國立臺灣大學電機資訊學院電機工程學系

碩士論文

Graduate Institute of Electrical Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master's Thesis

具局部錯誤修正的自我穩定演算法應用於極小控制集

Self-Stabilizing Algorithms for Minimal Dominating Sets with Local Error Correction

葉子瑄

Tzu-Hsuan Yeh

指導教授: 陳和麟 博士

Advisor: Ho-Lin Chen, Ph.D.

中華民國 113 年 7 月

July 2024

doi:10.6342/NTU202401576

doi:10.6342/NTU202401576

Acknowledgements

時光冉冉，兩年的碩士學程即將畫上一個句點，在這個難忘的時刻，我的心中湧現出無數的感激與感謝。首先，最要感謝我的指導老師陳和麟教授，感謝您在整個研究過程中給予的指導和支持。從我加入實驗室起，您就告訴我有什麼需要幫忙就告訴您，那時候的我以為這只是客套話。但在一次我很冒然地找您討論，老師仍然盡力地挪出時間，在校區裡跟我邊走邊討論，親切熱誠地教導，讓我深深地感受到您在教學上的盡心盡力。

猶記在研究期間，有多次當我認為是帶著滿滿準備而來和您請教時，您總能迅速分析指出其中的錯誤和不足；當我研究困住而擔心進度時，您總會再三分析與我討論，並給予方向。您的專業知識和耐心教誨讓我在每次討論完後，常常又領悟出一些關於研究的新觀念，這些也都成為我一路上解決各種研究阻礙的墊腳石。感謝您總是為學生的角度著想，讓我們思考、規劃去做任何自己想做的事，而您都會完全地給予支持和協助。

在此也感謝我的家人在生活中的關照和支援，成為我最堅強的後盾，你們的支持和鼓勵讓我無後顧之憂地向前邁進，持續保有成長衝勁; 同時，感謝好友們的相伴和加油打氣，讓我在這段漫長的研習中充滿了歡笑和力量; 還要感謝實驗室學長們無私地傳承與分享經驗，為初到的我解答許多疑惑，此外，也很珍惜和學弟妹一起在實驗室各自拼搏、吃飯休息時又能無話不談的時光，總能讓我從研究壓

doi:10.6342/NTU202401576

力中跳脫出來，放鬆心情。最後感謝每一位曾經幫助過我的人，是你們在背後無形地影響著我，成為我堅持不懈的動力。因為有你們，我的學習旅程才變得如此豐富而精彩無比。

doi:10.6342/NTU202401576

摘要

極小控制集（minimal dominating set, MDS）在解決優化問題方面有廣泛的應用，特別是在分散式系統中。本論文以分散式系統作為應用背景，這些系統通常由多個相互連接的計算機節點組成，透過網絡進行通信和協作。然而，這種分散式環境中的系統設計面臨著多重挑戰，如節點故障、通信延遲和節點間的不一致性狀態。

為了應對這些挑戰，本研究採用了自我穩定演算法（self-stabilizing algorithms）。自我穩定演算法能夠使系統在面對節點狀態變化或故障時自動恢復到一個合法的運行狀態，而無需中央控制的介入。我們提出了兩種基於自我穩定演算法的設計，分別應用於不同距離的圖形模型，並證明他們皆能夠在距離為 3 的範圍內達到穩定狀態。其中，第二個演算法在距離為 2 的更短圖形模型上運行，但所需的移動次數和輪數的上界僅比第一個演算法高出常數倍。

此外，我們證明任何演算法穩定所需的時間都必然與節點數量成線性關係，因為無法輕易破壞圖形對稱性。在一個任意起始狀態下的圖形中，我們的第一個演算法從最近的邱等人 [3] 那篇研究最多需要 $2n$ 的移動次數減少至 n，而第二個演算法除了與先前在完全相同的衡量標準上進行比較外，還新增了一個重要特性: 具有局部修正的能力，能有效地防止錯誤的傳播，因此，我們的演算法實際需要的穩定時間與系統中被修改（即出現錯誤）的節點數量成正比。

doi:10.6342/NTU202401576

關鍵字：極小控制集、自我穩定演算法、局部錯誤修正、圖形對稱性、分散式系統、有限資源

doi:10.6342/NTU202401576

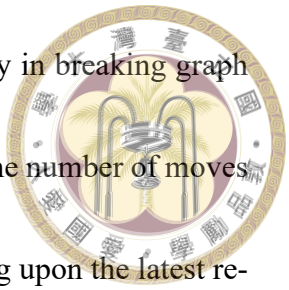
Abstract

The minimal dominating set (MDS) is widely applied in optimization problems, particularly in distributed systems. This thesis focuses on distributed systems where multiple interconnected computing nodes communicate and collaborate via networks. However, designing systems in such distributed environments presents several challenges, including transient faults, communication delays, and inconsistent node states.

To address these challenges, this study employs self-stabilizing algorithms. These algorithms enable systems to automatically recover to a legitimate configuration in response to node state changes or transient faults, without the need for centralized control intervention. We propose two self-stabilizing algorithms under different distance models, demonstrating their ability to achieve system stabilization within a distance of 3 hops. The second algorithm, although designed for a shorter distance-2 model, requires only a constant factor more moves and rounds than the first algorithm to stabilize.

Furthermore, we prove that the stabilization time required by any algorithm must in-

doi:10.6342/NTU202401576



herently scale linearly with the number of nodes, due to the difficulty in breaking graph symmetry. Starting from any initial configuration, we have reduced the number of moves required by the first algorithm from a maximum of $2n$ to n, improving upon the latest research by Chiu et al. [3]. Additionally, our second algorithm, while maintaining the same performance standards as the latest research by Chiu et al., introduces a significant new property: localized error correction. This feature effectively prevents error propagation, resulting in the stabilization time being proportional to the number of nodes that have been modified (i.e., contain errors).

Keywords: minimal dominating set, self-stabilizing algorithm, local error correction, graph symmetry, distributed system, limited resources

doi:10.6342/NTU202401576

Contents

doi:10.6342/NTU202401576

doi:10.6342/NTU202401576

List of Figures

doi:10.6342/NTU202401576

doi:10.6342/NTU202401576

List of Tables

doi:10.6342/NTU202401576

doi:10.6342/NTU202401576

List of Algorithms

doi:10.6342/NTU202401576

doi:10.6342/NTU202401576

Chapter 1 Introduction

1.1 Motivation

A minimal dominating set (MDS) has broad applications across diverse fields, particularly in network theory [1, 3, 4, 10, 12, 14, 16, 17, 19, 21] and image processing [2, 6, 7, 13, 15, 20]. In communication networks, an MDS optimizes communication efficiency by selecting the minimal number of nodes necessary to cover all nodes, thereby reducing overhead and congestion. It is also utilized in wireless sensor networks to improve network coverage and energy consumption. Additionally, in image processing, it helps identify key features. Knowing that finding a minimum dominating set is an NP-complete problem [9], this paper focuses on the study of minimal dominating sets. Consequently, finding an MDS is crucial for solving a variety of optimization problems where minimalism and efficiency are paramount.

Having introduced MDS and highlighted its significance in covering all nodes efficiently while reducing resource usage, it is crucial to consider the robustness and adaptability of such solutions. In this context, self-stabilizing algorithms play a pivotal role. Self-stabilizing algorithm, first introduced by Dijkstra [5], is a fault-tolerant algorithm designed to ensure system recovery from arbitrary initial states, making it particularly useful in distributed systems. The algorithm iteratively executes rules for each node based on its current state and the states of its neighbors by evaluating the preconditions of these rules,

doi:10.6342/NTU202401576

which are boolean expressions composed of predicates. When a precondition is satisfied, the node undergoes a state change, referred to as a *move*. A node is *privileged* if it satisfies the precondition of any rule. A *round* is defined as the period during which all privileged nodes simultaneously complete once state changes, after which the system proceeds to the next round. The algorithm guarantees stabilization in a legitimate state after a finite number of moves and will remain in that state indefinitely until faults occur.

Consider a wireless system with limited resources. Each node must transmit its state to neighboring nodes through a broadcasting process triggered by a move, which consumes significant energy. The system must remain functional until the final round is fully completed, making the duration of rounds critical to the system's overall runtime. Consequently, the complexity of constructing a minimal dominating set using self-stabilizing algorithms is directly influenced by the number of moves and rounds required. Additionally, the system must stabilize locally after any arbitrary transient faults to ensure continuous and reliable operation. Thus, our objective is to develop self-stabilizing algorithms that minimize both the number of moves and rounds and ensure local stabilization after arbitrary transient faults. This approach aims to enhance efficiency and maintain reliability within the constraints of available resources.

1.2 Previous Works

In this section, we review previous works (see in Table 1.1) that have utilized self-stabilizing algorithms to find a minimal dominating set or a minimal k-dominating set when $k = 1$, which is equivalent to finding an MDS, under different scheduler models.

Hedetniemi et al. [11] presented the first self-stabilizing algorithm for finding an MDS under a centralized scheduler, stabilizing in $(2n + 1)n$ moves. Xu et al. [19] pro-

doi:10.6342/NTU202401576

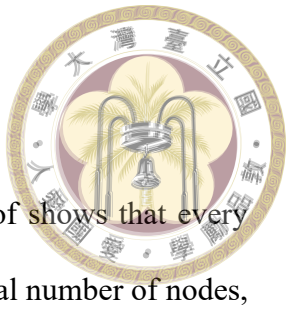
posed a $4n$ rounds algorithm in the synchronous model. In [8], Gairing et al. explored an algorithm for finding a minimal k-dominating set under the distance-2 model, demonstrating $(k+1)n$ moves. For the specific case of $k = 1$, their algorithm achieves a minimal dominating set with a maximum of $2n$ moves. This research pioneered the use of unique identifiers to break graph symmetry. Since then, every subsequent study has also used unique identifiers, necessitating that the system cannot be anonymous. Moreover, they provided a mechanism to convert the algorithm from the distance-2 model to the distance-1 model. Under the distance-1 model, their algorithm requires $\mathcal{O}(mn)$ moves, where m represents the number of edges.

Furthermore, Turau [17] introduced an algorithm requiring $9n$ moves under an unfair distributed scheduler. This work was the first to introduce the WAIT state, in addition to the commonly used IN and OUT states. Goddard et al. [10] provided a $4n + 1$ rounds algorithm when executed under a synchronous scheduler. However, under a distributed scheduler, the same algorithm requires $5n$ moves. Tsai [16] proposed an algorithm that converges in $4n$ moves under a distributed scheduler.

In [18], Turau conducted a second study on the minimal k-dominating set, showing that when $k = 1$, it stabilizes after $2n$ moves in the distance-2 model using a centralized scheduler. He also introduced a more efficient transformer requiring $\mathcal{O}(mn)$ moves using a distributed scheduler in the distance-1 model. Chiu et al. [3] presented an algorithm that requires $4n - 2$ moves using an unfair distributed scheduler, establishing that this bound is tight. Additionally, they proposed a self-stabilizing MDS-silent algorithm within the distance-2 model, which stabilizes in at most $2n - 1$ moves under an unfair distributed scheduler.

doi:10.6342/NTU202401576

1.3 Our Results



Whereas earlier studies began with any configuration, our proof shows that every protocol requires moves and rounds inherently proportional to the total number of nodes, n. This is because breaking graph symmetry is a significant challenge, as it requires distinguishing between equivalent states within a graph, often leading to repeated analysis of identical segments. In contrast, our approaches start with a stabilized system, assuming that only the states of k nodes have been altered. This allows us to achieve at most $k \cdot poly(\Delta)$ total state changes and rounds, where $poly(\Delta)$ is a polynomial function of the maximum degree of the graph.

Reference	**Scheduler**	**Stabilization Time**	
		Moves (number)	**Rounds (number)**
Hedetniemi et al. [11]	Centralized	$\leq 2n^2 + n$	-
Xu et al. [19]	Synchronous	-	$\leq 4n$
Gairing et al. [8] (distance-2)	Centralized	$\leq 2n$	-
Gairing et al. [8]	Centralized	$\mathcal{O}(mn)$	-
Turau [17]	Distributed	$\leq 9n$	-
Goddard et al. [10]	Distributed	$\leq 5n$	-
Goddard et al. [10]	Synchronous	-	$\leq 4n + 1$
Tsai [16]	Distributed	$\leq 4n$	-
Turau [18](distance-2)	Centralized	$\leq 2n$	-
Turau [18]	Distributed	$\mathcal{O}(mn)$	-
Chiu et al. [3]	Distributed	$\leq 4n - 2$	-
Chiu et al. [3](distance-2)	Distributed	$\leq 2n - 1$	-
any protocol	Distributed	$\Omega(n)$	$\Omega(n)$
See Sec. 4.1 of this study (distance-4)	Distributed	$\leq \min(n, k\Delta^2)$	$\leq k\Delta^2 - k\Delta + 1$
See Sec. 5.1 of this study (distance-2)	Distributed	$\leq \min(n(2\Delta - 1), 3k\Delta^2 - 2k\Delta)$	$\leq k\Delta^2 + 1$

Table 1.1: Comparison of Complexity in Self-Stabilizing Algorithms for Finding an MDS

In this study, we utilize the self-stabilizing concept to devise two 3-hop local stabilization algorithms for an MDS under a distributed scheduler. A *hop* is defined as the

doi:10.6342/NTU202401576

distance that can be traversed by crossing a single edge in the graph. These algorithms differ in the distances over which each node can gather information from its neighbors. The first algorithm operates in a distance-4 model, re-stabilizing in at most $\min(n, k\Delta^2)$ moves (based on Theorem 4.2.9 and Theorem 4.2.7) and $k\Delta^2 - k\Delta + 1$ rounds (based on Theorem 4.2.8), detailed as Algorithm 1 in Section 4.1. The second algorithm operates in a distance-2 model, achieving stabilization in at most $\min(n(2\Delta - 1), 3k\Delta^2 - 2k\Delta)$ moves (based on Theorem 5.2.12 and Theorem 5.2.10) and $k\Delta^2 + 1$ rounds (based on Theorem 5.2.11), detailed as Algorithm 2 in Section 5.1, consistent with the distance-2 model assumptions of Chiu et al.[3]'s recent work.

Our algorithms cascade faults and achieve local stabilization by focusing only on the altered nodes. This innovation optimizes the stabilization time complexity, making it directly proportional to the number of errors (altered nodes k). To the best of our knowledge, we are the first to propose local stabilization algorithms that effectively limit error propagation and achieve stabilization proportional to the number of altered nodes. Furthermore, by Theorem 4.2.9, starting from any configuration graph that adheres to the same assumptions as prior works, Algorithm 1 stabilizes in at most n moves while incorporating a better property.

An overview of this thesis is organized as follows. We first establish foundational definitions and notations in Chapter 2. Following this, Chapter 3 explores the lower bound of rounds for all protocols. Chapters 4 and 5 present two self-stabilizing protocols with 3-hop localized error correction for an MDS under a distributed scheduler, including proofs of correctness and convergence.

doi:10.6342/NTU202401576

doi:10.6342/NTU202401576

Chapter 2 Preliminaries

2.1 Model

In this thesis, we consider a distributed system under a distance-m model, represented by an undirected graph $G = (V, E)$ with n nodes. Here, V denotes the set of vertices (nodes) and E denotes the set of edges, each indicating a connection between nodes. Each node has a finite internal state and a locally unique numeric identifier (ID). Under the *distance-m model*, each node can synchronously gather the internal states and IDs of neighboring nodes within a distance of m hops, without knowing the entire graph's structure or size. In each round, every node first evaluates rules based on its state, ID and local information to determine if it is privileged. Then, each privileged node performs a move and updates its state simultaneously. Therefore, the system operates in discrete rounds.

It is important to note that unlike previous studies that began with arbitrary initial configuration, our approach initiates from a stabilized graph where the system has already reached a stable state with k nodes have been altered.

2.2 Self-Stabilization

An algorithm achieves *self-stabilization* if and only if, starting from any configuration, it is guaranteed to reach a legitimate state after a finite number of moves and then re-

doi:10.6342/NTU202401576

mains in that legitimate state indefinitely. Self-stabilizing algorithms are typically defined by rules that follow an "if <*precondition*>, then <*action*>" structure, each specifying a precondition that triggers an associated action. A node is privileged when the precondition of at least one rule holds true. We consider the system to be *stabilized* when all nodes are unprivileged.

2.3 Minimal Dominating Set

A *minimal dominating set*, denoted as S, is a subset of vertices in a graph $G = (V, E)$ such that each vertex $v \in V$ is either in S or has at least one neighbor in S. Additionally, there is no vertex v for which S remains a dominating set if v is removed from S.

2.4 Scheduler

In the context of self-stabilizing algorithms, schedulers play a critical role in managing the execution of processes to ensure system stability. These schedulers are typically categorized into three types: the centralized scheduler, which employs a single control node to determine which node should make a move in each round; the synchronous scheduler, which selects all the nodes to make a move simultaneously in each round; and the distributed scheduler, which selects a nonempty subset of nodes, where these nodes independently decide whether to make a move based on their local information and the information from their neighbors in each round.

doi:10.6342/NTU202401576

Chapter 3 Lower Bound on the Number of Rounds

In previous discussions, we highlighted the critical importance of the number of rounds, rather than the number of moves, in determining an algorithm's complexity, as the system must function continuously until the final round is completely concluded.

In this chapter, our objective is to establish the lower bound on the number of rounds required for convergence in an arbitrary graph. Determining this lower bound is pivotal for minimizing the number of rounds, thereby optimizing the algorithm's efficiency and reducing overall energy consumption. We consider a linear graph with node identifiers ranging sequentially from the minimum to the maximum over a length of n, as shown in Figure 3.1. Initially, all nodes in the graph are in a dominating set, denoted as S', meaning they are in the IN state, represented by the color gray. Otherwise, nodes not in S' are in the OUT state, represented by the uncolored nodes. We aim to let S' stabilize in a minimal dominating set.

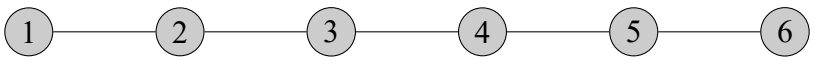


Figure 3.1: An Example of $\Omega(n)$ Rounds Required for Stabilization

Theorem 3.0.1. *Starting from the configuration described above, every protocol requires at least* $\Omega(n)$ *bound in rounds to stabilize.*

Proof. In the initial configuration above, the nodes that are not neighbors of the minimal

doi:10.6342/NTU202401576

ID node are presented symmetrically. Consequently, they may change to the same state simultaneously in a single round. Regardless of whether these nodes collectively change to the OUT or IN state, the resulting set does not constitute a minimal dominating set. This interdependency necessitates a linear number of rounds for the system to converge. Therefore, in the described initial configuration, every self-stabilizing protocol requires $\Omega(n)$ rounds for stabilization. □

This proof highlights the challenge of breaking symmetry in the initial configuration and avoiding linear scaling of rounds with the number of nodes. In the subsequent chapter, we present an advancement in self-stabilizing algorithms, which require the number of rounds to be proportional to the number of altered nodes. Starting from a stabilized graph, our system efficiently manages arbitrary transient faults without propagating errors. Unlike previous works that require starting from an arbitrary configuration and, after stabilization, a single fault may cause the entire graph to re-stabilize in a linear number of rounds, our approach only necessitates localized adjustments, providing practical enhancements in system stability and efficiency.

doi:10.6342/NTU202401576

Chapter 4 First Local Stabilization Algorithm for MDS (distance-4)

This chapter presents a 3-hop local stabilization in the self-stabilizing algorithm for a minimal dominating set, operating under an unfair distributed scheduler in a distance-4 model. Consider an undirected graph $G = (V, E)$, representing a distributed system. Initially, the graph is in a stable state. However, arbitrary transient faults alter the states of k nodes. Despite these changes, utilizing our algorithm, the graph can stabilize within 3 hops from the k altered nodes, eventually achieving a minimal dominating set.

It is established that if a node and all of its neighbors are not in the set, then the set does not constitute an MDS. Conversely, if removing any node from the set still results in a dominating set, then the set is not an MDS. Therefore, we select these two types of nodes to implement modifications in our algorithm. The detailed algorithm is presented as Algorithm 1 in Section 4.1. The proof of Algorithm 1, along with the upper bound of $\min(n, k\Delta^2)$ moves and $k\Delta^2 - k\Delta + 1$ rounds is presented in Section 4.2.

doi:10.6342/NTU202401576

4.1 A $k\Delta^2$ Moves Algorithm (distance-4)

In this algorithm, we assume that each node can directly gather information from all nodes within a distance of four nodes. Furthermore, each node has a unique numeric ID, distinct within a 4-hop radius, and includes a two-valued variable called $state$, which can be either IN or OUT. The value IN (or OUT) indicates whether a node is (or is not) in the MDS. To break symmetry, we use the technique described by [8], which employs locally unique ID. In addition, our algorithm introduces a constraint ($InToOut_1\_noBtNbr_2(v) \wedge InToOut_2\_noBtNbr_2(v)$) to rule R2. This constraint ensures that no nodes transition from IN state to OUT state simultaneously within a 2-hop radius.

The first local stabilization algorithm for MDS (distance-4) is shown in Algorithm 1.

Algorithm 1 A $k\Delta^2$ moves Algorithm (distance-4)

variables

- $v.state \in \{\text{OUT}, \text{IN}\}$ // $S = \{v : v.state = \text{IN}\}$

macros

- $noInNbr(v) \equiv \nexists w \in N(v) : w.state = \text{IN}$
- $noBtNbr_2(v) \equiv \nexists w \in N(v) : w.state = \text{OUT} \wedge w.id < v.id \wedge w$ has no IN state neighbor
- $noDpNbr_2(v) \equiv \nexists w \in N(v) : w.state = \text{OUT} \wedge w$ has exactly one IN state neighbor
- $InToOut_1\_noBtNbr_2(v) \equiv \nexists w \in N(v) : w.state = \text{IN} \wedge \neg noInNbr(w) \wedge noDpNbr_2(w) \wedge w.id < v.id$
- $InToOut_2\_noBtNbr_2(v) \equiv \nexists u \in N(N(v)) : u.state = \text{IN} \wedge \neg noInNbr(u) \wedge noDpNbr_2(u) \wedge u.id < v.id$

rules

R1: if $v.state = \text{OUT} \wedge noInNbr(v) \wedge noBtNbr_2(v)$
then $v.state := \text{IN}$; // join S

R2: if $v.state = \text{IN} \wedge \neg noInNbr(v) \wedge noDpNbr_2(v) \wedge InToOut_1\_noBtNbr_2(v) \wedge InToOut_2\_noBtNbr_2(v)$
then $v.state := \text{OUT}$; // leave S

To formally establish the rules of $k\Delta^2$ moves algorithm (distance-4), we need to define the following predicates for each node v:

doi:10.6342/NTU202401576

- $noInNbr(v) \equiv \nexists w \in N(v) : w.state =$ IN
- $noBtNbr_2(v) \equiv \nexists w \in N(v) : w.state =$ OUT $\wedge\ w.id < v.id \wedge w$ has no IN state neighbor
- $noDpNbr_2(v) \equiv \nexists w \in N(v) : w.state =$ OUT $\wedge\ w$ has exactly one IN state neighbor
- $InToOut_1\_noBtNbr_2(v) \equiv \nexists w \in N(v) : w.state =$ IN $\wedge \neg noInNbr(w) \wedge noDpNbr_2(w) \wedge w.id < v.id$
- $InToOut_2\_noBtNbr_2(v) \equiv \nexists u \in N(N(v)) : u.state =$ IN $\wedge \neg noInNbr(u) \wedge noDpNbr_2(u) \wedge u.id < v.id$

The predicate $noInNbr(v)$ indicates that a node v has no neighbors in the IN state. Additionally, the predicate $noBtNbr_2(v)$, which requires 2-hop information, indicates that a node v has no better neighbors within a distance of 1 hop that are attempting to join the MDS simultaneously and have a smaller ID. Furthermore, the predicate $noDpNbr_2(v)$ indicates that an IN state node v has no dependent neighbors who are in the OUT state and has exactly one IN state neighbor. Finally, the predicate $InToOut_1\_noBtNbr_2(v)$ (resp., $InToOut_2\_noBtNbr_2(v)$) indicates that a node v has no better neighbors within a distance of 1 hop (resp., 2 hops) that are attempting to leave the MDS simultaneously and have a smaller ID than v's ID. Consequently, determining if any 2-hop neighbors intend to transition to the OUT state necessitates the use of 4-hop information.

4.2 Correctness and Convergence

Lemma 4.2.1. *Starting from a stabilized graph, suppose the states of k nodes have been altered. In any configuration where no node is privileged, the set S is a minimal dominat-*

doi:10.6342/NTU202401576

ing set of G.

Proof. In our proof, we incorporate a part of the proof presented in [11, Lemma 16.]. By contradiction, suppose S is not a minimal dominating set, but no node is privileged. This assumption leads to two possibilities: (i) S is not a dominating set, or (ii) S is a dominating set but lacks minimality.

In case (i), if S is not a dominating set, there must exist at least one node in the OUT state that has no neighbors in S. Denote the set of all such nodes by S'. Let $v1_0$ be a node with the minimal ID in S'. Then, $v1_0$ satisfies the precondition for rule R1 and executes the rule. There is a contradiction with (i) S is not a dominating set.

In case (ii), if S is a dominating set but lacks minimality, there must exist at least one node $v2$ in S that can be removed while still maintaining the dominating set property. Consequently, $v2$ has at least one IN state neighbor ($\neg noInNbr$) and satisfies the condition $noDpNbr_2$. Let $v2_0$ be a node with the minimal ID in S. Then, $v2_0$ satisfies the precondition for rule R2 and executes the rule. There is a contradiction with (ii) S is a dominating set but lacks minimality. □

Lemma 4.2.2. *Starting from a stabilized graph, suppose the states of k nodes have been altered. If an OUT node has a neighbor in the IN state, it will never change its state.*

Proof. Starting from a stabilized graph, suppose the states of k nodes have been altered. Consider a node v in the OUT state that has at least one IN state neighbor. For v to transition to the IN state, all its neighbors must first become OUT (using rule R2). However, in each round, only one neighbor can execute rule R2. Thus, there comes a crucial moment when v has just one IN state neighbor while the rest are in the OUT state. At this juncture, the lone IN neighbor cannot satisfy the condition $noDpNbr_2(v)$, failing to trigger the transition back to the OUT state. Consequently, the OUT node remains trapped in its

doi:10.6342/NTU202401576

state indefinitely. □

Lemma 4.2.3. *Starting from a stabilized graph, suppose the states of k nodes have been altered. If a node in the OUT state is at a distance of 2 hops or more from each of the k altered nodes, then this OUT state node will never change its state.*

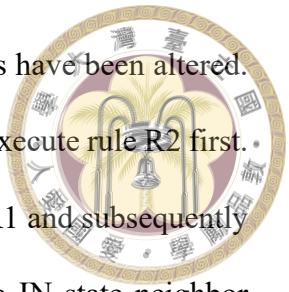
Proof. Starting from a stabilized graph, every OUT state node must have at least one neighbor in the IN state. To ensure that an OUT state node keeps its state, it is necessary that none of its IN state neighbors are altered. According to Lemma 4.2.2, OUT state nodes located at a distance of 2 hops or more from the k altered nodes will never change their state. □

Lemma 4.2.4. *Starting from a stabilized graph, suppose the states of k nodes have been altered. If a node in the IN state is at a distance of 4 hops or more from each of the k altered nodes, then this IN state node will never change its state.*

Proof. Starting from a stabilized graph, every IN node has only two possibilities: either it has no neighbor in the IN state ($noInNbr$) or it satisfies the condition $\neg noDpNbr_2$. To ensure that an IN state node keeps its state, neither it nor the nodes within 2 hops of it (by the definition of $noDpNbr_2$) will be altered. Based on Lemma 4.2.3, if all OUT nodes that are 2 hops away from the IN nodes are also at a distance of 2 hops or more from the k altered nodes, we can conclude that these OUT nodes will not change their state. Consequently, IN nodes that are at a distance of 4 hops or more from the k altered nodes will never change their state. Therefore, the state will change within a maximum of 3 hops. □

Lemma 4.2.5. *Starting from a stabilized graph, suppose the states of k nodes have been altered. Each node can execute any rule at most once in total; after that, it will not execute any further rules.*

doi:10.6342/NTU202401576



Proof. Starting from a stabilized graph, suppose the states of k nodes have been altered. If a node executes a rule, it will either (i) execute rule R1 first or (ii) execute rule R2 first.

First, consider the case (i). Let $v1$ be a node that executes rule R1 and subsequently changes to IN state. In the current round, we know that $v1$ has no IN state neighbor ($noInNbr$) and satisfies the condition $noBtNbr_2$. This implies that only $v1$ changes to IN (using rule R1) in the current round. In the next round, every OUT state neighbor of $v1$ has a neighbor in the IN state. By Lemma 4.2.2, the OUT state neighbors of $v1$ will never change their state. Thus, $v1$ will never have a neighbor in the IN state, which means that $v1$ will not execute any further rule.

Next, consider the case (ii). Let $v2$ be a node that executes rule R2 and subsequently changes to state OUT. In the current round, we know that $v2$ has an IN state neighbor ($\neg noInNbr(v2)$) and satisfies the condition $InToOut_1\_noBtNbr_2(v2) \wedge InToOut_2\_noBtNbr_2(v2)$. It implies that only $v2$ changes to OUT (using rule R2) in the current round. According to Lemma 4.2.2, since $v2$ has a neighbor in the IN state, it will never change its state. □

Lemma 4.2.6. *Starting from a stabilized graph, suppose the states of k nodes have been altered. Before the graph stabilizes, at most $k\Delta(\Delta-1)$ nodes in the IN state will change to the OUT state.*

Proof. According to Lemma 4.2.3, at most $k\Delta$ nodes will change their state from OUT to IN, and they are at a distance of 1 hop from the k altered nodes. In the current round, none of the $k\Delta$ nodes have an IN state neighbor ($noInNbr$), meaning that there are at most $k\Delta(\Delta-1)$ OUT state nodes at a distance of 2 hops from the k altered nodes. According to the definition of $noDpNbr$, we know that if an OUT state node, denoted as w, has at least one neighbor change to IN state, then w has at most one IN state neighbor change to OUT (satisfying $noDpNbr_2$ in the rule R2 precondition). Based on Lemma 4.2.4, if a

doi:10.6342/NTU202401576

node in the IN state is at a distance of 4 hops or more from each of the k altered nodes, it will never change its state. Hence, at most $k\Delta(\Delta-1)$ IN state nodes will change to OUT (using rule R2). □

Theorem 4.2.7. *Starting from a stabilized graph, suppose the states of k nodes have been altered. Algorithm 1 stabilizes to a minimal dominating set within at most $k\Delta^2$ moves, where Δ denotes the maximum degree of the graph.*

Proof. Starting from a stabilized graph, suppose the states of k nodes have been altered. According to Lemma 4.2.3, at most $k\Delta$ nodes will change their state from OUT to IN, and they are at a distance of 1 hop from the k altered nodes. According to Lemma 4.2.6, at most $k\Delta(\Delta-1)$ IN state nodes will change to OUT. Based on Lemma 4.2.5, a node can only execute a rule at most once before the graph stabilizes. Summarizing the above and Lemma 4.2.1, the graph stabilizes after at most $k\Delta+k\Delta(\Delta-1)=k\Delta^2$ moves, achieving a minimal dominating set. □

Theorem 4.2.8. *Starting from a stabilized graph, suppose the states of k nodes have been altered. Algorithm 1 stabilizes to a minimal dominating set within at most $k\Delta^2-k\Delta+1$ rounds, where Δ denotes the maximum degree of the graph.*

Proof. Starting from a stabilized graph, suppose the states of k nodes have been altered. According to Lemma 4.2.3, at most $k\Delta$ nodes will change their state from OUT to IN, and they are at a distance of 1 hop from the k altered nodes. According to Lemma 4.2.6, at most $k\Delta(\Delta-1)$ nodes in the IN state will change to the OUT state. Since transitions between OUT and IN states happen simultaneously for all nodes, the maximum number of rounds occurs when only the node with the minimum ID transitions from OUT to IN in the first round. Subsequently, each of the $k\Delta(\Delta-1)$ nodes in the IN state requires at

doi:10.6342/NTU202401576

most one round to transition. Summarizing the above, the graph stabilizes after at most $1 + k\Delta(\Delta - 1) = k\Delta^2 - k\Delta + 1$ rounds with a minimal dominating set. □

Theorem 4.2.9. *Starting from any configuration graph, Algorithm 1 stabilizes to a minimal dominating set within at most n moves.*

Proof. Starting from an arbitrary configuration graph, at most n nodes may have altered states. According to Lemma 4.2.5, each node can execute any rule at most once. Therefore, the graph stabilizes after at most n moves, achieving a minimal dominating set. □

doi:10.6342/NTU202401576

Chapter 5 Second Local Stabilization Algorithm for MDS (distance-2)

This chapter presents the second 3-hop local stabilization self-stabilizing algorithm for a minimal dominating set, operating under an unfair distributed scheduler in a distance-2 model. This algorithm operates under similar assumptions and conditions as the first algorithm but with a stricter constraint: each node can only gather information within a shorter distance, thus knowing less about its neighbors. Algorithm 2, presented in Section 5.1, introduces this new algorithm. The proof of Algorithm 2, along with the upper bound of $\min(n(2\Delta-1), 3k\Delta^2-2k\Delta)$ moves and $k\Delta^2+1$ rounds, is presented in Section 5.2.

5.1 A $3k\Delta^2-2k\Delta$ Moves Algorithm (distance-2)

In this algorithm, we assume each node can directly gather information from all nodes within a 2-hop distance. Each node has a locally unique ID within this range and a state variable, which can take one of three values: IN, WAIT, or OUT. The state IN (or OUT) indicates whether a node is (or is not) in the MDS. The WAIT state signifies a node that is in the MDS but is transitioning to the OUT state. After stabilization, no node remains

doi:10.6342/NTU202401576

in the WAIT state.

The second local stabilization algorithm for MDS (distance-2) is detailed in Algorithm 2.

Algorithm 2 A $3k\Delta^2 - 2k\Delta$ moves Algorithm (distance-2)

variables

- $v.state \in \{\text{OUT}, \text{WAIT}, \text{IN}\}$ // $S = \{v : v.state = \text{IN}\}$

macros

- $noInNbr(v) \equiv \nexists w \in N(v) : w.state = \text{IN}$
- $noWaitNbr(v) \equiv \nexists w \in N(v) : w.state = \text{WAIT}$
- $noBtNbr'_2(v) \equiv \nexists w \in N(v) : w.state = \text{OUT} \wedge w.id < v.id \wedge w$ has no IN state neighbor and no WAIT state neighbor
- $noDpNbr_2(v) \equiv \nexists w \in N(v) : w.state = \text{OUT} \wedge w$ has exactly one IN state neighbor
- $WaitDpNbr_2(v) \equiv \exists w \in N(v) : w.state = \text{OUT} \wedge w$ has exactly one WAIT state neighbor and no IN state neighbor
- $WaitToOut_1\_noBtNbr(v) \equiv \nexists w \in N(v) : w.state = \text{WAIT} \wedge w.id < v.id$
- $WaitToOut_2\_noBtNbr(v) \equiv \nexists u \in N(N(v)) : u.state = \text{WAIT} \wedge u.id < v.id$

rules

S1: if $v.state = \text{OUT} \wedge noInNbr(v) \wedge noWaitNbr(v) \wedge noBtNbr'_2(v)$
then $v.state := \text{IN}$; // join S

S2: if $v.state = \text{IN} \wedge \neg noInNbr(v) \wedge noDpNbr_2(v)$
then $v.state := \text{WAIT}$;

S3: if $v.state = \text{WAIT} \wedge (noInNbr(v) \vee WaitDpNbr_2(v))$
then $v.state := \text{IN}$; // join S

S4: if $v.state = \text{WAIT} \wedge WaitToOut_1\_noBtNbr(v) \wedge WaitToOut_2\_noBtNbr(v)$
then $v.state := \text{OUT}$; // leave S

This paper utilizes the same predicates $noInNbr(v)$ and $noDpNbr_2(v)$ as defined in Algorithm 1 in Section 4.1. Additionally, new predicates specific to each node v are introduced as follows:

- $noInNbr(v) \equiv \nexists w \in N(v) : w.state = \text{IN}$
- $noWaitNbr(v) \equiv \nexists w \in N(v) : w.state = \text{WAIT}$
- $noBtNbr'_2(v) \equiv \nexists w \in N(v) : w.state = \text{OUT} \wedge w.id < v.id \wedge w$ has no IN state

doi:10.6342/NTU202401576

neighbor and no WAIT state neighbor

- $WaitDpNbr_2(v) \equiv \exists w \in N(v) : w.state =$ OUT $\wedge\, w$ has exactly one WAIT state neighbor and no IN state neighbor
- $WaitToOut_1\_noBtNbr(v) \equiv \nexists w \in N(v) : w.state =$ WAIT $\wedge\, w.id < v.id$
- $WaitToOut_2\_noBtNbr(v) \equiv \nexists u \in N(N(v)) : u.state =$ WAIT $\wedge\, u.id < v.id$

The predicate $noInNbr(v)$ indicates that a node v has no neighbors in the IN state, and the predicate $noWaitNbr(v)$ means that a node v has no neighbors in the WAIT state. Additionally, the predicate $noBtNbr'_2(v)$ indicates that a node v has no better neighbors within a 1-hop distance that has no IN state neighbor and no WAIT state neighbor (meaning they are attempting to join the MDS), and have a smaller ID. Furthermore, the predicate $WaitDpNbr_2(v)$ indicates that an IN state node v has no dependent neighbors in the OUT state, has exactly one WAIT state neighbor, and no IN state neighbors. Finally, the predicate $WaitToOut_1\_noBtNbr(v)$ (resp., $WaitToOut_2\_noBtNbr(v)$) indicates that a node v has no better neighbors within a 1-hop (resp., 2-hop) distance that are in the WAIT state (meaning they are attempting to leave the MDS) and have a smaller ID than v's ID.

5.2 Correctness and Convergence

Lemma 5.2.1. *Starting from a stabilized graph, suppose the states of k nodes have been altered. In any configuration in which no node is privileged, there is no node in the WAIT state, and the set S is a minimal dominating set of G.*

Proof. In our proof, we incorporate a part of the proof presented in [11, Lemma 16.]. Suppose there exists a node in the WAIT state. Let v be a node with the minimal ID. Then v

doi:10.6342/NTU202401576

satisfies the precondition for rule S4 and executes the rule. This contradiction implies that a legitimate configuration cannot include any WAIT state nodes. By contradiction, suppose S is not a minimal dominating set for G. This assumption leads to two possibilities: (i) S is not a dominating set, or (ii) S is a dominating set but lacks minimality.

In case (i), if S is not a dominating set, there must exist at least one node in the OUT state that has no neighbors in S. Denote the set of all such nodes by S'. Let $v1_0$ be a node with the minimum ID in S'. Since we prove that there is no WAIT state node after stabilizing, $v1_0$ satisfies the precondition for rule S1 and executes the rule. There is a contradiction with (i) S is not a dominating set.

In case (ii), if S is a dominating set but lacks minimality, there must exist at least one node $v2$ in S that can be removed while still maintaining the dominating set property. Consequently, $v2$ has at least one IN state neighbor ($\neg noInNbr$) and satisfies the condition $noDpNbr_2$. Then, $v2_0$ satisfies the precondition for rule S2 and executes the rule. There is a contradiction with (ii) S is a dominating set but lacks minimality. □

Lemma 5.2.2. *Starting from a stabilized graph, suppose the states of k nodes have been altered. If an OUT node has a neighbor in the IN state, it will never change its state.*

Proof. Starting from a stabilized graph, suppose the states of k nodes have been altered. Consider a node v in the OUT state that has at least one IN state neighbor. For v to transition to the IN state, all its neighbors must first become OUT (using rule S4). However, in each round, only one neighbor can execute rule S4. Thus, there comes a crucial moment when v has just one WAIT state neighbor while the rest are in the OUT state. At this juncture, the lone WAIT neighbor satisfied the condition $WaitDpNbr_2(v)$ becomes IN state, failing to trigger the transition back to the OUT state. Consequently, the OUT node remains trapped in its state indefinitely. □

doi:10.6342/NTU202401576

Lemma 5.2.3. *Starting from a stabilized graph, suppose the states of k nodes have been altered. If a node in the OUT state is at a distance of 2 hops or more from each of the k altered nodes, then this OUT state node will never change its state.*

Proof. Starting from a stabilized graph, every OUT state node must have at least one neighbor in the IN state. To ensure that an OUT state node keeps its state, it is necessary that none of its IN state neighbors are altered. According to Lemma 5.2.2, OUT state nodes located at a distance of 2 hops or more from the k altered nodes will never change their state. □

Lemma 5.2.4. *Starting from a stabilized graph, suppose the states of k nodes have been altered. If a node in the IN state is at a distance of 4 hops or more from each of the k altered nodes, then this IN state node will never change its state.*

Proof. Starting from a stabilized graph, every IN node has only two possibilities: either it has no neighbor in the IN state ($noInNbr$) or it satisfies the condition $\neg noDpNbr_2$. To ensure that an IN state node keeps its state, neither it nor the nodes within 2 hops of it (by the definition of $noDpNbr_2$) will be altered. Based on Lemma 5.2.3, if all OUT nodes that are 2 hops away from the IN nodes are also at a distance of 2 hops or more from the k altered nodes, we can conclude that these OUT nodes will not change their state. Consequently, IN nodes that are at a distance of 4 hops or more from the k altered nodes will never change their state. Therefore, the state will change within a maximum of 3 hops. □

Lemma 5.2.5. *Starting from a stabilized graph, suppose the states of k nodes have been altered. If a node executes rule S1, after that, it will not execute any further rules.*

Proof. Starting from a stabilized graph, suppose the states of k nodes have been altered. Let v be a node that executes rule S1 and subsequently changes to IN state. In the current

doi:10.6342/NTU202401576

round, we know that v has no neighbors in the IN state ($noInNbr$) or in the WAIT state ($noWaitNbr$), while also satisfying the condition $noBtNbr_2$. This implies that only v changes to IN (using rule S1) in the current round. In the next round, every OUT state neighbor of v will have a neighbor in the IN state. By Lemma 5.2.2, the OUT state neighbors of v will never change their state. Thus, v will never have a neighbor in the IN state, which means that v will not execute any further rule. □

Lemma 5.2.6. *Starting from a stabilized graph, suppose the states of k nodes have been altered. If a node executes rule S4, after that, it will not execute any further rules.*

Proof. Starting from a stabilized graph, suppose the states of k nodes have been altered. Let v be a node that executes rule S4 and subsequently changes to IN state. In the current round, v satisfies the condition $WaitToOut_1\_noBtNbr(v) \wedge WaitToOut_2\_noBtNbr(v)$ but does not satisfy $noInNbr$ or $WaitDpNbr_2$ in the rule S3 precondition. It implies that only v changes to OUT (using rule S4), and v has an IN state neighbor ($\neg noInNbr(v)$) in this round. According to Lemma 4.2.2, since $v2$ has a neighbor in the IN state, it will never change its state. □

Lemma 5.2.7. *Starting from a stabilized graph, suppose the states of k nodes have been altered. Each node can execute rule S2 at most $\Delta - 1$ times.*

Proof. In our proof, we use the same term from [3, Lemma 1.]. Let k be a nonnegative integer, and consider the sequence $\langle r_1, r_2, ..., r_k \rangle$ of rules, where each r_i may not be distinct. The sequence $\langle r_1, r_2, ..., r_k \rangle$ is termed a *move sequence* if a node can sequentially execute the rules r_1 through r_k. Starting from a stabilized graph, suppose the states of k nodes have been altered. Let v be a node that executes rule S2 and subsequently changes to WAIT state. After that, if v executes rule S2 twice, the possible sequence of moves for v is either (i) ⟨S4, S1, S2⟩ or (ii) ⟨S3, S2⟩.

doi:10.6342/NTU202401576

First, consider the case (i). According to Lemma 5.2.6, once v executes rule S4, it will not execute any further rules, implying that v cannot execute rule S2 again.

Next, consider the case (ii). Before executing rule S2, we know that v has at least one IN state neighbor ($\neg noInNbr(v)$) and satisfies the condition $noDpNbr_2(v)$. This implies that v has at most $\Delta - 1$ neighbors in the OUT state, which has more than one IN state neighbor. According to Lemma 5.2.6, once v executes rule S4, it will not execute any further rules, implying that v can execute rule S2 (satisfying $noDpNbr_2$ in the rule S2 precondition) at most $\Delta - 1$ times. □

Lemma 5.2.8. *Starting from a stabilized graph, suppose the states of k nodes have been altered. Each node can execute any rule at most $2(\Delta - 1) + 1$ times in total; after that, it will not execute any further rules.*

Proof. Starting from a stabilized graph, suppose the states of k nodes have been altered. By Lemma 5.2.7, a node has two possible move sequences: (i) the node never executes rule S2, or (ii) the node executes rule S2 at least once.

First, consider the case (i). Let $v1$ be a node that never executes rule S2. By Lemma 5.2.5 and Lemma 5.2.6, once $v1$ executes rule S1 or rule S4, it will not execute any further rules, implying that the move sequence of $v1$ is either $\langle$S1$\rangle$, $\langle$S3$\rangle$, or $\langle$S4$\rangle$.

Next, consider the case (ii). Let $v2$ be a node that executes rule S2 at least once. In this case, we divide the move sequence of $v2$ into actions taken before and after executing rule S2 once. Lemma 5.2.5 states that rule S1 cannot be executed before rule S2. Thus, it can only execute rule S3 before rule S2. After executing S2, $v2$ can proceed to either S3 or S4. By Lemma 5.2.6, we know that if a node executes rule S4, it will not execute any further rules after that. Consider these segments together. According to Lemma 5.2.7, a node can execute rule S2 at most $\Delta - 1$ times. Summarizing the above, each node can

doi:10.6342/NTU202401576

execute any rule at most $2(\Delta - 1) + 1$ times in total; after that, it will not execute any further rules. □

Lemma 5.2.9. *Starting from a stabilized graph, suppose the states of k nodes have been altered. Before the graph stabilizes, at most $k\Delta(\Delta - 1)$ nodes in the IN state will change to the WAIT state.*

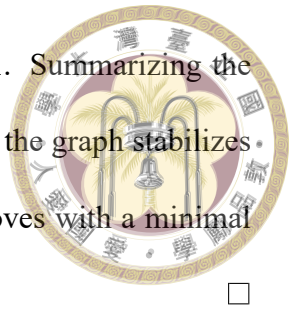
Proof. According to Lemma 5.2.3, at most $k\Delta$ nodes will change their state from OUT to IN, and they are at a distance of 1 hop from the k altered nodes. In the current round, none of the $k\Delta$ nodes have an IN state neighbor ($noInNbr$), meaning that there are at most $k\Delta(\Delta - 1)$ OUT state nodes at a distance of 2 hops from the k altered nodes. According to the definition of $noDpNbr$, we know that if an OUT state node, denoted as w, has at least one neighbor change to IN state, then w has at most one IN state neighbor change to OUT (satisfying $noDpNbr_2$ in the rule S2 precondition). Based on Lemma 5.2.4, if a node in the IN state is at a distance of 4 hops or more from each of the k altered nodes, it will never change its state. Hence, at most $k\Delta(\Delta - 1)$ IN state nodes will change to WAIT (using rule S2). □

Theorem 5.2.10. *Starting from a stabilized graph, suppose the states of k nodes have been altered. Algorithm 2 stabilizes to a minimal dominating set within at most $3k\Delta^2 - 2k\Delta$ moves, where Δ denotes the maximum degree of the graph.*

Proof. Starting from a stabilized graph, suppose the states of k nodes have been altered. According to Lemma 5.2.3, at most $k\Delta$ nodes will change their state from OUT to IN, and they are at a distance of 1 hop from the k altered nodes. According to Lemma 5.2.9, at most $k\Delta(\Delta - 1)$ IN state nodes will change to WAIT (using rule S2). Let n_i denote the number of times rule S2 is executed before node i stabilizes. Let sum_n_i be the sum of all n_i. Based on Lemma 5.2.8, node i can execute any rule at most $2n_i + 1$ times in total. Hence,

doi:10.6342/NTU202401576

the sum of all moves can be represented by $k\Delta + \sum_{i=1}^{sum\_n_i} 2n_i + 1$. Summarizing the above, sum_n_i will not exceed $k\Delta(\Delta - 1)$. Based on Lemma 5.2.1, the graph stabilizes after at most $k\Delta + 2k\Delta(\Delta - 1) + k\Delta(\Delta - 1) = 3k\Delta^2 - 2k\Delta$ moves with a minimal dominating set. □



Theorem 5.2.11. *Starting from a stabilized graph, suppose the states of k nodes have been altered. Algorithm 1 stabilizes to a minimal dominating set within at most $k\Delta^2 + 1$ rounds, where Δ denotes the maximum degree of the graph.*

Proof. Starting from a stabilized graph, suppose the states of k nodes have been altered. According to Lemma 5.2.3, at most $k\Delta$ nodes will change their state from OUT to IN, and they are at a distance of 1 hop from the k altered nodes. According to Lemma 5.2.9, at most $k\Delta(\Delta-1)$ nodes in the IN state will change to the WAIT state. When $k\Delta$ OUT state nodes each execute one round, all nodes attempting to transition to the WAIT state will successfully do so in the next round. This is because a node's transition to the WAIT state is independent of other nodes' IDs. Subsequently, each of the $k\Delta(\Delta - 1)$ nodes in the WAIT state requires at most one round to transition. Summarizing the above, the graph stabilizes after at most $k\Delta+1+k\Delta(\Delta-1) = k\Delta^2+1$ rounds with a minimal dominating set. □

Theorem 5.2.12. *Starting from any configuration graph, Algorithm 2 stabilizes to a minimal dominating set within at most $n(2\Delta - 1)$ moves.*

Proof. Starting from an arbitrary configuration graph, at most n nodes may have altered states. According to Lemma 5.2.8, each node can execute any rule at most $2(\Delta-1)+1 = 2\Delta - 1$ times in total. Therefore, the graph stabilizes after at most $n(2\Delta - 1)$ moves, achieving a minimal dominating set. □

doi:10.6342/NTU202401576

doi:10.6342/NTU202401576

Chapter 6 Discussion and Conclusion

6.1 Discussion and Fail Attempt

In our research, we explored various approaches to reduce the number of moves and rounds from $\mathcal{O}(k\Delta^2)$ to $\mathcal{O}(k\Delta)$. Our initial hypothesis involved modifying Algorithm 1, specifically rule R1, to improve the traditional method, which selects nodes based on their ID comparison. This method does not always yield the optimal node selection. By prioritizing certain nodes, we aimed to ensure that each step of the selection process was more accurate and beneficial to the overall system.

In particular, we define a node as a *danger* node if it is in the OUT state, does not have any IN state neighbors, and satisfies $\neg noDpNbr_2$. If such a node transitions to the IN state, it could negatively impact nodes exactly 2 hops away, causing them to transition to the OUT state. Therefore, if a node has a neighbor that meets these criteria, we define the node as having a $DangerNbr_3$, which requires 3-hop neighboring information.

To address this, we revised rule R1 by splitting it into two separate conditions, while leaving the remaining rule R2 unchanged, but renumbering it as rule R3. The new predicate definition was structured as follows:

- $DangerNbr_3(v) \equiv \exists w \in N(v) : w.state = \text{OUT} \wedge noInNbr(w) \wedge \neg noDpNbr_2(w)$
- $PriorNbr_4(v) \equiv |\{w \in N(v) : w.state = \text{OUT} \wedge DangerNbr_3(w) \wedge noInNbr(v) \wedge$

doi:10.6342/NTU202401576

$noDpNbr_2(v)\}|$

- **R1: if** $v.state = \text{OUT} \wedge noInNbr(v) \wedge noDpNbr_2(v) \wedge DangerNbr_3(v)$

  then $v.state := \text{IN};$ // join S

- **R2: if** $v.state = \text{OUT} \wedge noInNbr(v) \wedge PriorNbr_4(v) = 0 \wedge noBtNbr_2(v)$

  then $v.state := \text{IN};$ // join S

Despite this improved methodology, it did not consistently yield the desired results. However, this attempt provided valuable insights into the challenges of optimizing self-stabilizing algorithms.

6.2 Conclusion

In this thesis, we have demonstrated that the number of moves and rounds in any protocol is inherently proportional to the number of nodes, primarily due to the significant challenge of breaking graph symmetry. Consequently, previous protocols that began with an arbitrary state required stabilization processes tied to the total number of nodes. In contrast, our approach begins with a stabilized graph, assuming only k nodes have been altered.

Our primary contribution is the development of two self-stabilizing algorithms in distance-4 and distance-2 models. These algorithms achieve error correction within 3 hops under a distributed scheduler, enabling stabilization proportional to the number of altered nodes, k. This enhancement significantly improves the robustness and efficiency of self-stabilizing protocols, particularly in applications where reducing the impact of localized changes is crucial.

doi:10.6342/NTU202401576

Starting from any configuration, as in previous works, our first protocol ensures stabilization in at most n moves, as demonstrated by Algorithm 1. Furthermore, we introduced a second algorithm, which not only operates under the same standard as the most recent related study [3], but also incorporates an effective local error correction mechanism. Although the second algorithm gathers information from a shorter distance, the required number of moves and rounds is only a constant factor higher compared to the first algorithm.

doi:10.6342/NTU202401576

doi:10.6342/NTU202401576

References

[1] L. Bao and J. J. Garcia-Luna-Aceves. Topology management in ad hoc networks. In Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing, pages 129–140, 2003.

[2] D. Chalupa. An order-based algorithm for minimum dominating set with application in graph mining. Information Sciences, 426:101–116, 2018.

[3] W. Y. Chiu and C. Chen. Linear-time self-stabilizing algorithms for minimal domination in graphs. In International Workshop on Combinatorial Algorithms, pages 115–126. Springer, 2013.

[4] W. Y. Chiu, C. Chen, and S.-Y. 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.

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

[6] X. Ding, J. Chen, and J. Qiu. A fast and differentiable optimization model for finding the minimum dominating set in large-scale graphs. Journal of Computational Science, 73:102146, 2023.

[7] Y. Fan, Y. Lai, C. Li, N. Li, Z. Ma, J. Zhou, L. J. Latecki, and K. Su. Efficient local

doi:10.6342/NTU202401576

search for minimum dominating sets in large graphs. In International Conference on Database Systems for Advanced Applications, pages 211–228. Springer, 2019.

[8] M. Gairing, W. Goddard, S. T. Hedetniemi, P. Kristiansen, and A. A. McRae. Distance-two information in self-stabilizing algorithms. Parallel Processing Letters, 14(03n04):387–398, 2004.

[9] M. R. Garey and D. S. Johnson. Computers and intractability, volume 174. freeman San Francisco, 1979.

[10] W. Goddard, S. T. Hedetniemi, D. P. Jacobs, P. K. Srimani, and Z. Xu. Self-stabilizing graph protocols. Parallel Processing Letters, 18(01):189–199, 2008.

[11] 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 & Mathematics with Applications, 46(5-6):805–811, 2003.

[12] F. Molnár, S. Sreenivasan, B. K. Szymanski, and G. Korniss. Minimum dominating sets in scale-free network ensembles. Scientific reports, 3(1):1736, 2013.

[13] A. Potluri and C. Bhagvati. Novel morphological algorithms for dominating sets on graphs with applications to image analysis. In Combinatorial Image Analaysis: 15th International Workshop, IWCIA 2012, Austin, TX, USA, November 28-30, 2012. Proceedings 15, pages 249–262. Springer, 2012.

[14] H. Samuel, W. Zhuang, and B. Preiss. Dtn based dominating set routing for manet in heterogeneous wireless networking. Mobile Networks and Applications, 14(2):154–164, 2009.

[15] C. Shen and T. Li. Multi-document summarization via the minimum dominating set.

doi:10.6342/NTU202401576

In Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010), pages 984–992, 2010.

[16] S. Y. Tsai. An efficient self-stabilizing algorithm for the minimal dominating set problem under a distributed scheduler. http://hdl.handle.net/11536/47522, 2011.

[17] V. Turau. Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler. Information Processing Letters, 103(3):88–93, 2007.

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

[19] Z. Xu, S. T. Hedetniemi, W. Goddard, and P. K. Srimani. A synchronous self-stabilizing minimal domination protocol in an arbitrary network graph. In Distributed Computing-IWDC 2003: 5th International Workshop, Kolkata, India, December 27-30, 2003. Proceedings 5, pages 26–32. Springer, 2003.

[20] B. Yao and L. Fei-Fei. Action recognition with exemplar based 2.5 d graph matching. In Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part IV 12, pages 173–186. Springer, 2012.

[21] D. Zhao, G. Xiao, Z. Wang, L. Wang, and L. Xu. Minimum dominating set of multiplex networks: Definition, application, and identification. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 51(12):7823–7837, 2020.

doi:10.6342/NTU202401576