moves
[KK05]	MKDS	Arbitrary	No	Synchronous	$O(n^2)$ rounds
[HLCW07]	M2DS	Arbitrary	No	Distributed	
[DLV10]	MKDS	Arbitrary	No	Distributed	$O(k)$ rounds
[DHR$^+$11]	MKDS	Arbitrary	No	Distributed	$O(Dn^2)$ rounds
[Tur12]	MKDS	Arbitrary	No	Distributed	$O(nm)$ moves
[JG05]	CDS	Arbitrary	No	Synchronous	$O(n^2)$ moves
[DFG06]	CDS	Arbitrary	No	Distributed	$O(n)$ moves
[GS10]	CDS	Arbitrary	Yes	Distributed	
[KK10]	CMDS	BFS tree	No	Central	$O(k)$ rounds
[KK08]	CMDS	Arbitrary	No	Synchronous	$O(n)$ rounds
[RTAS09]	CMDS	DGB	No	Central	$O(n^2)$ moves
[SX07]	WCMDS	BFS tree	No	Distributed	$O(2^2)$ moves
[TH09]	WCMDS	BFS tree	No	Distributed	$O(mn)$ moves
[DWS14b]	WCMDS	Arbitrary	No	Synchronous	$O(n)$ rounds
[Ben21]	DS	Dynamic Network	No	Central	$O(n)$ moves

Table 2.2: Self-stabilizing algorithms for dominating sets and its variants.

Reference	Result	Topology	Anon.	Daemon	Complexity
[SRR95]	MIS	Arbitrary	Yes	Central	$O(n)$ moves
[IKK02]	MIS	Arbitrary	No	Distributed	$O(n^2)$ steps
[GHJS03b]	MIS	Arbitrary	No	Synchronous	$O(n)$ rounds
[SGH04]	1-MIS	Tree	Yes	Central	$O(n^2)$ moves
[Tur07]	MIS	Arbitrary	No	Distributed	$O(n)$ moves
ISDS	MDS/MIS	Arbitrary	No	Distributed	$O(n)$ rounds
[Ben21]	MIS	Dynamic Network	No	Central	$O(n)$ moves

Table 2.3: Self-stabilizing algorithms for maximal independent set and its variants.

Reference	Result	Topology	Anon.	Daemon	Complexity
[SS93]	CG	Bipartite graph	Yes	Central	
[KK06]	CG	Bipartite graph	YES	Central	$O(mn^3d)$ steps
[KK06]	CG	Bipartite graph	No	Distributed	$O(mn^3\Delta d)$ steps
[GK93]	CG	Planar graph	No	Distributed	
[HHT05]	CG	Planar graph	Yes	Central	$O(d)$ rounds
[SRR94]	CG	Odd chain	Yes	Distributed	
[SRR94]	CG	Oriented rings	Yes	Central	
[GT00]	CG	Arbitrary	Yes	Central	$O(n\Delta)$ steps
[GT00]	CG	Arbitrary	No	Distributed	$O(n\Delta)$ steps
[GT00]	CG	Arbitrary	Yes	Distributed	$O(n\Delta)$ steps
[HHT03]	CG	Arbitrary	Yes	Central	$O(m)$ steps
[HHT03]	CG	Arbitrary	Yes	Central	$O(n)$ steps
[HHT03]	CG	Arbitrary	Yes	Central	
[Ben21]	CG	Scale-free graphs	No	Central	$O(n)$ moves

Table 2.4: Self-stabilizing algorithms for coloring graph problem and its variants.

Chapter 3

Algorithms for dominating sets

CHAPTER 3

ALGORITHMS FOR DOMINATING SETS

Contents

3.1 Introduction

The following problem can be said to be the origin of the study of dominating sets in graph. Figure 3.1 illustrates a standard 8 X 8 chessboard on which is placed a queen. According to the rules of chess a queen can, in one move, advance any number of squares horizontally, vertically, or diagonally (assuming that no other chess piece lies in its way). Thus, the queen in figure can move to (or attack, or dominate all of the squares marked with an 'x'). In the 1850s, chess enthusiasts in Europe considered the problem of determining the minimum number of queens that can be placed on a chessboard so that all squares are either attacked by a queen or are occupied by a queen. Figure 3.1 illustrates a set of six queens which together attack, or dominate, every square on the board. It was correctly thought in the 1850s, that five is the minimum number of queens that can dominate all of the squares of an 8X8 chessboard. The Five Queens Problem is to find a dominating set of five queens. [HHS98]

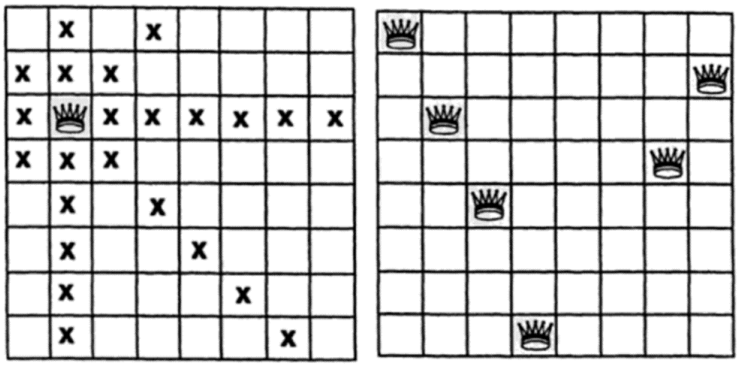

Figure 3.1: Five Queens Problem. [HHS98]

To model the queens problem on a graph, let G represent the chessboard such that each vertex corresponds to a square, and there is an edge connecting two vertices if and only if the corresponding square are separated by any number of squares horizontally, vertically, or diagonally. Such a set of queens in fact represents a dominating set. For another motivation of this concept, consider a bipartite graph where one part represents people, the other part represents jobs, and the edges represent the skills of each person. Each person may take more than one job. One is interested to find the minimum number of people such that are jobs are occupied. [Ken06]

The concept of dominating set; studied in this thesis by founding a self-stabilizing algorithm; occurs in variety of problems where the puzzle or five queens problem are only

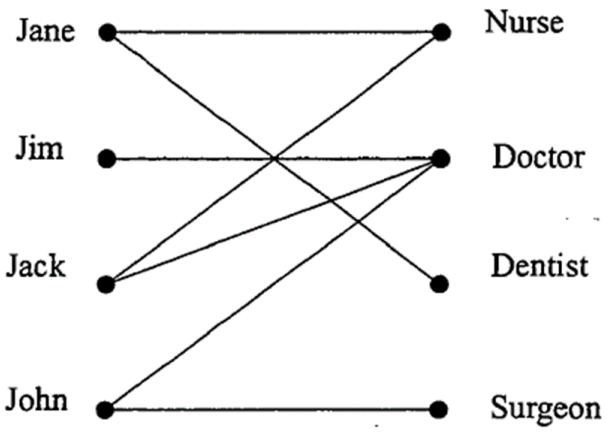

Figure 3.2: Dominated jobs. [Ken06]

interesting examples. A number of these problems are motivated by communication network problems, for example. The communication network includes a set of nodes, where one node can communicate with another if it is directly connected to that node. In order to send a message directly from a set of nodes to all others, one needs to choose this set such that all other nodes are connected to at least one node in the set. For other applications of domination, the facility location problem, land surveying, and routings can be mentioned.

In this chapter we will introduce preliminaries and concepts that define the dominating set in graphs; algorithms used and proposed to find the minimal dominating set in different fields.

3.2 Definitions

Definition 1: A set $S \subseteq V$ of vertices in a graph $G = (V, E)$ is called dominating set if every vertex $v \in V$ is either an element of S or is adjacent to an element of S.

Definition 2: For $S \subseteq V$, a vertex $v \in S$ is called an enclave of S if $N[v] \subseteq S$, and $v \in S$ is an isolate of S if $N(v) \subseteq V - S$. A set is said to be enclaveless if it does not contain any enclaves. [HHS98]

There are several different ways to define a dominating set in a graph, each of which illustrates a different aspect of the concept of domination. Consider the following equivalent definitions.

A set $S \subseteq V$ of vertices in a graph $G = (V, E)$. is a dominating set if and only if:

1. For every vertex $v \in V - S$, there exists a vertex $u \in S$ such that v is adjacent to u;

2. For every vertex $v \in V - S$, $d(v, S) \leq 1$.
3. $N[S] = V$
4. For every vertex $v \in V - S$, $|V(v) \cap S| \geq 1$, that is, every vertex $v \in V - S$ is adjacent to at least one vertex in S;
5. For every vertex $v \in V$, $|V(v) \cap S| \geq 1$.
6. $V - S$ is enclaveless.

Notice that if S is a dominating set of a graph G, then every superset $S' \supseteq S$ is also a dominating set. On the other hand, not every subset S" $\subseteq S$ is necessarily a dominating set. We will be interested in studying minimal dominating sets in graphs, where a dominating set S is a minimal dominating set if no proper subset S" $\subseteq S$ is a dominating set. The set of all minimal dominating sets of a graph G is denoted by **MDS(G)**.

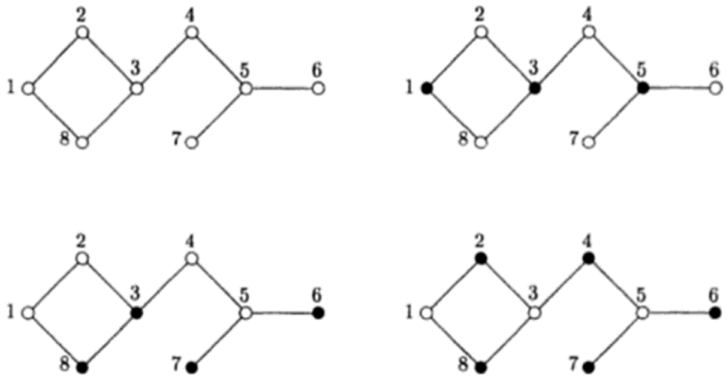

Figure 3.3: Example of minimal dominating sets. [HHS98]

The theorems about dominating sets were given by [Ore62]:

Theorem 1: A dominating set S is a minimal dominating set if and only if for each vertex $u \in S$, one of the following two conditions is holds:

- u is an isolate of S.
- there exists a vertex $v \in V - S$ for which $N(v) \cap S = \{u\}$.

The theorem 1 proofed in [Ore62], suggests the following definition:

let S be a set of vertices, and let $u \in S$; we say that a vertex v is a *private neighbor* of u (with respect to S) if $N[V] \cap S = \{u\}$.

Furthermore, we define the *private neighbor* set of u, with respect to S, to be $pn[u, S] = \{v : N[v] \cap S = \{u\}\}$. Notice that $u \in pn[u, S]$ if u is an isolate in (S), in which case we say that u is its own private neighbor.

Given this terminology, we can say that a dominating set S is a minimal dominating set if and if every vertex in S has at least one private neighbor, that is, for every $u \in S$, $pn[u, S] \neq \phi$.

For example, consider the MDS3, 6, 7, 8 in the graph in Figure 3.3. Vertex 3 has vertices 2 and 4 as private neighbors, vertex 8 has vertex 1 as a private neighbor, while vertices 6 and 7 are their own private neighbors.

Theorem 2: every connected graph G of order $n \geq 2$ has a dominating set S whose complement $V - S$ is also a dominating set.

Proof [HHS98]; Let T any spanning tree of G, and let u be any vertex in V. then the vertices in T fall into two disjoint sets S and S' consisting, respectively, of the vertices with an even and odd distance from u in T; both S and S'$= V - S$ are dominating sets for G.

Theorem 3: if G is a graph with no isolated vertices, then the complement $V - S$ of every minimal dominating set S is a dominating set.

Proof [HHS98]: Let S be any minimal dominating set of G; assume vertex $u \in V$ is not dominated by any vertex in $V - S$; since G has no isolated vertices, u must be dominated by at least one vertex in $S - \{u\}$, that is $S - \{u\}$ is a dominating set, contradicting the minimality of S. Thus every vertex in S is dominated by at least one vertex in $V - S$, and $V - S$ is dominating set.

Definition 3: the domination number $\gamma(G)$ of a graph G equals the minimum cardinality of a set in MDS(G), or equivalently, the minimum cardinality of a dominating set in G. the upper domination number $\Gamma(G)$ equals the maximum cardinality of a set in MDS(G), or equivalently, the maximum cardinality of a minimal dominating set of G.

It is easy to see that for the graph G in Figure 3.3, $\gamma(G) = 3$; while $\Gamma(G) = 5$. Notice that the set S = 1, 3, 5 is dominating set of minimum cardinality; this is called a $\gamma - set$ of G.

Notice further that S is an *independent set*. This is also called an *independent dominating set of* G. The minimum cardinality of an independent dominating set of G is the independent domination number $i(G)$.

One of the earliest and most basic theorems about the domination number of a graph is the following, due to Nieminen [NIE73]: Let $\varepsilon_F(G)$ denote the maximum number of pendant edges in spanning forest of G (a forest is an acyclic graph).

Theorem 4: For any graph G, $\gamma(G) + \varepsilon_F(G) = n$. [Ore62]

3.3 Algorithms of dominating set

3.3.1 K-domination

A subset $D \subseteq V(G)$ is dominating in G if every vertex of $V(G)\backslash D$ has at least one neighbor in D. Similarly, a subset $D \subseteq V(G)$ is k-dominating in G if every vertex of $V(G)\backslash D$ has at least k neighbors in D. The domination number $\gamma(G)$ and the k-domination number $\gamma_k(G)$ of G are the minimum cardinalities of a dominating and a k-dominating set of G, respectively. [EB20]

We say that a connected graph G is a (γ, γ_k)-graph if $\gamma_k(G) = \gamma(G) + k - 2$ and $\Delta(G) \geq k$. A connected graph G is (γ, γ_k)-perfect if $\delta(G) \geq k$ and every connected induced subgraph H of G with $\delta(G) \geq k$ satisfies the equality $\gamma_k(H) = \gamma(H) + k - 2$.

Fink and Jacobson [FJ85], introduced k-domination in graphs as a generalization of the concept of domination. Motivated by this definition, related problems have been studied extensively by many researchers [BJ16], [Sha09], a good survey on k-domination and k-independence is given by Chellali et al. [CFHV12]. Fink and Jacobson [FJ85] proved the following result on the relation between the domination number and the k-domination number of G.

theorem 5: For any graph G with $\Delta(G) \geq k \geq 2$, $\gamma_k(G) \geq \gamma(G) + k - 2$.

Although it is proved that the above inequality is sharp for every $k \geq 2$, the characterization of graphs attaining the equality is still open, even for the small values of k. Recently, we considered a large class of graphs and gave a characterization for the members satisfying the equality $\gamma_2(G) = \gamma(G)$.

[EB20] also proved that it is NP-hard to decide whether this equality holds for a graph. Moreover, he gave a necessary and sufficient condition for a graph to satisfy $\gamma_2(G) = \gamma(G)$ hereditarily. Some similar problems involving different domination-type graph and hypergraph invariants were considered.

3.3.1.1 Varieties of k-domination

k-Independent dominating sets [FHHR02] mainly deals with subsets of vertices which are k-independent and 1-dominating.

(p, k)-Domination and (p, k)-Independence in [BHS94], Bean, Henning and Swart defined (p, k)-dominating and (p, k)-independent sets as follows: let p and k be positive inte-

gers. A set D of vertices of a graph G is a (p,k)-dominating set if every vertex not in D is within distance p from at least k vertices of D. The subset D is a (p,k)-independent set if every vertex of D is within distance p from at most $k-1$ other vertices of D. The (p,k)-*domination number*, $\gamma_{(p,k)}(G)$, is the minimum cardinality among all (p,k)-dominating sets of G, and the (p,k)-*independence number*, $\beta_{(p,k)}(G)$, is the maximum cardinality among all (p,k)-independent sets of G.

The concept of (p,k)-domination is a generalization of the two concepts of distance domination and k-domination, and the concept of (p,k)-independence is a generalization of the two concepts of distance independence and k-independence. In particular, for $p=1$, a (p,k)-dominating set of G is a k-dominating set and a (p,k)-independent set of G is a k-independent set.

Connected k-Domination defined such as: subset $D \subseteq V(G)$ is a connected k-dominating set of a connected graph G, if D is a k-dominating set of G and the subgraph induced by the vertex set D is connected. The connected k-domination number $\overset{c}{\underset{k}{\gamma}}(G)$ is the minimum cardinality among the connected k-dominating sets of G. [Vol09]

Roman k-Domination in this section, we focus on results about an extension of the Roman dominating function which was suggested by ReVelle and Rosing [RR00]. According to [CPADJH04], Constantine the Great (Emperor of Rome) issued a decree in the fourth century AD for the defense of his cities. He decreed that any city without a legion stationed to secure it must neighbor another city having two stationed legions. If the first were attacked, then the second could deploy a legion to protect it without becoming vulnerable itself. The objective, of course, is to minimize the total number of legions needed. However, the Roman Empire had a lot of enemies, and if a number of k enemies attack k cities without a legion, then the cities are secured in the above sense if they are neighbored to at least k cities having two stationed legions. This leads in a natural way to the following generalization of the Roman dominating function.

A Roman k-dominating function on G is a function $f : V(G) \rightarrow 0,1,2$ such that every vertex u for which $f(u) = 0$ is adjacent to at least k vertices $v_1, v_2, ..., v_k$ with $f(v_i) = 2$ for $i = 1, 2, ..., k$. The weight of a Roman k-dominating function is the value $f(V(G)) = \sum_{u \in V(G)} f(u)$. The minimum weight of a Roman k-dominating function on a graph G is called the Roman k-domination number $\gamma_{kR}(G)$ of G. The Roman k-domination number was introduced by KÃ¤mmerling and Volkmann [KV09] in 2009. Note that the Roman 1-domination number $\gamma_{1R}(G)$ is the usual Roman domination number $\gamma_R(G)$. A Roman k-dominating function of minimum weight is called a γ_R-function.

The 2-Domination Subdivision Number The 2-domination subdivision number $sd_{\gamma 2}(G)$ of a graph G is the minimum number of edges that must be subdivided (where each edge in G can be subdivided at most once) in order to increase the 2-domination number. For

example, $sd_{\gamma 2}(K_n) = 2$ when $n \geq 3$ and $sd_{\gamma^2}(Kp, q) = 3$ when $p, q \geq 4$. In 2008, Atapour, Sheikholeslami, Hansberg, Volkmann and Khodkar [ASH$^+$08] initialized the study of the 2-domination subdivision number.

Positive influence dominating set [FHE$^+$11] defines the meaning of positive and negative node as: each node can have either positive or negative impact on its neighbor nodes. We call a node with *positive impact* a positive node and a node with *negative impact* a negative node. *The positive degree* of a node is the number of its positive neighbors. The same holds for negative degree. The compartment of a node decides whether the node is a positive or a negative node. Nodes that are chosen into the PIDS are marked as positive nodes. Thus a neighbor u of v is a positive neighbor if u is initially a positive node or u is selected into the PIDS. A positive influence dominating set P of a graph G is a subset of nodes in G that any node u in G are dominated by at least $\lceil \frac{d(u)}{2} \rceil$ nodes in P where $d(u)$ is the degree of node u.

The main idea of PIDS algorithm is as follows: first prune the original graph by removing the initial positive nodes, then iteratively choose a 1-dominating set of the graph consisting of nodes with less than half neighbors as positive neighbors until all nodes in the original graph are either positive nodes or have more positive neighbors than negative ones. To choose a 1-dominating set of a graph, we use a *greedy algorithm*. This algorithm selects the node with the largest node degree into the dominating set.

3.4 Dominating set applications

The applications of MDS are quite rich that deal with large problems that we cannot find exact solutions, those problems represent a wide dissemination of information. They are reformulated in graphs where the nodes are the actors of the problem and the vertex are the interactions and the relationship between those nodes. Dominating set can be used in very large-scale networks, the study of social networks, the design of wireless sensor network, the protein interaction networks, covering codes, ...

3.4.1 MDS in wireless networks

Many works are proposed in the literature using the algorithms of MDS to solve problems appearing in wireless networks such as:

- **Very large-scale networks** [NHNT20] used k-domination set to determine the domination relation between pairs of vertices; they construct the minimum k dominating set of a graph. Its application in determining a good approximation of large-scale social networks. Considering the MkDSP in the context of social networks.

The novel features of their method are (i) a prepossessing phase that reduces the graph's size; (ii) a construction phase with different greedy algorithms; and (iii) a post-optimization phase that removes redundant vertices. In all phases, they used techniques to reduce the number of times to compute k-neighbor set of vertices which is very expensive on graphs arisen in social networks.

The application was taken of a company that runs a very large social network in which users can be modeled as nodes and the relationship among users can be modeled as edges. One of the important tasks of the company is monitoring all the activities (conversations, interactions, etc.) of the network users to detect anomalies such as cheating or spreading fake news. With millions of users, it is impossible to observe all users in the network, a potential solution is to construct a subset of users that can represent key properties of the network. The obtained results show that this proposition provides a better tradeoff between the solution quality and the computation time than existing methods.

Data	HEU_1		HEU_3		HEU_4	
	Sol	T (s)	Sol	T (s)	Sol	T (s)
$k = 1$						
soc-delicious	215261	19.07	*56066*	1464.84	56600	5679.63
soc-flixster	1452450	999	_	_	*91543*	27374.44
hugebubbles	1213638	2087.83	_	_	*1169394*	7498.20
soc-livejournal	1538044	2689.72	_	_	*930632*	75185.96
soc-partner-1	6263241	64228.04	_	_	*29278*	26740.42
soc-partner-2	4129393	19109	_	_	*38303*	38644.65
$k = 2$						
soc-delicious	34516	3.57	_	_	*8155*	2064.00
soc-flixster	48789	85.93	_	_	*9860*	23694.58
hugebubbles	943233	792.96	_	_	*777960*	41874.54
soc-livejournal	447552	1582.87	_	_	*189121*	16728.22
soc-partner-1	*11102*	59.78	_	_	12200	31962.21
soc-partner-2	20343	84.15	_	_	*19896*	27159.79
$k = 3$						
soc-delicious	14806	2.44	_	_	*1505*	1695.77
soc-flixster	20996	29.71	_	_	*313*	3333.45
hugebubbles	843077	649.47	_	_	*688817*	17221.76
soc-livejournal	211894	394.98	_	_	*83710*	42600.51
soc-partner-1	6337	55.57	_	_	*5158*	5200.14
soc-partner-2	12807	78.3	_	_	*10905*	5481.59

Figure 3.4: Results of the MkDSP proposed by [NHNT20].

- **Study in social networks** [FHE+11] used the positive influence dominating set to (PIDS) to construct a minimum dominating set of a social network which consists of individuals with a certain type of social problem (such as drinking, smoking and drug related issues) is helpful for the success of intervention programs. Intervention programs are important tools to help combat some of the social problems and consist of disseminated education and therapy via mail, Internet, or face-to-face interviews. In a social setting, people can have both positive and negative impacts on each other, and a person can take and move among different roles since they are affected by their peers.

For example, within the context of drinking problem, a binge drinker can be converted to an abstainer through intervention program and have positive impact on his direct friends (called neighbors). However, he might turn back into a binge drinker and have negative impact on his neighbors if many of his friends are binge drinkers.

Ideally, [FHE+11] wanted to educate all binge drinkers, since this will reduce the possibility of converted binge drinker being influenced by his binge drinker friends who are not chosen in the intervention program. On the other hand, due to the budget limitations, it is impossible to include all the binge drinkers in the intervention program.

Therefore, how to choose a subset of individuals to be part of the intervention program so that the effect of the intervention program can spread through the whole group under consideration becomes an important research problem.

[CTB15] have introduced two novel algorithms that are able to determine efficiently an approximation to the minimum dominating set problem and, simultaneously, they will preserve the quality of the solution to an acceptable level. using Erdos-Renyi random graph model that generates a scale free network.

3.4.2 Design of wireless sensor networks

[VKK15] gave a pertinent survey and study among the use of connected dominating set CDS in wireless sensor networks. The CDS construction algorithms are classified into different types. It is based on whether the network topology is prescient or not, network models, efficiency of the algorithm in forming a small size CDS and its time and information complexity.

According to the network topology is prescient or not, they can be centralized or distributed algorithms and according to the network models, the CDS can be classified to undirected graphs or directed graphs. Also, According to the efficiency of the algorithm while forming the minimum CDS, the CDS construction algorithms can be further divided into four types: Global protocols, Quasi-global protocols, Quasi-local protocols and Local protocols.

3.4.3 Health service

[MMBP11] introduced a new network-based approaches and apply them to get new insight into biological function and disease. They used the notion of domination and find dominating sets (DSs) in the protein-protein interaction PPI network, i.e., sets of proteins such that every protein is either in the DS or is a neighbor of the DS. Clearly, a DS has a topologically central TC role, as it enables efficient communication between different network parts.

[AME21] proposed a method to apply constraint-based models for dominating protein interaction networks. the authors have introduced a new framework called MOIA, in which three models have been modified to generate multiple MDSets with minimum inter sections for PPI networks.

3.5 Conclusion

In this chapter, we introduced the domination set and how use to find a dominating set in a graph and the theorems proofed in the literature to find the minimal dominating set. Then we cited some works and some applications and benefits of finding the minimal dominating set.

Again we will return to the self-stabilizing algorithm that is a fault tolerance approach for distributed systems that has been introduced for the first time by Dijkstra [Dij74]. A self-stabilizing distributed system is able to achieve a global correct configuration (without any external intervention), in a finite time, starting from an initial illegitimate configuration.

Various self-stabilizing distributed algorithms have been proposed in the literature using graph theory such as leader election, nodes coloring, domination problems, independent set identification, spanning tree construction. The reader can refer to the survey [GK10] for more details of self-stabilizing algorithms.

We will show in the next chapter the use of self-stabilizing algorithm to find the minimal dominating set in a graph introducing the parameters proposed in the literature to give efficiency results in different applications. after that we will introduce the proposition and the objective of this thesis.

Chapter 4

Self-stabilizing Algorithms for Minimal Dominating Set

CHAPTER 4

SELF-STABILIZING ALGORITHMS FOR MINIMAL DOMINATING SET

Contents

4.1 Introduction

Domination has been extensively studied in literature [HHS98] and adopted in many real-life applications. It has been utilized for address routing, power management and clustering issues in ad-hoc networks [LLY09, AWF03, BDTC05]. Recently, particular parameters of domination have been used to influence (and change) the opinion of the users in the social networks [FEK09, AKL18]. A dominating set is a subset S of the graph nodes where every node is either in S or is a neighbor of at least one node of S. The dominating set S is minimal if there is no proper subset in S that could be a dominating set.

Before 2003, no self-stabilizing algorithms have been introduced to solve (minimal) dominating sets problem. At that time, algorithms are in general based on greedy, exact and heuristic methods. The first *self-stabilizing* algorithm for (minimal) dominating set was proposed by Hedetniemi *et al.* [HHJS03]. After that, many variants of self-stabilising algorithms have been proposed imposing additional parameters of domination like total domination [BYK14, GHJS03a], efficient domination [Tur13, HHJ$^+$12], connected dominating set [BBP13, DWS16], influence domination [WWTZ13, DWS14a], distance-k domination [DDL18]. Each parameter has its benefits according to the used application. For example, connected dominating sets are generally used as backbone (infrastructure) in ad-hoc and sensor networks.

This thesis deals with the problem of finding dominating set using self-stabilization paradigm in distributed systems. Usually, members of a dominating set are selected to be as cluster heads in Wireless Sensor Networks (WSN) in order to ensure a permanent service availability. Since failures occurs frequently inside WSN due to limited battery energy, self-stabilizing algorithm allows recomputing the dominating set, and hence the network returns to its ordinary running.

Existing works have introduced many variants of self-stabilizing algorithms that compute minimal dominating set S where each node out of S has neighbors in S more than it has out S. In this thesis, we introduce a generalized self-stabilizing algorithm called minimal (α, β)-dominating set. An α-dominating set is a subset of nodes S such that for any node v out of S, the rate of neighbors of v inside S must be greater than α, where $0 < \alpha \leq 1$. In the same way, an (α, β)-dominating set is a subset of nodes S such that: S is α-dominating set and for each node v in S, the rate of neighbors of v inside S is greater than β, where $0 \leq \beta \leq 1$. Mathematical proofs and simulation tests show the correctness and the efficiency of the proposed algorithm.

Through our proposed variant (α, β)-domination, we prove rigorously the conjecture of Carrier $et.\ al.$ [CDD$^+$15] who have proposed a self-stabilizing algorithm for a domination variant called (f, g)-alliance set only when $f \geq g$.

We prove the correctness of the case $f < g$.

4.2 Related Work

4.3 Self-stabilizing distributed algorithms for dominating set

In this section, we will describe some proposed and proofed self-stabilizing algorithms for dominating set in different environments. as we discuss in the second chapter, self-stabilization is a paradigm for distributed systems that allows the system to reach a desired global state, even in the presence of faults. A fundamental idea of self-stabilizing algorithms is that no matter what global state in which the system finds itself, after a finite amount of time, the system will achieve a correct and desired global state.

This concept is used to identify sets of nodes or sets of edges which satisfy a given property P called minimal dominating set. Previous work in this area has produced self-stabilizing algorithms for centers and medians of trees [AS99], maximal matchings [HH92][HJS01], graph colourings [GK93][SS93], shortest paths [Hua05], articulation points [Kar99], and spanning trees [SX07]. In many of these papers, correctness proofs for algorithms are given, but an analysis is not provided.

4.3.1 Dominating bipartition [GHJS03b]

Hedetniemi and al. proposed the domination bipartition algorithm where node i has a single binary variable $x(i)$.The rules allow a node to change its value if all nodes in its closed neighborhood have the same value. Upon stabilization, the two sets of nodes $i|x(i) = 0$ and $i|x(i) = 1$ are each dominating sets, if G has no isolated nodes, thus forming a dominating bipartition.

Figure 4.2 shows two different executions, (a)-(d) and (e)-(f), that begin with the same initial configuration i.e. starting with the initial configuration shown in (a), diagrams (a)-(d) depict one execution of Algorithm 2.1. Nodes i for which $x(i) = 1$ are black; starting with the same initial configuration, diagrams (e) and (f) show an alternate execution, stabilizing in only one move. Diagrams (d) and (f) depict stable states.

[GHJS03b] proofed the algorithm 4.1 in this way:

LEMMA 1: If node i ever makes a move, either Rl or $R2$, it will never make another move, nor will any of its neighbors.

Input: n: Dominating bipartition,
Begin

R1 if $x(i) = 0 \bigwedge(\forall j \in N(i))(x(j) = 0)$
then $x(i) = 1$
R2 if $x(i) = 1 \bigwedge(\forall j \in N(i))(x(j) = 1)$
then $x(i) = 0$

End.

Figure 4.1: Dominating bipartition.

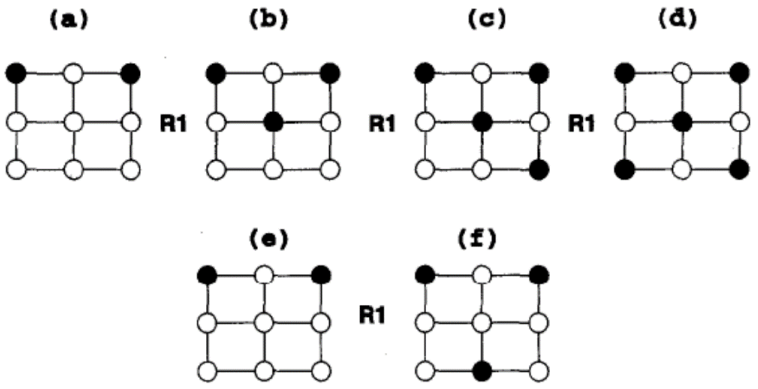

Figure 4.2: Execution of the algorithm Dominating bipartitions. [GHJS03b]

PROOF: Let i and j be neighbors. A node can move only if it and its neighbors all have the same value. Once i moves, $x(i)$ and $x(j)$ will be different, and so neither node can move.

LEMMA 2: Algorithm 4.1 can make at most $n - 1$ moves.

PROOF: Since the network has no isolated vertices, the first node that moves must have at least one neighbor. By Lemma 1, neither of these nodes will be able to move thereafter. Also by Lemma 1, any remaining nodes can move at most once.

LEMMA 3: When Algorithm 4.1 stabilizes, every node labeled 0 has at least one neighbor labeled 1, and conversely, every node labeled 1 has at least one neighbor labeled 0.

THEOREM 1: In any network having no isolated nodes, Algorithm 4.1 stabilizes with a dominating bipartition in at most $n - 1$ moves.

PROOF: This is immediate from Lemmas 2 and 3 The bound given in Theorem 1 is right. Consider any star $K_{1,n-1}$ in which every node is initially 1. If every leaf moves, there

Input: n: Minimal dominating set,
Begin

R1 if $x(i) = 0 \bigwedge(\forall j \in N(i))(x(j) = 0)$
then $x(i) = 1$
R2 if $(x(i) = 1) \bigwedge(\nexists j \in N(i))(j \rightarrow i) \bigwedge(\exists k \in N(i))(x(k) = 1)$
then $x(i) = 0$
P1 if $x(i) = 1) \bigwedge(i \nrightarrow null$
then $i \rightarrow null$
P2 if $x(i) = 0) \bigwedge(\exists$ exactly one $j \in N(i))((x(j) = 1) \bigwedge(i \nrightarrow j))$
then $i \rightarrow j$
P3 if $(x(i) = 0) \bigwedge(\exists$ more than one $j \in N(i))((x(j) = 1)) \bigwedge(i \nrightarrow null)$
then $i \rightarrow null$

End.

Figure 4.3: Minimal dominating set.

will be exactly $n - 1$ moves.

4.3.2 Minimal dominating set [GHJS03b]

Hedetniemi and al. proposed the algorithm using a graph without isolated nodes, it produces a dominating bipartition where the nodes labeled 1 define a *minimal dominating set*, and the nodes labeled 0 define a *dominating set*.

Algorithm 4.2 uses two variables. The first variable is a binary variable $x(i)$ defining a minimal dominating set $S = i|x(i) = 1$. We will use S_t to denote this set at time t. The second variable is a pointer. By pointing to a neighbor j, written $i \rightarrow j$, a node i communicates to j that i is a private neighbor; that is, node j is the only node in S which currently dominates node i. The value null is used for nodes in S and nodes in $V - S$ that are not private neighbors.

We write $i \nrightarrow j$ to denote that i is not pointing to j, and we write $i \rightarrow null$ to denote that the pointer of i is not null. This algorithm is based on the following well-known and straightforward characterization of minimal dominating sets, whose proof can be found in [HHS98].

LEMMA 4: A set S is a minimal dominating set if and only if it is dominating and every $u \in S$ has a private neighbor.

[GHJS03b] called rules $P1 - P3$ pointer moves. They do not modify membership in the dominating set, but rather are used only to adjust pointer values so that

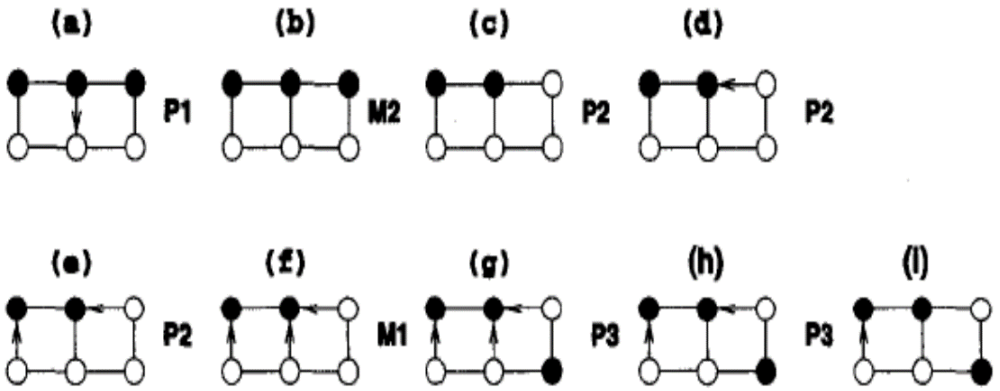

Figure 4.4: Execution of the algorithm minimal dominating set. [GHJS03b]

- every node $i \in S$ has a null pointer;
- every node $i \notin S$ having exactly one neighbor $j \in S$, points to j;
- every node $i \notin S$ having more than one neighbor $j \in S$, has a null pointer.

LEMMA 5: If at time t, S_t is not a minimal dominating set, then the system is not stable.

LEMMA 6: If a node uses $R1$, it will never make another membership move.

LEMMA 7: node can make at most two membership moves.

LEMMA 8: There can be at most n consecutive pointer moves.

LEMMA 9: The system can make at most $(2n + 1)n$ moves.

All proofs for this lemmas are in [GHJS03b].

THEOREM 2: Algorithm 4.2 produces a minimal dominating set and stabilizes in $O(n^n)$ moves.

THEOREM 3: The algorithm 4.2 is stable if

- S_t is a minimal dominating set;
- every private neighbor outside S_t points to its unique neighbor in S_t; and
- all other nodes have null pointers.

4.3.3 A Self-Stabilizing Distributed Algorithm for Minimal Total Domination [GHJS03a]

This algorithm requires that every node have a unique ID. Goddard and al. have used i interchangeably to denote a node, and the node's ID. they assumed there is a total ordering on the IDs. Each node i has two variables: *a pointer* $p(i)$ (which may be null) and *a boolean* $flagx(i)$. If $p(i) = j$ then they said that $i\ pointsto\ j$. At any given time, they denote with D the current set of nodes i with $x(i) = true$.

Definition 1: For a node i, it was defined $m(i)$ as its neighbor having the smallest ID.

Definition 2: they have defined the pointer expression $q(i)$ as follows:

$$q(i) = \begin{cases} m(i) & if N(i) \cap D = \phi \\ j & if N(i) \cap D = j \\ null & if N(i) \cap D \geq 2 \end{cases}$$

Note that the value $q(i)$ can be computed by i (i.e., it uses only local information).

definition 3: The boolean condition was defined by $y(i)$ to be true if and only if some neighbor of i points to it.

The algorithm consists of one rule; thus, a node i is privileged if $x(i) \neq y(i)$ or $p(i) \neq q(i)$. If it executes, then it sets $x(i) = y(i)$ or $p(i) = q(i)$.

The algorithm achieves the lemmas proofed in [GHJS03a]:

LEMMA 10: If the algorithm stabilizes, then D is a minimal total dominating set.

To proof this lemma, the authors claim that D is a total dominating set, by contradiction, that some node i is not totally dominated (that is, has no neighbor in D). Then $N(i) \cap D = \phi$. Since the system is stable, $p(i) = q(i) = m(i)$ and $m(i) \notin D$. But this implies $y(m(i)) = true$ and $x(m(i)) = false$, and so node $m(i)$ is privileged, a contradiction. Thus D is a **total dominating set**.

After that, they claim that D is minimal. For suppose there is some $j \in D$ for which $D - j$ is a total dominating set. Since $j \in D$, or $x(j) = true$, there is some vertex $i \in N(j)$ for which $p(i) = j$. But since $p(i) = q(i)$, node j must be a unique neighbor of i with membership in D. Thus the removal of j will leave i undominated.

Goddard and al. said that node i invites node j if, at some time t, node i has no neighbor in D and then executes the rule, causing $p(i) = m(i) = j$. For a node to join D, it must either be pointed to from an initial erroneous state or be invited. Then this shows that the algorithm stabilizes.

LEMMA 11: Let i be a node and suppose that between time t and t', there is no in-move a move is an in-move if it causes $x(i)$ to become $true$, thereby causing a node i to enter D by any node $k > i$. Then during this time interval node i can make at most two in-moves.

THEOREM 4: The self-stabilizing distributed algorithm for minimal total domination always stabilizes, and finds a minimal total dominating set.

Proof: It suffices to show that every node makes only a finite number of in-moves. By Lemma 11, node n, which has largest ID, makes at most two in-moves. During each of the three time intervals, when node n is not making an in-move, using Lemma 11 again, node $n-1$ makes at most two in-moves. It is easy to show this argument can be repeated, showing that each node can make only finitely many in-moves during the intervals in which larger nodes are inactive.

4.3.3.1 Minimal Extended Domination [GHJS03a]

The previous algorithm was generalized by Goddard and al. to obtain algorithms for other domination problems. We have a dominating set is a set in which, for all i,

$$|N[i] \cap D| \geq 1$$

and a total dominating set satisfies

$$|N(i) \cap D| \geq 1$$

Now, for the algorithm, each node has a set of pointers, denoted $p(i)$, whose cardinality is bounded by $t(i)$; we allow $p(i)$ to contain i. Each node also has a boolean flag $x(i)$. As before, $x(i)$ should be true if and only if some node points to i, and also as before, D will denote the set of nodes with true flags at any point in time. At a given time, assume $|D \cap N(i)| = k \leq t(i)$. Then since $t(i) \leq |N(i)|$ there are at least $t(i) - k$ members in $N(i) - D$. Let M_i denote the unique set of those $t(i) - k$ nodes in $N(i) - D$ having smallest $ID's$. Note this set depends on D. The pointers are defined as:

$$Q(i) = \begin{cases} (D \cap N(i) \cap) \cup M_i & if |N(i) \cap D| - k \leq t(i) \\ \phi & if |N(i) \cap D| > t(i) \end{cases}$$

As before, Goddard and al. defined the boolean condition $y(i)$ to be true if and only if some neighbor of i points to it. The algorithm consists of one rule. Thus, a node i is privileged if $x(i) \neq y(i)$ or $p(i) \neq Q(i)$. If it executes, then it sets $x(i) = y(i)$ and $p(i) = Q(i)$. It is easy to see that this algorithm reduces to previous algorithm.

Input: n: Algorithm MTDS: Minimal Total Dominating Set (Expression model),
Begin

Input: $\varepsilon = NBS :: |x \in (v) : x.state = IN|$,
R1: $v.state = OUT \wedge \exists u \in (v) : u.NBS < 1$
$\rightarrow v.state = IN$
R2: $v.state = IN \wedge v.NBS \geq 1 \wedge (\forall u \in (v) : u.NBS \geq 2)$
$\rightarrow v.state = OUT$

End.

Figure 4.5: Algorithm MTDS: Minimal Total Dominating Set.

4.3.4 self-stabilizing algorithms for minimal total k-dominating [BYK14]

4.3.4.1 Minimal total dominating set

A total dominating set S is called minimal when no proper subset of S is a total dominating set of the graph G. Belhoul and al. have used the algorithm described in the previous section avoid the use of pointers, and expand node knowledge by aggregating the states of its distance-two neighbors using the expression model. this way leads to efficient algorithms. they first proposed a linear self-stabilizing algorithm for finding a minimal total dominating set, that they call MTDS, in the expression model under the unfair central scheduler. Then, they removed this assumption to have an algorithm that converges in $O(mn)$ under the unfair distributed scheduler.

[BYK14] proofed the coming lemmas:

LEMMA 12: MTDS is a silent algorithm and when no node is enabled for MTDS in the expression model, then S forms a minimal total dominating set of G, that proofs the correctness.

LEMMA 13: Once a node dominated, it remains dominated.

LEMMA 14: If a node leaves S, it will never move again.

LEMMA 15: Algorithm MTDS stabilizes after at most $2n$ moves in the expression model using the unfair central scheduler.

THEOREM 5: By Lemma 14, a node that executes $R2$ will never move again, so a node moves at most twice (the longest execution sequence of any node is $R1R2$). Thus, Algorithm MTDS stabilizes after at most $2n$ moves.

4.3.4.2 Total k-dominating set [BYK14]

Definition 4: Let $G = (V, E)$ be a graph and k be a positive integer $(k \geq 1)$. A total k-dominating set S (also called in literature, **k-tuple** total dominating set) is defined to be a subset of V such that for each $v \in V$, we have $|N(v) \cap S| \geq k$.

The condition on k for the existence of a total k-dominating set in a graph G is $k \leq \delta$, where δ denotes the minimum degree of G. Notice that a total 1-dominating set is the usual total dominating set. A total k-dominating set S is called minimal when no proper subset of S is a total k-dominating set of the graph G. The algorithm that constructs a minimal total k-dominating set, given as $MTDS_k$, using the expression model under the unfair central scheduler uses the same variable $state$as Algorithm MTDS. However, Belhoul and al. have to redefine the expression of NbS , denoted by NbS_k, according to the definition of the total k-dominating set.

The set of expressions ε_k of $MTDS_k$ for a given node v will be:

$$\varepsilon_k = NbS_k :: |x \in N(v) : x.state = IN| - k + 1$$

Rules of Algorithm $MTDS_k$ are obtained by replacing the expression NbS in rules of Algorithm $MTDS$ by NbS_k. Note that a node v is said to be k-dominated when $v.NbS_k \geq 1$.

4.4 Self-stabilizing Algorithm for Minimal α-Dominating Set

4.4.1 α-domination

The α-domination concept (without using self-stabilization concept) has been studied for the first time by [DHLM00]. they introduced the concept of α-domination as:

For any α with $0 < \alpha \leq 1$ and a set $S \subseteq V$, we say that S is α-dominating if for all $v \in V - S$, $|N(v) \cap S| \geq \alpha|N(v)|$. The size of a smallest such S is called the *α-domination number* and is denoted by $\gamma_\alpha(G)$.

Thus the Woolbright number [Woo95] of a graph G is $\gamma_{1/2}(G)$. The size of a largest minimal such set S is called the *upper α-domination* number and is denoted by $\Gamma_\alpha(G)$.

They were be interested in studying the relationship between the α-domination parameter and other domination-related parameters. One can easily find that more than 80 types of domination and domination-related parameters have appeared in the literature [HHS98].

Although this new parameter adds to the list in a unique way, it is related to other types of domination.

Other results on α-domination are given in [DRV04]. Let $G = (V, E)$ be a connected graph where V is the set of nodes and E is the set of edges. We say that $S \subseteq V$ is α-dominating if for all $v \in V - S, \frac{|N(v) \cap S|}{|N(v)|} \geq \alpha$, where $0 < \alpha \leq 1$ and $N(v)$ is the set of v neighbors i.e. $N(v) = \{u | vu \in E\}$. Besides the particular cases of α-domination introduced in the literature, we present a new parametric called (α, β)-dominating set. An (α, β)-dominating set is a subset of nodes S such that: S is α-dominating set and for each node v in S, the rate of neighbors of v inside S is greater than β, where $0 \leq \beta \leq 1$.

4.4.2 Related work of α-domination on self-stabilization

In self-stabilizing paradigm, few algorithms are proposed only for the particular instance of $\alpha = \frac{1}{2}$ (three works to the best of our knowledge). In these cases, authors try to find the minimal dominating set where each node (in $V - S$ or in V) is dominated by at least half ($\alpha = \frac{1}{2}$) of its neighborhood. Wang $et\ al.$ have introduced the *positive influence dominating set* [FEK09]. A self-stabilizing algorithm known as MPIDS is presented for this parameter in [WWTZ13]. We call $S \subseteq V$ a *positive influence dominating set* if each node $v \in V$ is dominated by at least $\lceil \frac{|N(v)|}{2} \rceil$ (that is, v has at least $\lceil \frac{|N(v)|}{2} \rceil$ neighbors in S).

This algorithm can be considered as total $\frac{1}{2}$-domination because the condition of $\frac{1}{2}$-domination must be respected by all the nodes. Positive influence domination has applications in social networks where this parameter is used in [AKL18, WWTZ13, DWS14a] in order to influence the opinion of the users and individual behaviors in social networks. For example, in a social network with smoking problem, smokers could be exposed to a possible conversion to abstain due to the domination of their friends.

Simultaneously, Yahiaoui *et al.* have proposed a self-stabilizing algorithm for *minimal global powerful alliance set* called MGPA [YBHK13] which has the same basic concept of MPIDS. A subset S is said *global powerful alliance set* if for each node $v \in V$, the majority of the neighbors of v are in S, that is, $|N[v] \cap S| \geq |N[v] \cap (V - S)|$, where $N[v] = N(v) \cup \{v\}$. Like MPIDS, MGPA can be also considered as total $\frac{1}{2}$-domination. Note that the global alliance concept is presented by Hedetniemi which includes the offensive and the defensive alliances. In the offensive alliance, every node in $V - S$ must be $\frac{1}{2}$-dominated. While for the defensive alliance, every node in S has to be $\frac{1}{2}$-dominated. S is said global powerful alliance if S is offensive and defensive.

Hedetniemi *et al.* have presented a collection of self-stabilizing algorithms to find an *unfriendly partition* into two dominating sets R and B [HHKM13]. A bipartition $\{R, B\}$ is called *unfriendly partition* if for every node $v \in R$, most of neighbors of v are in B and

for every node $u \in B$, most of neighbors of u are in R. Obviously, every node of R is $\frac{1}{2}$-dominated by B and every node of B is $\frac{1}{2}$-dominated by R.

In 2011, Dourado $et.\ al.$ have introduced a general variant of alliance called (f, g)-alliance problem, where f and g are functions that depend on nodes degree. A set S is called (f, g)-alliance if every node out S has a number of neighbors in S greater than or equal f. And every node in S, has a number of neighbors in S greater than or equal g. [DPRS11]

The algorithm $MA(f, g)$ is safely converging in the sense that, starting from any configuration, it first converges to a (not necessarily minimal) (f, g)-alliance in at most four rounds, and then continues to converge to a minimal (f, g)-alliance in at most $5n + 4$ additional rounds, where n is the size of the network.

Carrier $et.\ al.$ have presented in 2013 a distributed self-stabilizing algorithm for computing (f, g)-alliance set that converges in $O(\Delta^3 n)$ moves, where Δ is the maximal degree in the graph. However, the later algorithm works only under the hypothesis that $f \geq g$. [CDD$^+$15] Algorithm is written in the shared memory model, and is proven assuming an unfair (distributed) daemon, the strongest daemon of this model. Authors have conjectured that there is a self-stabilizing algorithm (f, g)-alliance when $f < g$.

The major drawback of this algorithm, is the integer values of f and g. Before, launching the self-stabilizing, the user must to have a previous knowledge on the graph degrees in order to determine values of f and g. For example, to determine g for a node v, it must be set at most $|N(v)|$.

In this work, we use values that express rate (known as α and β) rather than using integer values (f and g) that depends on the nodes degree. In this case, there is no need to have a previous knowledge on the distribution of the nodes degree.

4.5 Model and terminology

Generally, networks or distributed systems are represented by simple undirected graphs. Let $G = (V, E)$ be a connected graph where V is the set of nodes and E is the set of edges. For a node $v \in V$, the open neighborhood of v is defined as $N(v) = \{u \in V : vu \in E\}$, where the degree of v is $d(v) = |N(v)|$. $N[v] = N(v) \cup v$ denotes the closed neighborhood of v. In this thesis, we use $neighborhood$ to indicate the open neighborhood. In the following table, we provide description for notations used in the rest of the chapter.

Definition 5: A subset $S \subseteq V$ is a dominating set if for every node $v \in V - S$, there exists a node $u \in S$ such that v is adjacent to u [HHS98].

Notation	Description
V	set of nodes (vertices)
E	set of edges
$G(V, E)$	graph with a set of nodes V and a set of edges E
v, u, w	used generally to represent a node from V
vu	an edge of E from v to u
n	number of nodes i.e. $\|V\|$
m	number of edges i.e. $\|E\|$
S	a subset of nodes, generally S is defined as a dominating set
$N(v)$	set of neighbors of v
$N_S(v)$	set of neighbors of v which are in S i.e. $N(v) \cap S$
$N_{V-S}(v)$	set of neighbors of v which are out S
$d(v)$	degree of v that is equal to number of neighbors $\|N(v)\|$
Δ	maximal degree in a graph
α	threshold in $]0, 1]$
β	threshold in $[0, 1]$
α-dominated	used for nodes out the dominating set, a node $v \in V - S$ is α-dominated if $\frac{\|N_S(v)\|}{\|N(v)\|} \geq \alpha$
β-dominated	used for nodes in the dominating set, a node $v \in S$ is β-dominated if $\frac{\|N_S(v)\|}{\|N(v)\|} \geq \beta$
$v.state$	boolean variable of v that describes whether v is in S ($v.state = In$) or out S ($v.state = Out$)
$v.exp$	mathematical expression of v used to calculate rate of neighbors in S i.e. $\frac{\|N_S(v)\|}{\|N(v)\|}$

Table 4.1: Table of notations

The set $N_S(v)$ defines the neighbors of v in S i.e. $N_S(v) = \{u \in S : vu \in E\}$ and $N_{V-S}(v)$ represents neighbors of v in $V - S$. Consequently, $N(v) = N_S(v) \cup N_{V-S}(v)$.

An algorithm is self-stabilizing if it will be able to reach (during a finite time $convergence$) a global correct configuration called $legitimate$ and still in the legitimate state ($closure$) after it has started from an unknown configuration.

To show that an algorithm is self-stabilizing, it is sufficient to prove its $closure$ for the legitimate configuration and its $convergence$ to achieve the desired configuration in a finite time. So, a self-stabilizing algorithm guarantees to converge to the legitimate configuration even if there is any possible transient faults. Also, an algorithm is called $silent$ if in a legitimate state, there is no enabled nodes. Obviously, if an algorithm is silent, the closure is trivially satisfied.

In a *uniform* self-stabilizing system, all the nodes execute the same collection of rules having the form **if** $guard$ **then** $statement$ (written as: $guard \longrightarrow statement$). Nodes have also the same local variables that describe their $state$. The *guard* is a (or a collection of) boolean expression. Once a guard of any node is true, the corresponding statement must be executed (an action on the node's state).

Thus, the $state$ of every node is updated (modified or not) by the node itself using at least one of its own rules.

Each node has a partial view of the distributed system (i.e. $guard$ which consists of boolean expressions) on (1) its state and the states of its neighbors (called distance-one model) or (2) its state and the states of its neighbors and the states of the neighbors of its neighbors (called distance-two model). A rule is said $enabled$ if the guard is evaluated to be true. A node will be enabled if at least one of its rules is enabled. Executing the statement of the enabled rule by the node is called a $move$. A move allows updating the state (local variables) of the node in order to be in harmony with its neighborhood.

4.5.1 Execution model

The execution of self-stabilizing algorithms is managed by a daemon (scheduler) that selects nodes to move from a configuration to another configuration. Two types of daemons are widely used in self-stabilization literature: central and distributed daemons. In the central daemons, one enabled node is selected among all the enabled nodes to be moved. However, in the distributed daemons, a subset of nodes are selected among the set of enabled nodes to make a move simultaneously.

A particular case is distinguished for distributed daemons i.e. the *synchronous* daemon where all the enabled nodes are selected to move simultaneously. Indeed, $unfair\ distributed\ daemon$ is the most used scheduler in self-stabilizing literature, where, in any subset of the enabled nodes can make their moves simultaneously. A $fair$ daemon is a scheduler that selects the same enabled node continuously between configurations transition. Otherwise, the daemon is $unfair$ where it can delay the node move if there are other enabled nodes which allows to guarantee the convergence to the global legitimate configuration. A detailed taxonomy of the daemons variants can be found in [DT11].

4.5.2 Transformers

Generally, it is easy to prove the stabilization of an algorithm working under hypotheses like central daemon and expression distance-2 model. However, algorithms working under

distributed daemon and distance-one model are more difficult to prove, although they are more suitable for real applications.

A common approach, known in literature [Tur12, GS13], allows converting a self-stabilizing algorithm A which operates under a given hypotheses to a new self-stabilizing algorithm A^T, such that A^T operates under other hypotheses. This transformation guarantees that the two algorithms obtain the same legitimate configuration. Different kinds of transformers can be found in literature like *distance* transformers and *daemon* transformers. The transformation process causes generally an overhead of the algorithm complexity. In this thesis, we use the transformer D of Turau [Tur12] who can convert a self-stabilizing algorithm A (where A is silent, executes under expression distance-2 model and uses an unfair central daemon) to a self-stabilizing algorithm A^D (The reached algorithm A^D is silent, operates under distance-1 model and uses an unfair distributed daemon).

Transformers are designed in order to respect basis of self-stabilization. For example, in the case of distributed daemon, it is not allowed for neighbors to make simultaneously a move at the same time. This is achieved generally by using identifiers for nodes by the transformer. In the same round, the node who has the higher identifier could be enabled (and then moves) among their neighbors.

4.6 Self-stabilizing algorithm for minimal α-dominating set

In this section, we present a self-stabilizing algorithm called α-MDS for finding minimal α-dominating set.

First, we give definition of α dominating set:

Definition 6: Let S be a subset of V and $0 < \alpha \leq 1$. S is called *α-dominating set* if for every node $v \in V - S$, $\frac{|N_S(v)|}{|N(v)|} \geq \alpha$. S is minimal if no proper subset of S is α-dominating set. Every node in $V - S$ is called α-dominated.

Figure 4.6 illustrates the proposed self-stabilizing algorithm, where each node v maintains a local variable $state$ and two expressions $exp1$ and $exp2$. The value of $state$ can be In or Out. It is clear that $state$ is used to express whether any node belongs to α-dominating set or not. Once the legitimate configuration is achieved, the α-dominating set is defined as $S = \{v \in V : v.state = In\}$. The expression $exp1$ is used to check whether every node out of S is α-dominated.

However, $exp2$ is used when a node v wants to leave S. The later expression aims to check whether neighbors of v out of S still always α-dominated after v leaves S. Rule $R1$ allows every node out of S which is not α-dominated to move from Out to In. Consequently, $R1$ ensures that every node of $V - S$ must be α-dominated. $R2$ is used to verify

Each node v checks and executes (with infinite loop) the following

Expressions:

exp1:: $\frac{|N_S(v)|}{|N(v)|}$

exp2:: $\frac{|N_S(v)|-1}{|N(v)|} = exp1 - \frac{1}{|N(v)|}$

Rules:

R1: $v.state = Out \wedge v.exp1 < \alpha \longrightarrow v.state = In$

R2: $v.state = In \wedge (v.exp1 \geq \alpha) \wedge (\forall w \in N_{V-S}(v) : w.exp2 \geq \alpha) \longrightarrow v.state = Out$

Figure 4.6: α-MDS self-stabilizing algorithm

the minimality of S. Every node in S that can leave S without affecting the constraint α-domination of its neighbors in $V - S$ and still itself α-dominated, will leave S because it preserves the legitimate configuration of its neighborhood. Observe that $exp2$ is used when a node v wants to leave S, all the neighbors $w \in N_{V-S}(v)$ must still respect the α-domination condition after the leaving of v. If, we deduce that after v moves from In to Out, there will be at least a neighbor $w \in N_{V-S}(v)$ such that w will become not α-dominated, thus, v cannot move out of S.

4.6.1 Closure

LEMMA 16: Once all the nodes are not enabled, the set S is a minimal α-dominating set.

PROOF: We prove that : (a) every node v of $V - S$ is α-dominated and (b) S is minimal.

(a) For every node v out of S, $v.exp1$ must be $\geq \alpha$ because $R1$ is not enabled. Hence, each node out of S is α-dominated.

(b) We prove the minimality by contradiction. Suppose that all nodes are not enabled and there exists a subset S' of S which is also α-dominating set. Let v e a node in S such that $S' = S - \{v\}$. Thus, $\frac{|N_{S'}(v)|}{|N(v)|} \geq \alpha$ (by definition) which implies that $\frac{|N_S(v)|}{|N(v)|} \geq \alpha$ because $N_{S'}(v) = N_S(v)$. Since all the nodes of S are not enabled ($R2$ is not enabled for v) and $v.exp1 \geq \alpha$, there exists a node $w \in N_{V-S}(v)$ such that $w.exp2 < \alpha$, so $\frac{|N_S(w)|-1}{|N(w)|} < \alpha$. After moving v from S to $V - S$, the number of neighbors of w having state In will decrease by one i.e. $|N_{S'}(w)| = |N_S(w)| - 1$. Thus $\frac{|N_S(w)|-1}{|N(w)|} < \alpha$ becomes $\frac{|N_{S'}(w)|}{|N(w)|} < \alpha$. This is a contradiction with the definition that every node out of the dominating set must be α-dominated.

4.6.2 Convergence and complexity analysis

LEMMA 17: If any node $w \in (V - S)$ has $w.exp1 \geq \alpha$, the value $w.exp1$ still greater than or equal α and cannot be down less than α.

PROOF: Let w be a node from $V - S$ such that $w.exp1 \geq \alpha$. The value $w.exp1$ can be decrease by one way which is: if any neighbor v of w changes its state from In to Out. However, R2 imposes that when v moves from In to Out , all its neighbors in $V - S$ including w must have $exp2 \geq \alpha$ i.e. $\frac{|N_S(w)|-1}{|N(w)|} \geq \alpha$ [1]. Suppose S_2 is the new dominating set after v leaves S. Thus, $|N_{S_2}(w)| = |N_S(w) - 1|$. So, $\frac{|N_S(w)|-1}{|N(w)|} \geq \alpha$ becomes $\frac{|N_{S_2}(w)|}{|N(w)|} \geq \alpha$. Hence, $w.exp1$ remains $\geq \alpha$.

LEMMA 18: Once a node leaves S, it cannot reach S again.

PROOF: Since any node leaves S with $exp1 \geq \alpha$, R1 cannot be enabled again according Lemma 17.

LEMMA 19: Every node executes at most R1 then R2 which allows algorithm 1 to terminate in the worst case at $2n$ moves under the expression distance-2 model using unfair central daemon.

PROOF: It follows from Lemma 18.

THEOREM 6: α-MDS is a silent self-stabilizing algorithm giving minimal α-dominating set in finite time not exceeding $O(n)$ moves under expression distance-2 model using unfair central daemon.

PROOF: Lemma 16 gives the correctness and the closure because when the algorithm is silent (all the nodes are disabled), it provides a minimal α-dominating set. Lemma 19 shows the convergence where the algorithm reaches the minimal α-dominating set in $O(n)$. Therefore, the proof follows from Lemma 16 and Lemma 19.

After proving the stabilization of algorithm α-MDS under the central daemon and expression distance-2 model, we use the transformer D proposed by [Tur12] that gives another self-stabilizing algorithm α-MDS^D. This later is executable under distributed daemon and distance-one model.

THEOREM 7: α-MDS^D gives a minimal α-dominating set and stabilizes in $O(nm)$ moves in the distance-one model under unfair distributed daemon .

PROOF: Using the transformer D of Turau. According the Theorem 18 of Turau [Tur12], if there is a silent self-stabilizing algorithm A in the expression model that stabilizes after $O(A(G))$ moves under the unfair central daemon. Then, the transformed algorithm A^D is a silent algorithm that stabilizes after $O(mA(G))$ moves under the unfair distributed daemon

[1] If at least one of v neighbors: $w' \in S'$ has $w'.exp2 < \alpha$, then R2 cannot be enabled and v remains in S, thus $w.exp1$ still has $\geq \alpha$.

in the distance-one model. Thus, the proof follows from Theorem 18 of [Tur12] , where m is the number of edges.

Definition 7: Let $G = (V, E)$ be a graph with no isolated nodes. The subset S of V is called a *k-defensive dominating set* if for every node v in S, we have $\frac{|N_S[v]|}{|N[v]|} \geq k$, where $k \in]0, 1[$

4.7 Minimal (α, β) dominating set

In the α-domination problem presented by Dunbar et al. [DHLM00], only nodes out of the dominating set are imposed to be α-dominated. However, it is possible to apply this parametric on all the nodes (who are In and Out) like the total domination and the global powerful alliance. In the following, we introduce the new domination variant called (α, β)-domination.

Definition 8: Let S be a subset of V, $0 < \alpha \leq 1$ and $0 \leq \beta \leq 1$. S is called (α, β)-*dominating set* if (1) for every node $v \in V - S$, $\frac{|N_S(v)|}{|N(v)|} \geq \alpha$ and (2) for every node $v' \in S$, $\frac{|N_S(v')|}{|N(v')|} \geq \beta$. S is minimal if no proper subset of S is (α, β)-dominating set.

Observe that :

- $(\alpha, 0)$-domination is α-domination problem.
- $(\frac{1}{\Delta}, 0)$-domination is the habitual problem of domination, where Δ is the maximal degree in the graph.
- $(\frac{1}{2}, 0)$-domination is the offensive alliance.
- $(\frac{1}{2}, \frac{1}{2})$-domination is the global powerful alliance.
- For $(\alpha, 1)$-dominating set, $S = V$ and $V - S = \emptyset$.
- α could be equal to zero although it has been imposed $\in]0, 1]$ in the literature. In this case when $\alpha = 0$, the minimal dominating set will be an empty set.

Note that the (f, g)-alliance is the same as the (α, β)-domination problem when $\alpha = \frac{f}{|N(v)|}$ and $\beta = \frac{g}{|N(v)|}$. In (f, g)-alliance, the dominating set is calculated based on the *number* of neighbors that must be greater than or equal the *integer* thresholds f and g. However, in the (α, β)- domination, the dominating set is computed based on the *rate* of neighbors that must be greater than or equal the $real$ thresholds α and β, such that $0 < \alpha \leq 1$, $0 \leq \beta \leq 1$.

Values of f and g are depending on graph structure. The user must has a previous knowledge on the degree distribution of the graph. For example to determine values of f and g for

Each node v checks and executes (infinite loop) the following

Expressions:

exp1:: $\frac{|N_S(v)|}{|N(v)|}$

exp2:: $\frac{|N_S(v)|-1}{|N(v)|} = exp1 - \frac{1}{|N(v)|}$

Rules:

R1: $v.state = Out \wedge (v.exp1 < \alpha \vee \exists u \in N_S(v) : u.exp1 < \beta) \longrightarrow v.state = In$

R2: $v.state = In \wedge (v.exp1 \geq \alpha) \wedge (\forall w_{out} \in N_{V-S}(v) : w_{out}.exp2 \geq \alpha)$
$\wedge(\forall w_{in} \in N_S(v) : w_{in}.exp2 \geq \beta) \longrightarrow v.state = Out$

Figure 4.7: α, β-MDS self-stabilizing algorithm

any node v, we need to know its degree in order to decide whether v has a sufficient number of neighbors that overtakes the threshold. However, for the (α, β)-domination problem, it is sufficient to define α and β without any previous knowledge which facilitates computing dominating set regarding (f, g)-alliance.

4.7.1 Self-stabilizing algorithm for minimal (α, β)-dominating set

Figure 4.7 shows the proposed self-stabilizing algorithm for computing minimal (α, β)-dominating set called (α, β)-MDS. Variable $state$ and expressions $exp1$, $exp2$ have been used before in algorithm 4.5, are also implemented in the current self-stabilizing algorithm. The $state$ variable indicates whether a node is In or Out. Expression $exp1$ allows testing whether every node in S (resp. out S) is β-dominated (resp. α-dominated). While, $exp2$ checks that after a node v leaves S, their neighbors still always respecting conditions of α and β domination. $R1$ moves a node from Out to In, if it is not α-dominated or has a neighbor in S not β dominated. $R2$ moves a node v from In to Out with guarantees that new configuration in the neighborhood of v still respect α and β domination after v leaves S.

4.7.2 Closure

LEMMA 20: When all the nodes are disabled, S is a minimal (α, β)-dominating set.

PROOF:

(a) We show first that S is (α, β)-dominating set. When all the nodes are disabled, $R1$ is disabled. Thus, for every node v out of S, $v.exp1 \geq \alpha$ and $\forall u \in N_S(v) : v.exp1 \geq \beta$.

Therefore, each node out of S is α-dominated and each node $u \in S$ who has a neighbor in $V - S$ is β-dominated. As for the remainder nodes $u \in S$ having no neighbors in $V - S$, they are β-dominated because $u.exp1 = 1$ which is always greater than or equal β.

(b) We prove that S is minimal by contradiction. Suppose that S is (α, β)-dominating set and there is no enabled node in S. Let $S' = S - \{v\}$ be (α, β)-dominating set. So, this means that v is α-dominated regarding S' (and even α-dominated for S because $\frac{|N_S(v)|}{|N(v)|} = \frac{|N_{S'}(v)|}{|N(v)|}$) , and for every neighbor of v we have $\frac{|N_{S'}(w_{in})|}{|N(w_{in})|} \geq \beta$ when $w_{in} \in N_{S'}(v)$ and $\frac{|N_{S'-V}(w_{out})|}{|N(w_{out})|} \geq \alpha$ when $w_{out} \in N_{V-S}(v)$. Since, $\{v\} = S - S'$, it is easy to deduce for all w_{in} and w_{out} neighbors of v that $N_{S'}(w_{in}) = N_S(w_{in}) - 1$ and $N_{S'}(w_{out}) = N_S(w_{out}) - 1$. Therefore, for all $w_{in} \in N_S(v)$, $\frac{|N_S(w_{in})|-1}{|N(w_{in})|} \geq \beta$ and for all $w_{out} \in N_S(v)$, $\frac{|N_S(w_{out})|-1}{|N(w_{out})|} \geq \alpha$. This is contradiction, because $R2$ is enabled for v.

4.7.3 Convergence and complexity analysis

The convergence proof is the same as one of the α-domination has been showed. Once a node v leaves the set S and becomes has $v.exp1 \geq \alpha$. It still has $v.exp1 \geq \alpha$ and it cannot then come back to S. So, in the worst case, every node can make only two moves: $enter$ then $leave$. Thus, for n nodes of the graph, the algorithm achieves the legitimate configuration in $2n$ moves using a central daemon.

LEMMA 21: When a node v out of S reaches this predicate: $v.exp1 \geq \alpha \wedge \forall u \in N_S(v), u.exp1 \geq \beta$ (v is α-dominated and all its neighbors in S are β-dominated), it still remain in this predicate.

PROOF:

(a) How v still α-dominated?

Like is shown above in the proof of α-domination algorithm. Once a node v has $\frac{|N_S(v)|}{|N(v)|} \geq \alpha$, its expression $\frac{|N_S(v)|}{|N(v)|}$ can decrease by one way i.e. if a neighbor u of v leaves S. However, when u decides to leaves S, it has to check that $\forall w_{out} \in N_{V-S}(u) : w_{out}.exp2 \geq \alpha$, which means that $v.exp2 \geq \alpha$ before u leaves S i.e. $\frac{|N_S(v)-1|}{|N(v)|} \geq \alpha$. Suppose that $S' = S - u$, since $|N_{S'}| = |N_S(v) - 1|$, v remains α-dominated regarding S' because $\frac{|N'_S(v)|}{|N(v)|} \geq \alpha$.

(b) How neighbors of v in S i.e. $u \in N_S(v)$ still all β-dominated ?

The expression $\frac{|N_S(u)|}{N(u)}$ of every node $u \in S$ which is β-dominated can decrease if a neighbor z of u leaves S. When z decides to leave S, it checks the condition of $R2$. In other word, $u.exp2$ must be greater than or equal β. After z leaves S, u still β-dominated under the set S' because $\frac{|N_S(u)-1|}{|N(u)|} = \frac{|N'_S(u)|}{|N(u)|} \geq \beta$.

LEMMA 22: (α, β)-MDS terminates in the worst case in $2n$ moves under the expression distance-2 model using unfair central daemon.

PROOF: Lemma 21 shows that each node moves at most twice ($R1$ then $R2$). Therefore, for n nodes and under the central/sequential daemon, the convergence of (α, β)-MDS is bounded by $2n$ moves.

THEOREM 8: (α, β)-MDS is a silent self-stabilizing algorithm that calculates the minimal (α, β)-dominating set in $O(n)$ moves under the expression distance-2 model using unfair central daemon.

PROOF: Lemma 20 illustrates that (α, β)-MDS computes the minimal (α, β)-dominating set when all the nodes are disabled and no other node can move. Then, correctness and closure are reached as a silent algorithm. Lemma 22 shows the convergence of (α, β)-MDS where the minimal (α, β)-dominating set is reached in $O(n)$ moves. Thus, the proof follows from Lemma 20 and Lemma 22.

THEOREM 9: (α, β)-MDS^D is self-stabilizing algorithm that calculates minimal (α, β)-dominating set in $O(nm)$ moves under the distance-one model using unfair distributed daemon .

PROOF: Using the transformer D of Turau [Tur12], (α, β)-MDS could be converted to a self-stabilizing algorithm (α, β)-MDS^D that works under distance-one model using a distributed daemon. The proof follows from Theorem 18 of [Tur12].

4.8 Simulation and experimental results

In this section, simulation tests are carried out to test α-MDS and (α, β)-MDS on two levels. First, we attempt to observe the behavior of α-MDS according values of α in $]0, 1]$. Secondly, we test (α, β)-MDS according values of both α and β simultaneously. It is important to mention that this is the first work implementing the expression model. We have utilized the implementation of L. Kuszner [Kus05] written in Java which represents a platform for simulating uniform self-stabilizing algorithms. The proposed algorithms α-MDS and (α, β)-MDS have been implemented and integrated. Only a central daemon is considered which is the solely daemon provided by Kuszner plateform. Arbitrary graphs (Erdos-Reiny) of 1000 nodes using different densities have been synthetically generated. We have considered the average value after launching 10 executions for each test to find the dominating set.

Figures 4.8 and 4.9 show experiments performed on α-MDS only. In this case, we try to understand the behavior of α-MDS according values of α in $]0, 1]$.

Figure 4.8 illustrates that whatever the graph density, the α-MDS size grows proportionally whith α values. However, density of graph has some impact on the cardinality of the dominating set. For high density ($= 0.9$) where graphs are close to be complete, relation is clear: $\alpha \simeq cardinality$ like an equation of a line $x = y$, where $0 < \alpha \leq 1$. Once density

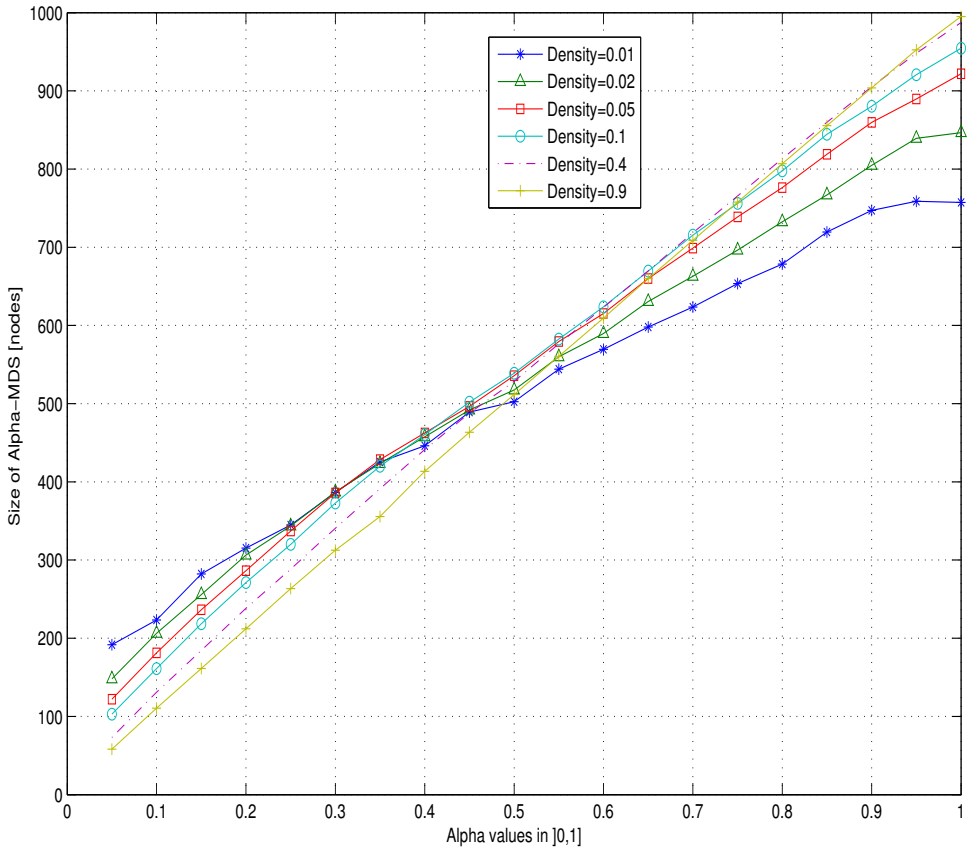

Figure 4.8: Cardinality of α-MDS according to α on graphs with 1000 nodes.

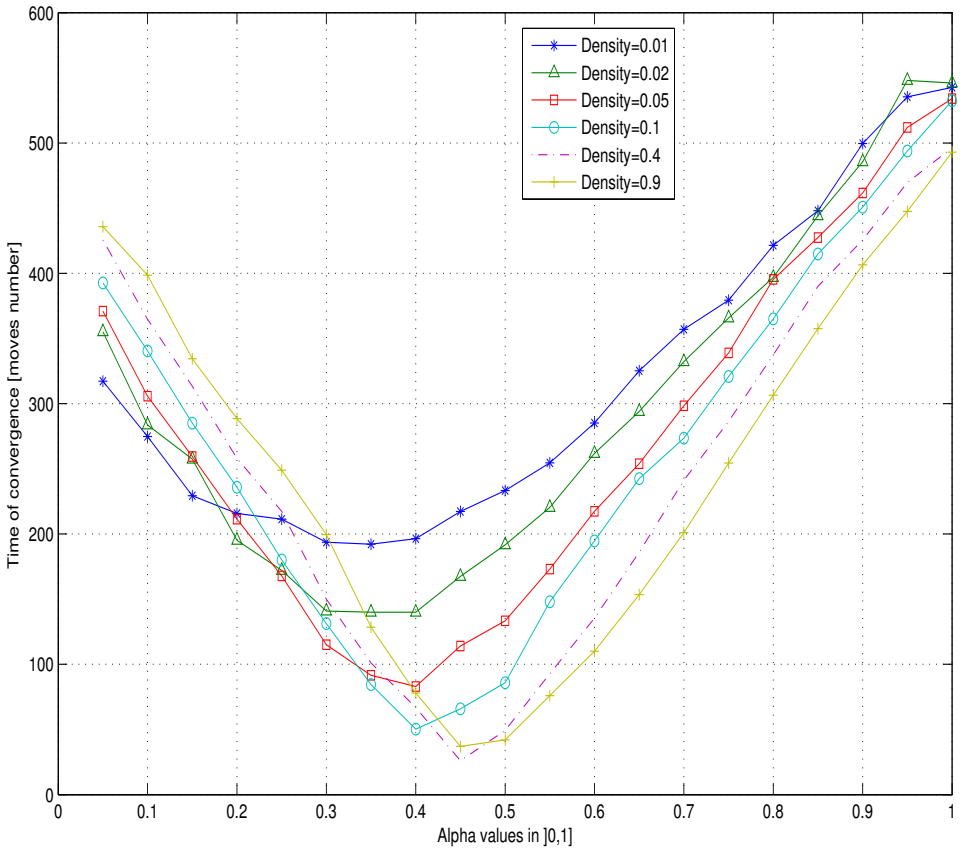

Figure 4.9: Convergence time according to α on graphs with 1000 nodes.

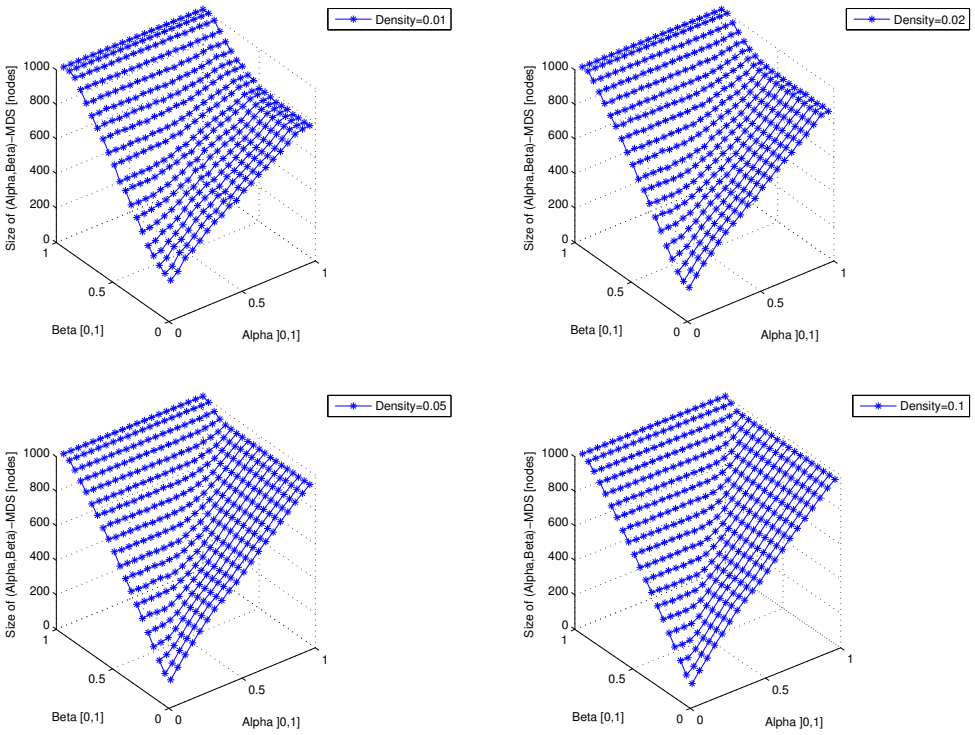

Figure 4.10: Cardinality of the minimal dominating set according α and β.

begins to be down, curves of α-MDS size starts to deviate from the line $x = y$ especially on the extremities of the interval $]0, 1]$. The worst deviation from $x = y$ is represented by the curve of the smallest density $= 0.01$ where for $\alpha = 0.05$ the cardinality of α-MDS is 20% and for $\alpha = 0.95$ the cardinality is 75%. Theses results are important from a point of view application. Reducing the value of α as possible gives a lower cardinality of the dominating set which is very practical in the reality. For example, for a given problem, if we want a small dominating set of nodes, it will be sufficient to set α as small as possible.

Figure 4.9 shows the necessary time to converge to the stable configuration according values of α. Theoretically, we have proved in section *Convergence and Complexity Analysis* that the number of moves cannot exceed $2n$ moves which is confirmed by the experiments where the number of moves is always less then $3n/5$. However, it is clear through Figure 4.9 that α-MDS needs more time (number of moves) on the extremities of $]0, 1]$ while it converges quickly in the middle of this area.

Figures 4.10 and 4.11 illustrates tests curried out according values of α and β simultaneously. In figure 4.10, the cardinality of the minimal $[\alpha, \beta)$-dominating set grows proportionally with α and β. Observe that cardinality of the dominating set reaches its maximal value (i.e. $S = V$) when $\beta = 1$. The convergence according values of α and β is illustrated by figure 4.11. Globally, the number of moves to reach the illegitimate configuration has a upper bound of 600 moves for a graph with 1000 nodes.

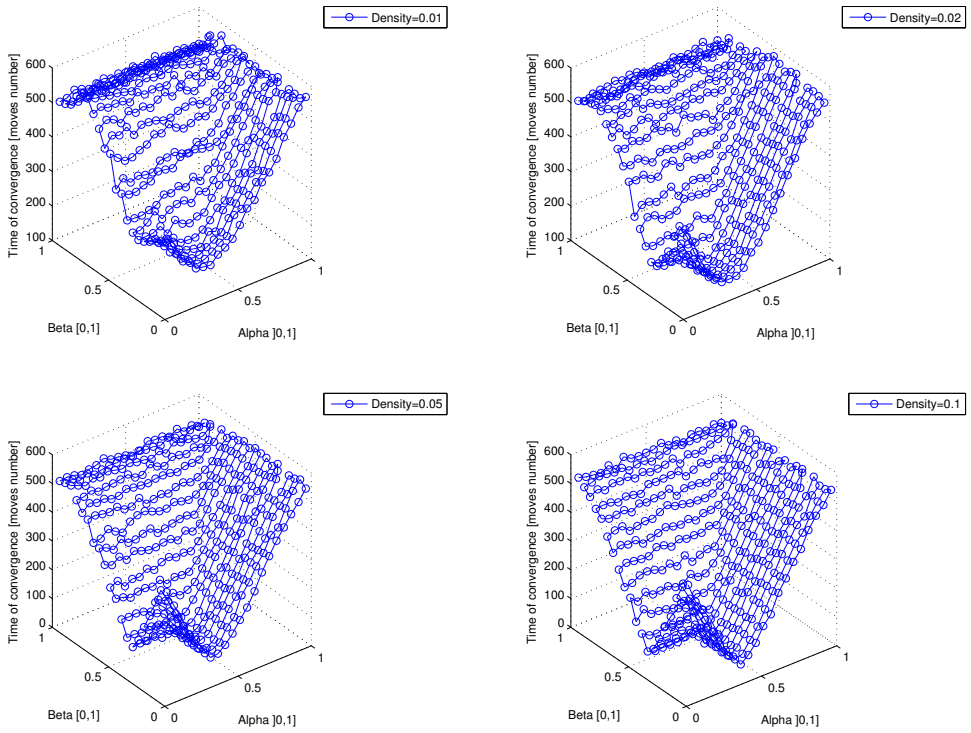

Figure 4.11: Convergence time according α and β..

4.9 Conclusion

Analysis shows that algorithm stabilizes in $2n$ moves, namely a complexity of $O(n)$ using the expression distance-2 model under central daemon. Simulations experiments illustrates the efficiency of the proposed algorithm. Note that expression distance-2 model is a variant of distance-two model. Using the transformer of [Tur12], α-MDS can be converted to α-MDS^D that converges in $O(nm)$ under distributed daemon and distance-one model.

Simulation experiments conducted under expression distance-2 model show that α-MDS outperforms other known algorithms.

For $\alpha \in]0, 1[$; for example, when α is fixed to be 0.2, each node (out of the domination set following our definition) must has at least 20% of neighbors in the dominating set. It is clear that the values $\alpha = 0$ or $\alpha = 1$ have no significance because when $\alpha = 0$ it means that all the nodes must be out of dominating set and if $\alpha = 1$ that leads to all nodes are in the dominating set.

Table 4.1 summarizes main formal results of α-MDS with other particular self-stabilizing algorithms (of $\alpha = \frac{1}{2}$) MPIDS, MGPA and Unfriendlier.

Table 4.2 presents results on both (α, β)-MDS and (f, g)-alliance.

Using the expression distance-2 model proposed by Turau, a self-stabilizing algorithm

Algorithm	Daemon	Complexity	α values
MPIDS [WWTZ13]	Central	$O(n^2)$	$\alpha = 1/2$
Unfriendlier [HHKM13]	Central	$O(n^3m)$	$\alpha = 1/2$
MGPA [YBHK13]	Distributed	$O(nm)$	$\alpha = 1/2$
α-MDS [this thesis]	Distributed	$O(nm)$	$0 < \alpha \leq 1$

Table 4.2: Self-stabilizing algorithms on one parameter α domination.

Algorithm	Daemon	Complexity (moves)	Constraints between parameters
minimal (f, g)-alliance [CDD$^+$15]	Distributed	$O(\Delta^3 n)$	Proved only for $f \geq g$, conjecture for $f < g$
(α, β)-MDS [this thesis]	Distributed	$O(nm)$	No constraint on α and β

Table 4.3: Self-stabilizing algorithms on two parameters α and β domination.

is proposed in this thesis for the new parametric (α, β)-domination called (α, β)-MDS. The algorithm is studied in both theoretical and experimental sides, where it is proved that it converges in $O(nm)$ moves using a distributed daemon. The proposed algorithm confirms the conjecture of [CDD$^+$15] that there is a self-stabilizing algorithm for (f, g)-alliance when $f < g$. The later result is achieved due to the expression distance-2 model that facilitates developing self-stabilizing algorithm for calculating minimal (α, β)-dominating set. Only a upper bound is provided in terms of $moves\ number$ that shows a convergence in $O(mn)$ moves. It will be interesting to work in the future in order to find the convergence in terms of $rounds$.

General Conclusion

Conclusion

In this thesis, we aimed to contribute in the field of fault tolerance in dynamic distributed systems and to propose a new algorithm allowing such a system to stabilize at a legitimate state, and like all work we started with a search and a state of the art to collect all the necessary information to achieve a certain objective.

Our first knowledge must be in distributed systems and their needs in daily life, today, these systems intervene in all aspects of life either in the form of services, media, giant library of information, ... or in different fields presented by sensor networks such as intelligent transport, military service, forest protection systems, industry, Internet of things, ... All these types of systems present a certain level of dynamicity and reliability for provide instant information to the user.

Then we briefly studied the faults and errors that can put a distributed system down. A failure is defined as a deviation from the service delivered by a system from the essential specification of this system to eliminate errors or faults; where the paradigm of fault tolerance refers to the ability of a system (computer, network, cloud cluster, etc.) to continue operating without interruption when one or more of its components fail. We have cited in this part the elements that categorize a tolerant system, the error and failure models, as well as the fault tolerance approaches. As our thesis deals with distributed systems, we talked about fault tolerance mechanisms in this type of system.

Among the methods used to overcome an error or failure failure in distributed systems is to bring the system to a stable legitimate state in an automatic way. self-stabilizing algorithms offer this opportunity to systems, it is a concept proposed by Disjktra in 1973 and is later used in fault tolerance.

self-stabilizing system is a system that can automatically recover following the occurrence of (transient) faults [Dol00]. The idea is to design systems that can be started in an arbitrary state and still converge to a desired behavior which means that this makes self-stabilization an elegant approach for transient fault-tolerance.

For this reason we chose to apply a self-stabilizing algorithm in distributed systems knowing that the application of these algorithms appears in different types of graphs with different topologies and network connections. So we presented in this thesis a state of the

art of these algorithms through their formal definition, their advantages and disadvantages, how to designate a self-stabilizing system. In addition, to run a self-stabilizing algorithm, two important elements must be taken into consideration: daemons and transformers that we have defined in this part; not forgetting the measures of complexity.

And to fully understand this paradigm, we presented a brief state-of-the-art on its use in different types of graph problems, Citing the matching, dominating set, independent set and coloring graph, in this part we present a good number of works in the literature.

As mentioned, among the graph problems treated in the literature is the search for the minimal dominating set which reflects a whole set of problems and real situations in distributed systems and sensor networks. Therefore the contribution of this thesis is to propose a new algorithm to find the minimal dominating set, we left the third chapter to clarify this paradigm, definitions and algorithms allowing to find the dominating set are presented, as well as the fields of application of this paradigm.

Finally, we moved on to our contribution in the last chapter which is the application of a self-stabilizing algorithm to find the minimal dominating set in distributed systems. To do this, we started by citing the first works applying a self-stabilizing algorithm in minimal dominating set, well the important are:

Hedetniemi and al. proposed the domination bipartition algorithm where node i has a single binary variable $x(i)$.The rules allow a node to change its value if all nodes in its closed neighborhood have the same value. Upon stabilization, the two sets of nodes $i|x(i) = 0$ and $i|x(i) = 1$ are each dominating sets, if G has no isolated nodes, thus forming a dominating bipartition [GHJS03b]

Another algorithm proposed by [GHJS03a] that requires that every node have a unique ID. Goddard and al. have used i interchangeably to denote a node, and the node's ID. they assumed there is a total ordering on the IDs. Each node i has two variables: *a pointer* $p(i)$ (which may be null) and *a boolean* $flagx(i)$. If $p(i) = j$ then they said that $i\ pointsto\ j$. At any given time, they denote with D the current set of nodes i with $x(i) = true$.

A total dominating set S is called minimal when no proper subset of S is a total dominating set of the graph G. Belhoul and al. have proposed a linear self-stabilizing algorithm for finding a minimal total dominating set, that they call MTDS, in the expression model under the unfair central scheduler. Then, they removed this assumption to have an algorithm that converges in $O(mn)$ under the unfair distributed scheduler.

After that we have described our contribution in this context. The proposed self-stabilizing algorithm for computing minimal (α, β)-dominating set called (α, β)-MDS, we have used the expression distance-2 model proposed by Turau to carry out the distributed system to legitimate state. The algorithm is studied in both theoretical and experimental sides, where it is proved that it converges in $O(nm)$ moves using a distributed daemon after proofing

the efficiency in the central daemon which is simple. The proposed algorithm confirms the conjecture of [CDD$^+$15] that there is a self-stabilizing algorithm for (f, g)-alliance when $f < g$. The later result is achieved due to the expression distance-2 model that facilitates developing self-stabilizing algorithm for calculating minimal (α, β)-dominating set. Only a upper bound is provided in terms of $moves\ number$ that shows a convergence in $O(mn)$ moves. It will be interesting to work in the future in order to find the convergence in terms of $rounds$.

After proofing the efficiency of the tow parameters α and β to find the minimal dominating set using a self-stabilizing algorithm, we conclude this thesis by some future proposal idea. We will aim to apply this algorithm in real problems especially social networks such as the spread of rumors and the influence of Internet users on each other. Also the identification of effective Leader Group of Social Network seems a very important idea to apply the algorithm. We can apply α-β-Domination in the study of the spread of diseases and infections. Also we will aim to study this algorithm in fault tolerance algorithms against complicated errors and failures such as Byzantine faults.

Bibliography

Bibliography

[ADDP19] Karine Altisen, StÃ©phane Devismes, Swan Dubois, and Franck Petit. *Introduction to distributed self-stabilizing algorithms*. by Morgan and Claypool, 2019.

[AG93] A. Arora and M. Gouda. Closure and convergence: a foundation of fault-tolerant computing. *Software Engineering, IEEE Transactions*, page 12, 1993.

[AKL18] F. N. Abu-Khzam and K. Lamaa. Efficient heuristic algorithms for positive influence dominating set in social networks. *Hot Topics in Pervasive Mobile and Online Social Networking, IEEE*, pages 610–615, 2018.

[AME21] A. A. Alofairi, E. Mabrouk, and I. Elsemman. Constraint based models for dominating protein interaction networks. *IET SystemsBiology*, 2021.

[AS88] Baruch Awerbuch and Michael Sipser. Dynamic networks are as fast as static networks. *Foundations of Computer Science Annual Symposium IEEE*, pages 206–219, 1988.

[AS99] Gheorghe Antonoiu and Pradip K. Srimani. A self-stabilizing distributed algorithm to find the median of a tree graph. *Journal of Computer and System Science*, 58:215–221, 1999.

[ASH+08] M. Atapour, S.M. Sheikholeslami, A. Hansberg, L. Volkmann, A. Khodkar, and S. Arumugam. $2-$domination subdivision number of graphs. *AKCE International Journal of Graphs and Combinatorics*, 5(2):165–173, 2008.

[Asp22] James Aspnes. Notes on theory od distributed systems. Distributed under a Commons Attribution-ShareAlike 4.0 International license, April 2022.

[AW04] H. Attiya and J. Welch. Distributed computing: fundamentals, simulations and advanced topics. *Wiley-Interscience*, page 414, 2004.

[AWF03] K. M. Alzoubi, P. Wan, and O. Frieder. Maximal independent set, weakly-connected dominating set, and induced spanners in wireless ad hoc networks. *International Journal of Foundations of Computer Science*, 14(2):287–303, 2003.

[BA99] A. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[BBP13] K. Bessaoud, A. Bui, and L. Pilard. Self-stabilizing algorithm for low weight connected dominating set. *17th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, IEEE Computer Society*, pages 231–238, 2013.

[BDTC05] Jeremy Blum, Min Ding, Andrew Thaeler, and Xiuzhen Cheng. *Connected dominating set in sensor networks and manets*. Department of Computer Science The George Washington University, Washington, DC 20002, 2005.

[Ben21] Badreddine Benreguia. *Self-stabilizing domination and localization in dynamic distributed systems*. Phd science, Department of computer sciences, Batna 2 university, 2021.

[BHS94] T. J. Bean, M. A. Henning, and H. C. Swart. On the integrity of distance domination in graphs. *Mathematics Australas. J Comb*, 10:29–43, 1994.

[BJ16] Csilla BujtÃ s and SzilÃ rd Jasko. Bounds on the 2-domination number. *Discrete Appl. Math*, 242:4–15, 2016.

[BNBJ+08] A. Bar-Noy, T. Brown, M. P. Johnson, T. La Porta, and O. Liu H. Rowaihy. Assigning sensors to missions with demands. *Algorithmic Aspects of Wireless Sensor Networks*, 2008.

[BYK14] Yacine Belhoul, Sad Yahiaoui, and Hamamache Kheddouci. Efficient self-stabilizing algorithms for minimal total k-dominating sets in graphs. *Information Processing Letters*, 114(7):339–343, 2014.

[CaR11] Ch. Cachin, R.Guerr aoui, and L. Rodrigues. Reliable and secure distributed programming. *Springer*, page 367, 2011.

[CCT14] W. Y. Chiu, C. Chen, and S. Tsai. A $4n$-move self-stabilizing algorithm for the minimal dominating set problem using an unfair distributed daemon. *Information Processing Letters*, 114(10):515–518, 2014.

[CDD+15] F. Carrier, A. K. Datta, S. Devismes, L. L. Larmore, and Y. Rivierre. Self-stabilizing (f, g)-alliances with safe convergence. *Stabilization, Safety, and Security of Distributed Systems*, 8255:61–73, 2015.

[CFHV12] M. Chellali, O. Favaron, A. Hansberg, and L. Volkmann. k-domination and k-independence in graphs: A survey. *Graphs and Combinatorics*, 28:1–55, 2012.

[CHS02] Subhendu Chattopadhyay, Lisa Higham, and Karen Seyffarth. Dynamic and self-stabilizing distributed matching. *In Proceedings of the Twenty-first Annual Symposium on Principles of Distributed Computing, PODC*, pages 290–297, 2002.

[CPADJH04] E. J. Cockaynea, S. M. Hedetniemic P. A. Dreyer Jr, and S. T. Hedetniemi. Roman domination in graphs. *Discrete Mathematics*, 278:11–22, 2004.

[Cro96] Jon Crowcroft. *Open Distributed System*. Artech House, first edition edition, 1996.

[CTB15] A. Campan, T. M. Truta, and M. Beckerich. Fast dominating set algorithms for social networks. In Michael Glass ans Jung Hee Kim, editor, *Modern AI and Cognitive Science Conference*, volume 26, pages 55–62, Greensboro, NC, USA, April 25-26, 2015.

[DDL18] A. K. Datta, S. Devismes, and L. L. Larmore. A silent self-stabilizing algorithm for the generalized minimal k-dominating set problem. *Theoretical Computer Science*, 2018.

[DFG06] Vadim Drabkin, Roy Friedman, and Maria Gradinariu. Self-stabilizing wireless connected overlays. *International Conference On Principles Of Distributed Systems OPODIS*, pages 425–439, 2006.

[DGS99] S. Dolev, Mohamed. G. Gouda, and M. Schneider. Memory requirements for silent stabilization. *Acta Informatica*, 17(111):447–462, 1999.

[DHLM00] J. Dunbar, D. Hoffman, R. Laskar, and L. Markus. α-domination. *Discrete Mathematics*, 211:11–26, 2000.

[DHR+11] S. Devismes, K. Heurtefeux, Y. Rivierre, A. K. Datta, and L. L. Larmore. Self-stabilizing small k-dominating sets. *Networking and Computing ICNC, Second International Conference*, pages 30–39, 2011.

[Die05] Reinhard Diestel. *Graph theory*. Springer-Verlag New York 1997, 2000, 2005, third edition edition, 2005.

[Dij74] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[DIM90] S. Dolev, A. Israeli, and S. Moran. Self-stabilization of dynamic systems assuming only read/write atomicity. In *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, pages 103–117, New York, NY, USA, 1990. Association for Computing Machinery.

[DLV10] Ajoy K. Datta, Lawrence L. Larmore, and Priyanka Vemula. A selfstabilizing o(k)-time k-clustering algorithm. *The Computer Journal, OUP*, 53(3):342–350, 2010.

[Dol00] Shlomi Dolev. *Self-stabilization*. The MIT Press, Cambridge, Massachusetts; London, England, 2000.

[DPRS11] M.C. Dourado, L.D. Penso, D. Rautenbach, and J.L. Szwarcfiter. The south zone: Distributed algorithms for alliances. *Stabilization, Safety, and Security of Distributed Systems*, 6976:178–192, 2011.

[DRV04] F. Dahme, D. Rautenbach, and L. Volkmann. Some remarks on α-domination. *Discuss. Math. Graph Theory*, 24:423–430, 2004.

[DT11] S. Dubois and S. Tixeuil. A taxonomy of daemons in self-stabilization. *HAL, open science*, page 15, 2011.

[DWS14a] Y. Ding, J. Z. Wang, and P. K. Srimani. A linear time self-stabilizing algorithm for minimal weakly connected dominating sets. *International Journal of Parallel Programming*, pages 151–162, 2014.

[DWS14b] Y. Ding, J. Z. Wang, and P. K. Srimani. Self-stabilizing selection of influential users in social networks. *17th International Conference on Computational Science and Engineering, IEEE*, pages 1558–1565, 2014.

[DWS16] Y. Ding, J. Z. Wang, and P. K. Srimani. A time self-stabilizing algorithm for minimal weakly connected dominating sets. *International Journal of Parallel Programming*, 44:151–162, 2016.

[EB20] G. B. Ekinci and C. BujtÃ s. Bipartite graphs with close domination and k-domination numbers. *Open mathematics*, 18:873–885, 2020.

[FEK09] W. Feng, C. Erika, and X. Kuai. Positive influence dominating set in online social networks. In Springer-Verlag, editor, *Proceedings of the 3rd International Conference on Combinatorial Optimization and Applications*, volume 1, pages 313–321, 2009.

[FHE+11] F.Wang, H.Du, E.Camacho, K.Xu W.Lee, Y.Shi, and S.Shan. On positive influence dominating sets in social networks. *Theoritical Computer Science*, 412:265–269, 2011.

[FHHR02] O. Favarona, S.M. Hedetniemib, S.T. Hedetniemib, and D.F. Rall. On k-dependent domination. *Discrete Mathematics*, 249:83–94, 2002.

[FJ85] John Frederick Fink and Michael S. Jacobson. n-domination in graphs, in: Graph theory with application to algorithms and computer science. *John Wiley and Sons, New York*, pages 283–300, 1985.

[Gar03] Felix C. Gartner. A survey of self-stabilizing spanning-tree construction algorithms. *Swiss Federal Institute of Technology EPFL; technical report*, page 21, 2003.

[Gha15] Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. *Computer Science, Data Structures and Algorithms*, 2015.

[Gha19] Mohsen Ghaffari. Distributed maximal independent set using small messages. *SODA19: Symposium on Discrete Algorithms*, pages 805–820, 2019.

[GHJS03a] W. Goddard, S. T. Hedetniemi, D. P. Jacobs, and P. K Srimani. A self-stabilizing distributed algorithm for minimal total domination in an arbitrary system graph. *Computers and Mathematics with Applications*, pages 240–243, 2003.

[GHJS03b] W. Goddard, S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani. Self-stabilizing protocols for maximal matching and maximal independent sets for ad hoc networks. *Proceedings International Parallel and Distributed Processing Symposium*, page 162, 2003.

[GHJS03c] W. Goddard, S.T. Hedetniemi, D.R. Jacobs, and P.K. Srimani. A robust distributed generalized matching protocol that stabilizes in linear time. *23rd International Conference on Distributed Computing Systems Workshops*, 2003.

[GHJS08] W. Goddard, S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani. Self-stabilizing graph protocols. *Parallel Processing Letters*, pages 189–199, 2008.

[GHS06] Wayne Goddard, Stephen T. Hedetniemi, and Zhengnan Shi. An anonymous selfstabilizing algorithm for 1-maximal matching in trees. *Information Processing Letters*, 91(2):77–83, 2006.

[Gib85] Alan Gibbons. Algorithmic graph theory. Cambridge, Massachusetts; London, England, 1985.

[GK93] S. Ghosh and M.H. Karaata. A self-stabilizing algorithm for coloring planar graphs. *Distributed Computing*, pages 55–59, 1993.

[GK10] Nabil Guellati and Hamamache Kheddouci. A survey on self-stabilizing algorithms for independence, domination, coloring, and matching in graphs. *Journal of parallel and distributed computing*, pages 406–415, 2010.

[GS10] W. Goddard and P. K. Srimani. Anonymous self-stabilizing distributed algorithms for connected dominating set in a network graph. *Proceedings of the international multi-conference on complexity, informatics and cybernetics, IMCIC*, pages 26–28, 2010.

[GS13] W. Goddard and P.K. Srimani. Daemon conversions in distributed self-stabilizing algorithms. *International Workshop on Algorithms and Computation, WALCOM 2013: WALCOM: Algorithms and Computation*, 7748:146–157, 2013.

[GT00] M. Gradinariu and S. Tixeuil. Self-stabilizing vertex coloration and arbitrary graphs. *Proceedings. 4th International Conference on Principles of Distributed Systems*, pages 55–70, 2000.

[HCW08] T. C. Huang, C. Chen, and C. Wang. A linear-time self-stabilizing algorithm for the minimal $2-$dominating set problem in general networks. *Journal of Information Science and Engineering*, 24(1):175–187, 2008.

[Her02] T.R. Herman. A comprehensive bibliography on self-stabilization. *Theoretical Computer Science, Chicago J.,*, page 21, 2002.

[HH92] Su-Chu Hsu and Shing-Tsaan Huang. A self-stabilizing algorithm for maximal matching. *Information Processing Letters*, 43(2):77–81, 1992.

[HHJ$^+$12] S. M. Hedetniemi, S. T. Hedetniemi, H. Jiang, K. Kennedy, and A. A. McRae. A self-stabilizing algorithm for optimally efficient sets in graphs. *Information Processing Letters*, 112:621–623, 2012.

[HHJS03] S.M. Hedetniemi, S.T. Hedetniemi, D.P. Jacobs, and P.K. Srimani. Self-stabilizing algorithms for minimal dominating sets and maximal independent sets. *Computers and Mathematics with Applications*, 46(5):805–811, 2003.

[HHKM13] S. M. Hedetniemi, S. T. Hedetniemi, K.E. Kennedyn, and A. Mcrae. Self-stabilizing algorithms for unfriendly partitions into two disjoint dominating sets. *Parallel Processing Letters*, 23(1), 2013.

[HHS98] T. W. Haynes, S. Hedetmieni, and P. Slater. *Fundamentals of Domination in Graphs: Advanced Topics*. Marcel Dekker, Inc, 270 Madison Avenue, New York, 10016, first edition edition, 1998.

[HHT03] S.T. Huang, S.S. Hung, and C.H. Tzeng. Linear time self-stabilizing colorings. *Information Processing Letters 87*, pages 251–255, 2003.

[HHT05] S.T. Huang, S.S. Hung, and C.H. Tzeng. Self-stabilizing coloration in anonymous planar networks. *Information Processing Letters*, 95(1):307–312, 2005.

[HJS01] S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani. Maximal matching stabilizes in time $o(m)$. *Information Processing Letters*, pages 221–223, 2001.

[HLCW07] T. C. Huang, J. Lin, C. Chen, and C. Wang. A self-stabilizing algorithm for finding a minimal $2-$dominating set assuming the distributed demon model. *Computers et Mathematics with Applications*, 54(3):350–356, 2007.

[HSM+07] H.Rowaihy, S.Eswaran, M.Johnson, D.Verma, A.Bar-Noy, T.Brown, and T.LaPorta. A survey of sensor selection schemes in wireless sensor networks. *In Society of Photo-Optical Instrumentation Engineers SPIE Conference Series*, page 22, 2007.

[Hua05] T. C. Huang. A self-stabilizing algorithm for the shortest path problem assuming read/write atomicity. *Journal of Computer and System Sciences*, 71:70–85, 2005.

[HW92] Walter L. Heimerdinger and Charles B. Weinstock. A conceptual framework for system fault tolerance. Technical Report CMU/SEI-92-TR-033, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, 1992.

[IKK02] M. Ikeda, S. Kamei, and H. Kakugawa. A space-optimal self-stabilizing algorithm for the maximal independent set problem. *Third International Conference on Parallel and Distributed Computing Applications and Technologies PDCAT*, pages 70–74, 2002.

[Ioa02] Kleoni Ioannidou. Transformations of self-stabilizing algorithms. *Distributed Computing (DISC); Lecture Notes in Computer Science*, pages 113–117, 2002.

[ISO95] ISO. Open distributed processing reference model. *International Standard ISOIlEC IS 10746*, page 5, 1995.

[JG05] Ankur Jain and A. Gupta. A distributed self-stabilizing algorithm for finding a connected dominating set in a graph. *Sixth International Conference on Parallel and Distributed Computing Applications and Technologies PDCAT, IEEE*, pages 615–619, 2005.

[Kar99] Mehmet Hakan Karaata. A self-stabizing algorithm for finding articulation point. *International Journal of Foundations of Computer Science*, 10(1):33–46, 1999.

[Ken06] Nadia Nosrati Kenareh. Domination in graphs. Master of science, B.sc, Sharif university, 2006.

[Kes88] J. L. W. Kessels. An exercise in proving self-stabilization with a variant function. *Information Process Letter*, 29(1):39–42, 1988.

[KK03] S. Kamei and H. Kakugawa. A self-stabilizing algorithm for the distributed minimal k-redundant dominating set problem in tree networks. *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT, IEEE*, pages 720–724, 2003.

[KK05] S. Kamei and H. Kakugawa. A self-stabilizing approximation algorithm for the distributed minimum k-domination. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 88(5):1109–1116, 2005.

[KK06] A. Kosowski and L. Kuszner. Self-stabilizing algorithms for graph coloring with improved performance guarantees. *International Conference on Artificial Intelligence and Soft Computing, Artificial Intelligence and Soft Computing ICAISC*, pages 1150–1159, 2006.

[KK08] Sayaka Kamei and Hirotsugu Kakugawa. A self-stabilizing approximation for the minimum connected dominating set with safe convergence. *International Conference On Principles Of Distributed Systems, OPODIS, Principles of Distributed Systems*, pages 496–511, 2008.

[KK10] Sayaka Kamei and Hirotsugu Kakugawa. A self-stabilizing distributed approximation algorithm for the minimum connected dominating set. *International Journal of Foundations of Computer Science*, pages 459–476, 2010.

[KP93] Shmuel Katz and Kenneth J. Perry. Self-stabilizing extensions for message-passing systems. *Distributed Computing*, 7(1):17–26, 1993.

[KS00] M.H. Karaata and K.A. Saleh. A distributed self-stabilizing algorithm for finding maximum matching. *Computer Systems Science and Engineering*, 15(3):175–180, 2000.

[KS08] A. D. Kshemkalyani and M. Singhal. *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, New York, NY, USA, 2008.

[Kus05] L. Kuszner. Tools to develop and test self-stabilizing algorithms. 2005.

[KV09] K. Kammerling and L. Volkmann. Roman k-domination in graphs. *J. Korean Math. Soc.*, 46:1309–1318, 2009.

[Lam84] Leslie Lamport. Solved problems, unsolved problems and non-problems in concurrency. *PODC, Proceedings of the third annual ACM symposium on Principles of distributed computing*, page 11, 1984.

[LLY09] D. Li, L. Liu, and H. Yang. Minimum connected r-hop k-dominating set in wireless networks. *Discrete Mathematics, Algorithms and Applications*, 1:45–57, 2009.

[MM07] Fredrik Manne and Morten Mjelde. A self-stabilizing weighted matching algorithm. *Stabilization, Safety, and Security of Distributed Systems*, pages 383–393, 2007. Lecture Notes in Computer Science book series LNTCS.

[MMBP11] T. Milenkovic, V. Memiscc, A. Bonato, and N. Przculj. Dominating biological networks. *PLoS ONE*, 6, 2011.

[MMPT07] Fredrik Manne, Morten Mjelde, Laurence Pilard, and Sebastien Tixeuil. A new self-stabilizing maximal matching algorithm. *Proceedings of the 14th International Colloquium on Structural Information and Communication Complexity (Sirocco 2007)*, pages 96–108, 2007.

[MPR19] A. R. Molla, S. Pandit, and S. Roy. Optimal deterministic distributed algorithms for maximal independent set in geometric graphs. *Journal of Parallel and Distributed Computing*, pages 36–47, 2019.

[Neg15] B. Neggazi. *Self-stabilizing algorithms for graph parameters*. PhD thesis, UniversitÃ© Claude Bernard Lyon 1, 04 2015.

[NGHK15] B. Neggazi, N. Guellati, M. Haddad, and H. Kheddouci. Efficient self-stabilizing algorithm for independent strong dominating sets in arbitrary graphs. *International Journal of Foundations of Computer Science*, 26(06):751–768, 2015.

[NHNT20] M. H. Nguyen, M. Hoang Ha, D. N. Nguyen, and The Trung Tran. Solving the k−dominating set ptoblem on very large−scale networks. *Computational Social Networks*, 7(4), 2020.

[NIE73] J. NIEMINEN. Two bounds for the domination number of a graph. *Inst. Maths Applies*, pages 183–187, 1973.

[Ore62] Oystein Ore. *Theory of graphs*, volume 38. Providence, American Mathematical Society, fourth 1967 edition, 1962.

[PT14] Sushant Patial and Jawahar Thakur. Checkpointing and rollback recovery algorithms for fault tolerance in manets: A review. *Int. J. Advanced Networking and Applications*, 6(3):2308–2313, 2014.

[RMM05] George Roussos, Andy J. Marsh, and Stavroula Maglavera. Enabling pervasive computing with smart phones. *Published by the IEEE CS and IEEE ComSoc*, pages 20–28, April 2005.

[RR00] C. S. ReVelle and K. E. Rosing. Defendens imperium romanum: A classical problem in military strategy. *The American Mathematical Monthly*, 107(7):585–594, 2000.

[RTAS09] H. Raei, M. Tabibzadeh, B. Ahmadipoor, and S. Saei. A self-stabilizing distributed algorithm for minimum connected dominating sets in wireless sensor networks with different transmission ranges. *eleventh International Conference on Advanced Communication Technology, IEEE*, pages 526–530, 2009.

[SA15] Arif Sari and Murat Akkaya. Fault tolerance mechanisms in distributed systems. *International Journal of Communications, Network and System Sciences*, 8(12):471–482, 2015.

[SGH04] Z. Shi, W. Goddard, and S. T. Hedetniemi. An anonymous self-stabilizing algorithm for 1-maximal independent set in trees. *Information Processing Letters*, 91(2):77–83, 2004.

[Sha09] Ramy S. Shaheen. Bounds for the 2-domination number of toroidal grid graphs. *International Journal of Computer Mathematics*, 86(4):584–588, 2009.

[SRR94] S.K. Shukla, D.J. Rosenkrantz, and S.S. Ravi. Developing self-stabilizing coloring algorithms via systematic randomization. *Proceedings. 1st International Workrrshop on Parallel Processing*, pages 668–673, 1994.

[SRR95] S. K. Shukla, D. J. Rosenkrantz, and S.S. Ravi. Observations on self-stabilizing graph algorithms for anonymous networks. *Proceedings of the second workshop on self-stabilizing systems*, page 15, 1995.

[SS93] S. Sur and P.K. Srimani. A self-stabilizing algorithm for coloring bipartite graphs. *Information Sciences*, 69(3):219–227, 1993.

[SX07] P.K. Srimani and Z. Xu. Self-stabilizing algorithms of constructing spanning tree and weakly connected minimal dominating set. *27th International Conference on Distributed Computing Systems Workshops (ICDCSW'07)*, pages 20–28, 2007.

[SX08] Pradip K. Srimani and Zhenyu Xu. Self-stabilizing graph protocols. *Parallel Processing Letters*, 18(1):189–199, 2008.

[Tel94] Gerard Tel. Maximal matching stabilizes in quadratic time. *Information Processing Letters*, 49(6):271–272, 1994.

[TH09] Volker Turau and Bernd Hauck. A new analysis of a self-stabilizing maximum weight matching algorithm with approximation ratio 2. *Theoretical Computer Science*, 412(40):5527–5540, 2009.

[TH11] Volker Turau and Bernd Hauck. A new analysis of a self-stabilizing maximum weight matching algorithm with approximation ratio 2. *Theory Computer Science*, 412(40):5527–5540, 2011.

[Tix10] Sebastien Tixeuil. Self-stabilizing algorithms. *Algorithms and theory of computation handbook (2009) 26.1-26.45*, pages 9–10, 2010.

[TS06] A. Tanenbaum and M. V. Steen. *Distributed Systems: Principles and Paradigms*. Prentice-Hall, Upper Saddle River, NJ, second edition edition, 2006.

[TS16] A. S. Tanenbaum and M. V. Steen. A brief introduction to distributed systems. *Computing*, 98(10):1009, 2016.

[Tur07] Volker Turau. Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler. *Information Process Letter*, 103(3):88–93, 2007.

[Tur12] Volker Turau. Efficient transformation of distance-2 self-stabilizing algorithms. *Journal of Parallel and Distributed Computing*, 72(4):603–612, 2012.

[Tur13] V. Turau. Self-stabilizing algorithms for efficient sets of graphs and trees. *Information Processing Letters*, 113:771–776, 2013.

[UT11] S. Unterschutz and V. Turau. Construction of connected dominating sets in large-scale manets exploiting self-stabilization. *Distributed Computing in Sensor Systems and Workshops DCOSS*, pages 1–6, 2011.

[VKK15] S. Vijayasharmila, P.G. Kumar, and S. Kamalesh. A survey on connected dominating sets (cds) both in the wireless sensor networks and wireless ad hoc networks. *International Journal of engeneering research and technology (IJERT)*, 04, 2015.

[VMCL09] L.M. Vaquero, L.R. Merino, J. Caceres, and M. Lindner. A break in the clouds: Towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39:50–55, 2009.

[Vol09] L. Volkmann. Connected p-domination in graphs. *Util. Math*, 79:81–90, 2009.

[Woo95] Woolbright. Personal communication. 1995.

[WS05] Xinzhou Wu and R. Srikant. Regulated maximal matching: A distributed scheduling algorithm for multi-hop wireless networks with node-exclusive spectrum sharing. *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC 05, 44th IEEE Conference*, pages 342–5347, 2005.

[WWTZ13] G. Wang, H. Wang, X. Tao, and J. Zhang. A self-stabilizing algorithm for finding a minimal positive influence dominating set in social networks. In Australia Australian Computer Society, Inc., editor, *Proceedings of the Twenty-Fourth Database Technologies 2013 (ADC 2013)*, pages 93–99, 2013.

[XHGS03] Zhenyu Xu, Stephen T. Hedetniemi, Wayne Goddard, and Pradip K. Srimani. A synchronous self-stabilizing minimal domination protocol in an arbitrary network graph. *International Workshop on Distributed Computing, IWDC*, pages 26–32, 2003.

[YBHK13] S. Yahiaoui, Y. Belhoul, M. Haddad, and H. Kheddouci. Self-stabilizing algorithms for minimal global powerful alliance sets in graphs. *Information Processing Letters*, 113:365–370, 2013.

Résumé

Les systèmes distribués dynamiques sont largement utilisés dans la vie quotidienne en traitant et exploitant des informations, des documents, des services, des médias et tous les moyens de divertissement. De nombreuses entreprises créatrices de logiciels et de systèmes se font concurrence pour offrir aux utilisateurs des services performants et tolérants.

Ces SD dynamiques sont obligées à répondre aux demandes des utilisateurs dans les délais et avec des informations fiables, elles offrent tous les niveaux de sécurité; l'authenticité de tous les services et informations personnel. Ce type de systèmes apparaît dans notre vie dans tous les domaines en utilisant différents moyens de capteurs qui collectent tout type d'informations et de demandes où le défi est de s'assurer une disponibilité permanente du service dès que des pannes se produisent. De nombreux algorithmes sont utilisés ici pour tolérer les pannes et dépasser ces situations, parmi les algorithmes auto-stabilisants qui amènent le système à un état légitime même en présence de pannes et d'erreurs.

Cette thèse traite du problème de la recherche d'ensembles dominants en utilisant le paradigme auto-stabilisant dans les systèmes distribués dynamiques. Habituellement, les membres d'un ensemble dominant sont sélectionnés pour être des chefs de cluster dans les réseaux de capteurs sans fil (WSN) afin d'assurer une disponibilité permanente du service. Étant donné que les pannes se produisent fréquemment à l'intérieur du WSN en raison de l'énergie limitée de la batterie, l'algorithme d'auto-stabilisation permet de recalculer l'ensemble dominant, et donc le réseau revient à son fonctionnement normal.

Les travaux existants ont introduit de nombreuses variantes d'algorithmes auto-stabilisants qui calculent l'ensemble dominant minimal S où chaque noeud sur S a plus de voisins dans S qu'il n'en a hors S. Dans cette thèse, nous introduisons un algorithme auto-stabilisant généralisé appelé ensemble (α, β)-dominant minimal. Un ensemble à dominance α est un sous-ensemble de noeuds S tel que pour tout noeud v hors S, le taux de voisins de v à l'intérieur de S doit être supérieur à α, où $0 < \alpha \leq 1$. De même, un ensemble (α, β)-dominant est un sous-ensemble de noeuds S tel que : S est un ensemble α-dominant et pour chaque noeud v dans S, le taux de voisins de v à l'intérieur de S est supérieur à β, où $0 \leq \beta \leq 1$. Des preuves mathématiques et des tests de simulation montrent la justesse et l'efficacité de l'algorithme proposé.

A travers notre variante proposée (α, β)-domination, nous prouvons rigoureusement la conjecture de Carrier et. Al. ((f, g)-alliances auto-stabilisantes avec convergence sûre) qui ont proposé un algorithme auto-stabilisant pour une variante de domination appelée (f, g)-alliance établie uniquement lorsque $f \geq g$. Nous prouvons la correction du cas $f < g$.

Key words: *Algorithmes auto-stabilisant, Ensemble dominant minimal, α-domination, Systèmes distribués dynamiques.*

ملخص:

تُستخدم الأنظمة الديناميكية الموزعة على نطاق واسع في الحياة اليومية من خلال التعامل مع الاستغلال والمعالجة للمعلومات والوثائق والخدمات والوسائط وجميع وسائل الترفيه. تتنافس العديد من الشركات التي تنشئ برامج وأنظمة لتقديم خدمات قوية ومتسامحة للمستخدمين.

يلتزم DS الديناميكي هذا بالاستجابة لطلبات المستخدم في الوقت المناسب وبمعلومات جديرة بالثقة ، فهي توفر جميع مستويات الأمان ؛ صحة جميع الخدمات ومعلومات الموظفين. يظهر هذا النوع من الأنظمة في حياتنا في جميع المجالات باستخدام وسائل مختلفة من أجهزة الاستشعار التي تجمع كل أنواع المعلومات والطلبات حيث يكون التحدي هو ضمان توفر خدمة دائمة منذ حدوث الأعطال. يتم استخدام العديد من الخوارزميات هنا للتسامح مع الأخطاء وتجاوز تلك المواقف ، من بين خوارزميات الاستقرار الذاتي التي تنقل النظام إلى حالة شرعية حتى مع وجود حالات الفشل والأخطاء.

تتناول هذه الأطروحة مشكلة إيجاد المجموعة المسيطرة باستخدام نموذج الاستقرار الذاتي في الأنظمة الموزعة الديناميكية. عادة ، يتم اختيار أعضاء المجموعة المسيطرة ليكونوا رؤساء مجموعة في شبكات الاستشعار اللاسلكية (WSN) من أجل ضمان توافر خدمة دائم. نظرًا لأن الأعطال تحدث بشكل متكرر داخل WSN بسبب طاقة البطارية المحدودة ، فإن خوارزمية الاستقرار الذاتي تسمح بإعادة حساب المجموعة المسيطرة ، وبالتالي تعود الشبكة إلى عملها العادي.

قدمت الأعمال الحالية العديد من المتغيرات من خوارزميات الاستقرار الذاتي التي تحسب الحد الأدنى من الهيمنة على المجموعة S حيث كل عقدة خارج S لها جيران في S أكثر من S في هذه الأطروحة ، نقدم خوارزمية معممة ذاتية الاستقرار تسمى الحد الأدنى (α، β) – المجموعة المسيطرة و تظهر البراهين الرياضية واختبارات المحاكاة صحة وكفاءة الخوارزمية المقترحة.

الكلمات المفتاحية: خوارزميات الاستقرار الذاتي، المجموعة المسيطرة، الأنظمة الديناميكية الموزعة