Ensembles indépendants et au-delà, à travers le prisme des systèmes distribués et des graphes colorés

Jonas Sénizergues

université
PARIS-SACLAY

Independent sets and beyond, through the prism of distributed systems and colored graphs

Ensembles indépendants et au-delà, à travers le prismes des systèmes distribués et des graphes colorés

Thèse de doctorat de l'université Paris-Saclay

École doctorale : n°580 : sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat: Informatique
Graduate School : Informatique et sciences du numérique, Référent : Faculté des sciences d'Orsay

Thèse préparée dans l'unité de recherche Laboratoire Interdisciplinaire des Sciences du Numérique (Université Paris-Saclay et CNRS), sous la direction de Yannis MANOUSSAKIS, Professeur des universités, et de Johanne COHEN, directrice de recherche.

Thèse soutenue à Paris-Saclay, le 12 Décembre 2022, par

Jonas SÉNIZERGUES

Composition du jury

Devan SOHIER Professeur des universités, LI-PaRAD, Université de Versailles Saint-Quentin-en-Yvelines	Président
Jérémie CHALOPIN Directeur de recherche, LIS, Université d'Aix-Marseille et CNRS	Rapporteur & Examinateur
Pierre FRAIGNIAUD Directeur de recherche, IRIF, Université Paris Cité et CNRS	Rapporteur & Examinateur
Swan DUBOIS Maître de conférence, LIP6, Sorbonne Université	Examinateur

THESE DE DOCTORAT

NNT : 2022UPASG091

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Résumé étendu en français:

On peut faire remonter l'histoire de l'étude des graphes jusqu'au 18ème siècle lorsque Euler proposa le problème des Sept ponts de Königsberg : "Est-il possible de trouver un chemin qui passe exactement une fois par chacun des ponts de Königsberg ?". Dans Königsberg, vue comme un graphe dans le formalisme moderne, les ponts seraient les arêtes, et chaque rive ainsi que chaque île un sommet. Si la réponse avait été "oui", au vu de la taille du graphe, la question aurait très bien pu ne jamais être exprimée par Euler, et la théorie des graphes pourrait avoir attendu quelques temps encore pour apparaitre dans le paysage mathématique. Mais la réponse était "non", et prouver que quelque chose n'est pas possible est souvent plus dur que prouver que quelque chose l'est. Euler introduit donc le premier raisonnement que l'on peut lier explicitement à la théorie des graphes, ainsi que la notion éponyme de *chemin Eulerien*. Depuis l'époque d'Euler, le monde est devenu de plus en plus connecté. Et si les graphes étaient déjà utiles pour modéliser des problèmes géographiques en en son temps, ils sont devenus un outil de modélisation majeur du monde moderne. Cependant, lorsqu'on veut modéliser des choses complexes, il est souvent nécessaire d'ajouter de l'information à la modélisation, sur les sommets, les arêtes, ou les deux. Ainsi on peut s'intéresser aux graphes auxquels on a ajouté des couleurs sur les arêtes ou les sommets, et aux problèmes qui émergent de cet ajout. C'est l'une des facettes des graphes abordées dans cette thèse. Les graphes apparaissent aussi très naturellement dans le contexte des réseaux d'ordinateurs pour représenter le graphe de communication entre les membres dudit réseau. Le domaine d'étude sur les calculs qui peuvent être faits sur de tels réseaux est appellé *calcul distribué*. En pratique dans un tel réseau, on ne peut pas toujours garantir que tous les membres calculent et communiquent à la même vitesse, et les proposition d'algorithme dans ce contexte doivent prendre en compte cette nature asynchrone pour capturer cette réalité. Dans certains cas, on ne peut même pas garantir que toutes les machines se comportent correctement : elles peuvent faire des erreurs. Deux types sont généralement considérés. D'une part il y a les pannes transitoires qui sont la conséquence d'une erreur ponctuelle, possibilité capturée dans la notion d'algorithme auto-stabilisant qui peut récupérer de n'importe quelle erreur de ce type. De l'autre, les pannes dites Byzantines sont le résultat d'un comportement malveillant et ne sont pas bornées dans le temps. Le travail présenté par cette thèse peut être divisé en deux parties, la première se concentrant sur l'autostabilisation dans des systèmes distribués, la seconde sur de l'algorithmique de graphe colorés. La partie portant sur l'autostabilisation s'intéresse aux pannes Byzantines pour des problèmes qui n'avaient pas d'algorithme connu supportant celles-ci. L'un d'eux est ensuite utilisé pour proposer un mécanisme produisant des algorithme autostabilisants pour tout problème raccommodable dans des réseaux anonymes. La partie d'algorithmique de graphes introduit un nouveau problème étendant des travaux antérieurs sur les couplages colorés et donne un résultat de difficulté algorithmique ainsi qu'un algorithme FPT pour un certain paramètre. Le chapitre 3 introduit un algorithme qui supporte les pannes Byzantines et résout le problème de l'indépendant maximal dans les systèmes anonymes en $O(n^2)$ rounds avec forte probabilité sous le démon distribué juste. Il donne ensuite une version légèrement modifiée de cet algorithme qui résout le même problème sous le démon distribué antagoniste (sans supporter de pannes Byzantines) en $O(n^2)$ opérations. Le chapitre 5 introduit un algorithme qui supporte les pannes Byzantines et résout le problème de partition minimales en cliques en $O(\Delta n)$ rounds sous le démon distribué juste dans des systèmes à identifiants uniques. Le chapitre 4 introduit un algorithme qui résout le problème du $(k, k-1)$-ensemble dirigeant dans les réseaux anonymes sous le démon Gouda. La construction en parallèle de tels ensembles dirigeants permet de trouver une coloration à distance K, dont on utilise les couleurs comme identifiants pour résoudre n'importe quel problème raccommodable sur des réseaux anonymes. Enfin, le chapitre 6 introduit un nouveau problème, le problème du couplage maximum minimalement coloré, qui étend des travaux antérieurs sur les couplages colorés. Ce problème est ici démontré NP-dur, et difficile à approximer sous un ratio logarithmique de la taille du graphe. Il y est également démontré qu'il $W[2]$-difficile en considérant le paramètre "taille de la solution", mais FPT en considérant le paramètre "taille d'un couplage maximum".

Title: Independent sets and beyond, through the prism of distributed systems and colored graphs.

Keywords: graphs, distributed, algorithms, self-stabilizing, complexity, approximation

Abstract: The work presented in this thesis can be divided in two, the first part focusing on self-stabilization in distributed systems, and the second one on classical graph algorithms. The self-stabilization part deals with Byzantine faults for problems that had no prior algorithm handling those. One of them is then used to propose a way to produce self-stabilizing algorithms for mendable problems in anonymous networks. The classical graph algorithm part introduces a new problem that extends some previous work on colored matchings and gives a hardness result as well as an FPT algorithm for a specific parameter. Chapter 3 introduces an algorithm that handles Byzantine faults and solves the MIS problem in anonymous systems in $O(n^2)$ rounds with high probability under the fair distributed daemon. It then gives a slightly modified version of this algorithm, that solves the same problem under the adversarial distributed daemon (without handling Byzantine faults) in $O(n^2)$ moves. Chapter 5 introduces an algorithm that handles Byzantine faults and solves the Minimal Clique Decomposition problem in $O(\Delta n)$ rounds under the fair distributed daemon in systems with unique identifiers. Chapter 4 introduces an algorithm that solves the $(k, k-1)$-ruling set problem in anonymous networks under the Gouda daemon. The parallel construction of multiple such ruling sets allows to find a distance-K coloring in an anonymous network, whose colors are used as identifiers to solve any mending problems on anonymous networks. Finally, Chapter 6 introduces a new problem, the Minimum Colored Maximum Matching problem, that extends what had already been done on colored matchings. The problem is shown to be NP-hard and hard to approximate within a logarithmic ratio of the size of the graph. It is also proven $W[2]$-hard with the parameter "size of the solution", but fixed-parameter tractable with the parameter "size of a maximum matching".

Titre: Ensembles indépendants et au-delà, à travers le prismes des systèmes distribués et des graphes colorés.

Mots clés: graphes, distribué, algorithmes, autostabilisation, complexité, approximation

Résumé: Le travail présenté par cette thèse peut être divisé en deux parties, la première se concentrant sur l'autostabilisation dans des systèmes distribués, la seconde sur de l'algorithmique de graphe colorés. La partie portant sur l'autostabilisation s'intéresse aux pannes Byzantines pour des problèmes qui n'avaient pas d'algorithme connu supportant celles-ci. L'un d'eux est ensuite utilisé pour proposer un mécanisme produisant des algorithme autostabilisants pour tout problème raccommodable dans des réseaux anonymes. La partie d'algorithmique de graphes introduit un nouveau problème étendant des travaux antérieurs sur les couplages colorés et donne un résultat de difficulté algorithmique ainsi qu'un algorithme FPT pour un certain paramètre. Le chapitre 3 introduit un algorithme qui supporte les pannes Byzantines et résout le problème de l'indépendant maximal dans les systèmes anonymes en $O(n^2)$ rounds avec forte probabilité sous le démon distribué juste. Il donne ensuite une version légèrement modifiée de cet algorithme qui résout le même problème sous le démon distribué antagoniste (sans supporter de pannes Byzantines) en $O(n^2)$ opérations. Le chapitre 5 introduit un algorithme qui supporte les pannes Byzantines et résout le problème de partition minimales en cliques en $O(\Delta n)$ rounds sous le démon distribué juste dans des systèmes à identifiants uniques. Le chapitre 4 introduit un algorithme qui résout le problème du $(k, k-1)$-ensemble dirigeant dans les réseaux anonymes sous le démon Gouda. La construction en parallèle de tels ensembles dirigeants permet de trouver une coloration à distance K, dont on utilise les couleurs comme identifiants pour résoudre n'importe quel problème raccommodable sur des réseaux anonymes. Enfin, le chapitre 6 introduit un nouveau problème, le problème du couplage maximum minimalement coloré, qui étend des travaux antérieurs sur les couplages colorés. Ce problème est ici démontré NP-dur, et difficile à approximer sous un ratio logarithmique de la taille du graphe. Il y est également démontré qu'il $W[2]$-difficile en considérant le paramètre "taille de la solution", mais FPT en considérant le paramètre "taille d'un couplage maximum".

Remerciements

Après quatre ans -un peu plus que prévu, oups-, voici venu le temps de clore cette page de ma vie que constitue la thèse. Et avec ça, celui de remercier tous ceux qui ont contribué de près ou de loin à rendre cette aventure envisageable, possible, et agréable.

Ainsi, pour m'avoir formé jusqu'à un niveau menant au début de cette aventure, tout en ayant affûté mon goût de la subtilité mathématique, je remercie mes enseignants de CPGE, de l'ENS de Cachan, et encadrants de stage : Guillaume, Laurent, Denis, Hubert, Michel, Enrico, et Reynald.

Pour m'avoir donné la possibilité de travailler avec eux pendant trois -oups, quatre- ans alors que ma situation était un peu particulière suite à quelques accidents de la vie, je remercie mes encadrants de thèse Yannis et Johanne.

Pour le temps passé autour d'un tableau blanc pendant des durées extensives jusqu'à réussir à mettre des raisonnements bout-à-bout, je remercie mes coauteurs Jérôme, Laurence, Mickael, François, et bien sûr Yannis et Johanne. Parmi ceux-ci, certains ne pourront jamais lire ces lignes, car ils sont partis trop tôt, Jérôme et Yannis. Et si pour partie les travaux correspondant n'ont pas pu trouver leur place dans ce manuscrit, j'espère pouvoir finir les travaux que nous avions entamé ensemble.

Pour les discussions scientifiques et beaucoup moins scientifiques, la boite à chocolat, et les arts décoratifs sur tableau, je remercie ceux avec qui j'ai partagé le bureau 35 pendant ces années : Justine, Hugo, Balthazar, et Daniel. Pour seulement la partie discussions -on ne peut pas partager le chocolat avec tout le monde-, je remercie également tous les membres de l'équipe GALaC du LISN.

Pour les expériences positives d'enseignement, je remercie les responsables de cours et co-chargés de TD avec qui j'ai pu enseigner à l'Université Paris-Saclay : Frédéric et Laurent, Pierre et Marie, et à l'ENSIIE : Julien, Dimitri, Christophe et Kahina.

Pour s'être rendus disponibles pour participer au jury de thèse, et pour avoir pris le temps de parcourir l'intégralité de mon manuscrit dans le cas des rapporteurs, je remercie les membres du jury Pierre Fraigniaud, Jérémie Chalopin, Devan Sohier, et Swan Dubois.

Et si la thèse est l'aboutissement d'un processus académique, pour rendre vivable celui-ci le versant non-académique est essentiel. Je remercie tous mes amis, dont je n'ose pas essayer faire la liste de peur d'en oublier. En particulier : l'équipe des gros dej' du dimanche qui m'a largement servi de cobaye pour des expérimentations culinaires et pour avoir maintenu lien social régulier en période difficile ; les membres de la guilde Evolution qui ont écouté mes jérémiades lorsque

j'avais besoin de lâcher de la pression, et m'ont accompagné me nouer le cerveau lorsque j'avais besoin de changer d'air ; les membres des club anime et nanar pour les projections à valeur culturelle variable mais toujours divertissantes.

Merci à ceux qui m'ont permis de profiter d'une accalmie pour la fin de la rédaction de ce manuscrit, lorsque la quiétude se faisait rare : Catherine et Christian, ainsi que Chloé.

Merci enfin à ma famille pour m'avoir accompagné et soutenu, depuis le début, et bien avant encore : Geneviève, Philipe, Rose-Claire, Irène, Géraud, Delphin et Loup.

Contents

1 - Introduction

Before graphs became the staple of modeling that they now are, their history can be traced back to the 18th century, when Leonardo Euler proposed the *Seven bridges of Königsberg* problem: "Is it possible to devise a path in Königsberg that crosses each bridge exactly once?". Bridges would be edges of the graph, while each bank of the river and each island would be a vertex of the graph, in nowadays formalism. Had the answer to this question been "yes", given the size of the graph, the question may not even have been asked by Euler, and graph theory could have waited some decades more to appear in the mathematical landscape. But the answer was "no", and proving that something is *not* possible is often harder than proving that something is. Thus Euler introduced the problem and the first reasoning that we can link explicitly to graph theory, as well as the eponymous notion of *Eulerian path*.

Since Leonardo Euler's time, the world has become more and more interconnected. If graphs were already useful to model geographical problems in his time, they have become a widespread modeling tool in the contemporary world, justifying the study of fundamental graph theory. However, when one wants to model a complex thing, there is often the need to add some information to the modeling, either on the vertices, the edges, or both. For example, with road networks, a natural way to model them is to use edge-weighted graphs, where vertices represent locations, and edges roads, with the weights representing either the length of the road, or its maximum traffic capacity, depending on the problem you are dealing with. For some applications, we might want to use a qualitative -instead of quantitative- way of adding information to the graph. For example, modeling a social graph might benefit from adding colors to the edges, corresponding to the type of relationship represented. For the study of the graph of webpages and its patterns, it may be more useful to add colors to the vertices, corresponding to the type of content of the page. This last kind of graph, namely vertex-colored graphs, is considered in Chapter 6.

Graphs also appear very naturally in the context of computer networks as the underlying communication graph between the said computers. The field of study of calculations that can be made on such a network is called distributed computing. In such a context, a "computer" is a calculation unit that is only aware of its immediate neighborhood in the communication graph, and information has to be communicated between units to reach a solution. Note that as many such machines are supposed to work together in a network, one cannot always guarantee that every one of them computes and communicates at the same speed, and attempts at proposing algorithms for such a network should take into account

this asynchronous nature of the global computation to succeed. Sometimes, you cannot even guarantee that every machine in the network is behaving perfectly well: they can be subject to some faults. Two types of faults are generally considered. On one hand, transient faults are the result of a one-time computation error, and this is captured by the notion of self-stabilizing algorithms that are able to recover from any such fault [24, 26]. On the other hand, Byzantine faults are the result of malevolent behavior, and cannot be guaranteed to stop at any time [48]. In Chapters 3,4 and 5 we consider self-stabilizing algorithms, and in Chapters 3 and 5 we include considerations about Byzantine faults.

In Chapter 2, notions used throughout the thesis about mathematics, graph theory, and distributed systems are introduced.

In Chapter 3, in collaboration with Johanne Cohen, Laurence Pilard, and François Pirot, we present two algorithms to compute a maximal independent set in an anonymous network. The first one, working under the fair distributed daemon, is robust to Byzantine faults. The second one works under the adversarial distributed daemon.

In Chapter 4, in collaboration with Johanne Cohen and Mikaël Rabie, we introduce an algorithm to compute a ruling set in an anonymous network under the Gouda daemon. We then use it to introduce a general process to solve any mendable problem (a generalization of locally greedy problems).

In Chapter 5, in collaboration with Johanne Cohen and Laurence Pilard, we give an algorithm that tackles the Clique Decomposition problem in a (non-anonymous) network while handling Byzantine faults.

In Chapter 6, in collaboration with Johanne Cohen and Yannis Manoussakis, we study the hardness to find a maximum matching that uses the minimum number of colors in a vertex-colored graph. On one hand, we show that it is $W[2]$-hard using the number of colors of the solution as a parameter, and hard to approximate. On the other hand, we show that it is fixed-parameter tractable with the size of a maximum matching as a parameter.

2 - Graphs and models

In this chapter, we introduce many notions and notations that we will use in the remainder of this thesis. Some of them may be already well-known to the reader, but we want to remain reasonably exhaustive for the sake of the accessibility of this document.

2.1 . Sets

As we use multiple types of sets in this thesis, we introduce them here beforehand. Recall that the notion of set captures the idea of a collection of objects called elements without order or repetition. If e is an element of a set A we write $e \in A$ for "e is a member of A". We use multiple methods to introduce and manipulate sets:

- The enumeration notation : $S = \{a, b, c\}$ is the set containing exactly the elements a, b and c.
- The semantic description : "S is the set containing the letters in the word *notation*" stands for $S = \{$'n', 'o', 't', 'a', 'i', 'n'$\}$.
- The set-builder notation : $S = \{e \in S' \mid \mathcal{P}(e)\}$ is the set containing all elements e of the set S', such that the predicate $\mathcal{P}$ is true on e. When S' is obvious from context, the part $\in S'$ can be omitted. More generally, e may be replaced by any function of e.
- And of course, we use operations on pre-defined sets to define new sets, including the basic operations written as follows when A and B are two sets:
  - The union $A \cup B$ is $\{e \mid e \in A \vee e \in B\}$.
  - The intersection $A \cap B$ is $\{e \mid e \in A \wedge e \in B\}$.
  - The set difference $A \setminus B$ is $\{e \mid e \in A \wedge e \notin B\}$.
  - The cartesian product $A \times B$ is $\{(a, b) \mid a \in A \wedge b \in B\}$.

Note that $\{f(e), g(e) \mid \mathcal{P}(e)\}$ is an alias for $\{f(e) \mid \mathcal{P}(e)\} \cup \{g(e) \mid \mathcal{P}(e)\}$.

The cardinal of a set S, also called its size, is denoted by $|S|$.

We use the standard notations for real numbers intervals. When $(x, y) \in \mathbb{R}$:

- $[x, y]$ is $\{r \in \mathbb{R} \mid x \leq r \leq y\}$,
- $]x, y]$ is $\{r \in \mathbb{R} \mid x < r \leq y\}$,

- $[x, y[$ is $\{r \in \mathbb{R} \mid x \leq r < y\}$,
- $]x, y[$ is $\{r \in \mathbb{R} \mid x < r < y\}$.

We also use the standard notation for integer segments. When $(x, y) \in \mathbb{Z}$, $[\![x, y]\!]$ is the set $\{r \in \mathbb{Z} \mid x \leq r \leq y\}$.

Recall the canonical order on sets: the inclusion. A set A is included in a set B if every element of A is also in B. It is written $A \subseteq B$. $A \subsetneq B$ is an alias for $A \subseteq B \wedge A \neq B$.

Definition 2.1.1 (Partition)**.** A partition of a set A is a set $\mathcal{A}$ of parts of A (*i.e.* elements of $\mathcal{A}$ are subsets of A) such that:

- The union of all elements of $\mathcal{A}$ is A,
- The elements of $\mathcal{A}$ are pairwise disjoints (*i.e.* their intersection is empty).

2.2 . Graph notions and notations

In this thesis, we only consider simple undirected graphs, which we call, for the sake of simplicity, "graphs".

A graph G is a couple (V, E), where V is called the set of vertices, and E called the set of edges is a set of (unordered) pairs of elements of V. The set of vertices (resp. edges) of G may also be denoted by $V(G)$ (resp. $E(G)$). By convention, we write $n = |V|$ and $m = |E|$.

When u and v are vertices of the graph, and $\{u, v\} = \{v, u\} \in E$ is an edge, we write that edge uv. u and v are called the *endpoints* of the edge uv.

Two vertices u and v are said *adjacent* when $uv \in E$. We also say that u and v are *neighbors*. Two edges are said *adjacent* when they share an endpoint.

When u is a vertex, $N(u)$ is the set of vertices that are neighbors of u, it is called the *neighborhood* of u. The *closed neighborhood* $N[u]$ is then $N(u) \cup \{u\}$. The degree of a node u is $|N(u)|$, written $deg(u)$. The maximum degree in the graph is denoted by Δ.

A graph $G' = (v', E')$ is said to be a *subgraph* of $G = (V, E)$ when $V' \subseteq V$ and $E' \subseteq E$. Moreover, G' is said to be an *induced* subgraph when it is a subgraph and E' contains every edge of E whose both endpoints are in V'. In such a case, we say that G' is the subgraph of G induced by V'.

Definition 2.2.1 (Path)**.** A *path* of length k, or P_k, is a graph with $k + 1$ distinct vertices on which there exists a total order relation such that, when you write the vertices in that order, the edges are exactly the pairs of successive vertices.

By extension, a finite sequence of distinct vertices $x_0, x_1, ..., x_k$ of a graph G is a *path* of G when $G' = (\{x_0, x_1, ..., x_k\}, \{x_0x_1, x_1x_2, ..., x_{k-1}x_k\})$ is a subgraph

of G. We denote by $dist(u, v)$ the distance between two vertices u and v *i.e.* the minimal length of a path from u to v in the graph. When S is a subset of V, $dist(u, S)$ is the minimum among all the distances from u to an element in S.

Definition 2.2.2 (Cycle)**.** A *cycle* of length $k \geq 3$, or P_k, is a graph with k distinct vertices on which there exists a total order relation such that, when you write the vertices in that order, the edges are exactly the pairs of successive vertices plus an edge between the smallest and the biggest vertices.

By extension, a finite sequence of distinct vertices $x_0, x_1, ..., x_k$ of a graph G is a *cycle* of G when $G' = (\{x_0, x_1, ..., x_k\}, \{x_0x_1, x_1x_2, ..., x_{k-1}x_k, x_kx_0\})$ is a subgraph of G.

Definition 2.2.3 (Tree)**.** A *tree* is a graph that has no cycle.

Definition 2.2.4 (Clique)**.** A *clique* of size k, is a graph with k distinct vertices where every vertex is adjacent to every other one.

By extension, in a graph G, we say that $C = \{x_0, x_1, ..., x_k\}$ is a clique of G (with $x_0, x_1, ..., x_k$ distinct vertices of G) when the subgraph of G induced by C is a clique.

2.3 . Models of distributed systems

In Chapters 3, 4 and 5 we deal with problems in distributed systems, to which we give solutions that are robust to transient faults. In this part, we introduce the model we use here, some related notions, and some useful notations.

A distributed system consists of a set of processes where two adjacent processes can communicate with each other. The communication relation is modeled by a graph $G = (V, E)$ where V is the set of the processes (we call *node* an element of V in such context) and E represents the neighborhood relation between them, *i.e.* $uv \in E$ when u and v are adjacent nodes.

In the context of distributed systems, many different assumptions can be made about the way nodes are able to interact with their neighbors. In Chapters 3 and 4 we assume the system to be *anonymous*, which means that the nodes do not have unique identifiers to distinguish themselves. They can, however, point toward a node in their neighborhood, and can tell if a given neighbor is pointing toward them or not. Conversely, in Chapter 5 we suppose that every node has a unique identifier

Regarding communication and local information, we use what is called the *state model*. Every node has a set of *local variables* which make up the *local state* of the node. A node can read its local variables and all the local variables of its neighbors, but can only rewrite its own local variables.

2.3.1 . Rules, transitions, and executions

Now that we know what the system looks like, it remains to define what is a computation in it. First of all, we have to define what corresponds to a picture of the system at a given time. A *configuration* is the value of the local states of all nodes in the system. When u is a node and x a local variable, the x-*value* of u in configuration γ is the value x_u^γ.

Then, to define how such a system can evolve, we introduce the notion of *rule*, which may be executed by a given node of the system. A rule is anything of the shape $\langle guard \rangle \to \langle command \rangle$, parametrized by a node, where:

- The *guard* is a predicate over the variables of the said node and its neighbors.
- The *command* is a sequence of actions that may change the values of the node's variables (but not those of its neighbors).

Notice how the notion of guard implements the fact that nodes can read the local variables of their neighbors.

A rule is *enabled* on a node u in a configuration γ if the guard of the rule holds on u in γ. A node is *activable* on a configuration γ if at least one rule is enabled on u. We say that a configuration is *stable* if no node is activable in that configuration.

We call *move* any couple (u, r) where u is a node and r is a rule. A move is said *valid* in a given configuration γ if r is enabled on u in γ.

The execution of a rule by a node may only change the value of variables of that specific node, but multiple moves may be performed at the same time, as long as they act on different nodes. To capture this, we say that a set of moves t is *valid* in a configuration γ when it is non-empty, contains only valid moves of γ, and does not contain two moves with the same node as first element. Then, a *transition* is a triplet (γ, t, γ') such that:

- t is a valid set of moves of γ,
- γ' is a possible configuration after every node u appearing in t performed simultaneously the command of the associated rule, beginning in configuration γ.

Note that here we implicitly implement *rule atomicity*, which means that evaluating the validity of the guard and executing the command (which possibly contains multiple actions) is made as a single action. We write such a triplet as $\gamma \xrightarrow{t} \gamma'$. We also write $\gamma \to \gamma'$ when there exists a transition from γ to γ'. $V(t)$ denotes the set of nodes that appear as first member of a couple in t.

We say that a rule r is *executed* on a node u in a transition $\gamma \xrightarrow{t} \gamma'$ (or equivalently that the move (u, r) is *executed* in $\gamma \xrightarrow{t} \gamma'$) when the node u has

performed the rule r in this transition, that is when $(u, r) \in t$. In this case, we say that u has been *activated* in that transition. Then, an *execution* is an alternate sequence of configurations and move sets $\gamma_0, t_1, \gamma_1 \cdots t_i, \gamma_i, \cdots$ where:

- The sequence is either infinite or ends by a configuration,
- For all $i \in \mathbb{N}$ such that it is defined, $(\gamma_i, t_{i+1}, \gamma_{i+1})$ is a transition.

We write such an execution as $\gamma_0 \xrightarrow{t_1} \gamma_1 \cdots \xrightarrow{t_i} \gamma_i \cdots$ When the execution is finite, the last element of the sequence is the *last configuration* of the execution. An execution is *maximal* if it is infinite, or it is finite and no node is activable in the last configuration. It is called *partial* otherwise. We say that a configuration γ' is *reachable* from a configuration γ if there exists an execution starting in configuration γ that leads to configuration γ'.

2.3.2 . Daemons

In a distributed system, the idea is that you can give instructions to individual nodes, but cannot, in general, ensure synchronization (neither the time synchronization nor the synchronization of the choices) of the executions of rules between different parts of the system. An *algorithm* in such a distributed system is then a set of rules, that are local rules as defined earlier.

Then, you have to introduce a notion to formalize that uncertainty about what the nodes do. This is done by introducing an adversary, called *daemon*, that chooses from a given configuration which moves to execute in the next transition. Depending on the powers we give to that adversary, it will result in different constraints on the algorithm we build to be able to solve a given problem. A daemon is formally a predicate on the executions, only allowing a certain subset of executions.

The most general daemon has no constraint at all, it is formally the predicate $true$. It is often called the *adversarial distributed daemon*, a naming that corresponds to two natural ways to classify daemons:

- A *fair* daemon is a daemon that only allows executions with fairness property: every node that is continuously activable must eventually be activated. An *adversarial* daemon is on the contrary a daemon that does not have such a constraint.
- A *synchronized* daemon is a daemon that only allows executions where every activable node is activated in every transition, leaving only the choice of the rule. A *distributed* daemon is on the contrary a daemon that does not have such a constraint.

In the first part of Chapter 3 and in Chapter 5 we work under the adversarial distributed daemon. In the second part of Chapter 3 we use the fair distributed

daemon. As we only consider distributed daemons in this thesis, we write *the fair daemon* (resp. **the adversarial daemon**) when dealing with the distributed fair daemon (resp. the distributed adversarial daemon).

In Chapter 4 we use a rather unconventional daemon called the *Gouda daemon*, which only allows executions such that a continuously reachable configuration must eventually be reached.

Definition 2.3.1. [29, 37] We say that an execution $\mathcal{E} = \gamma_0 \xrightarrow{t_1} \gamma_1 \cdots \xrightarrow{t_i} \gamma_i \cdots$ is *under the Gouda deamon* if for any configurations γ and γ' such that $\gamma \to \gamma'$ can be executed, if γ appears infinitely often in $\mathcal{E}$, then γ' also appears infinitely often in $\mathcal{E}$.

See [29] for a more complete taxonomy of daemons.

2.3.3 . Algorithm and self-stabilization

An algorithm is a set of *rules*, where each rule is of the form $\langle guard \rangle \to \langle command \rangle$ and is parametrized by the node where it would be applied. The *guard* is a predicate over the variables of the said node and its neighbors. The *command* is a sequence of actions that may change the values of the variables of the node (but not those of its neighbors).

The notion of *self-stabilization* is then to have algorithms that can recover from any transient fault. This is captured by the fact that in this context we require algorithms to be able to converge toward a correct solution from any starting configuration.

Given a specification of a problem and $\mathcal{L}$ the associated set of *legitimate configuration*, *i.e.*, the set of the configurations that verify the specification, an algorithm is *self-stabilizing* when the following properties are true:

- *Correctness*: every configuration of an execution starting by a configuration of $\mathcal{L}$ is in $\mathcal{L}$,
- *Convergence*: from any configuration, whatever the strategy of the daemon, the resulting execution eventually reaches a configuration in $\mathcal{L}$ with probability 1.

When the rules are deterministic, the convergence condition may often (depending on the daemon) be replaced by the same but without the "with probability 1" part.

2.3.4 . Byzantine faults

On top of transient faults, in distributed systems is often considered the possibility to have nodes that are secretly non-cooperative with the common goal of

the system and make intentional errors in the computation. We call this kind of node *Byzantine* nodes, and this kind of errors *Byzantine* errors or faults. Formally Byzantine nodes are modeled by special nodes that are always activable, and may change their state arbitrarily when activated.

In the presence of such Byzantine faults, it is often not possible to maintain the specification of a given problem. The goal becomes in such a case to contain the Byzantine node influence by finding a "solution" on a subgraph that avoids Byzantine nodes.

Note that with Byzantine nodes, you cannot hope for any algorithm that works under the adversarial distributed daemon, as the daemon may choose to always activate a Byzantine node and nothing else. It is then necessary to consider algorithms that work under a more constrained daemon. We give algorithms that work under the fair distributed daemon in the first part of Chapter 3 and in Chapter 5 for this reason.

2.3.5 . Complexities

The time complexity of a distributed algorithm may be evaluated by various metrics. Most of the time, in the context of distributed systems, the local computation time is not considered, and what is accounted for is communication. Here, in the state model, the communication lies in the reading of neighbors' variables when a node is activated, hence we want to count these events.

The most straightforward way to do this is to count the number of moves performed in one execution, this is what we do when we consider algorithms under the adversarial distributed daemon in the second part of Chapter 3.

However, in distributed systems, you may be interested in the speed of the "slowest" node, as events happen in parallel. This is especially true in the context of a fair daemon, as you can be sure that no node may be left aside by the daemon. This is captured by the notion of *round*. This concept was introduced by Dolev *et al.* [27], and reworded by Cournier *et al.* [18] to take into account activable nodes. We quote the two following definitions from Cournier *et al.* [18]: "

Definition 2.3.2. We consider that a node u executes a *disabling action* in the transition $\gamma_1 \to \gamma_2$ when:

- u is activable in γ_1,
- u does not execute any rule in $\gamma_1 \to \gamma_2$,
- u is not activable in γ_2.

The disabling action represents the situation where at least one neighbor of u changes its local state in $\gamma_1 \to \gamma_2$, and this change effectively makes the guard of all rules on u false in γ_2. The time complexity is then computed by capturing the speed of the slowest node in any execution through the round definition [27].

Definition 2.3.3. Given an execution $\mathcal{E}$, the first *round* of $\mathcal{E}$ (let us call it $\mathcal{R}_1$) is the minimal prefix of $\mathcal{E}$ containing the execution of one action (the execution of a rule or a *disabling action*) of every activable node from the initial configuration. Let $\mathcal{E}'$ be the suffix of $\mathcal{E}$ such that $\mathcal{E} = \mathcal{R}_1\mathcal{E}'$. The second round of $\mathcal{E}$ is the first round of $\mathcal{E}'$, and so on.

Observe that Definition 2.3.3 is equivalent to Definition 2.3.4, which is simpler in the sense that it does not refer back to the set of activable nodes from the initial configuration of the round.

Definition 2.3.4. Let $\mathcal{E}$ be an execution. A *round* is a sequence of consecutive transitions in $\mathcal{E}$. The first round begins at the beginning of $\mathcal{E}$; successive rounds begin immediately after the previous round has ended. The current round ends once every node $u \in V$ satisfies at least one of the following two properties:

- u has been activated in at least one transition during the current round,
- u has been non-activable in at least one configuration during the current round.

This is the notion of complexity that we use in the first part of Chapter 3 and in Chapter 5.

3 - Maximal Independent Set

An *independent set* I in a graph is a set of vertices such that no two of them form an edge in the graph. It is called *maximal* when it is maximal inclusion-wise (in which case it is also a minimal dominating set). Maximal independent sets have received a lot of attention in different areas. For instance, in wireless networks, the maximum independent sets can be used as a black box to perform communication (to collect or to broadcast information) (see [50, 32], for example). In self-stabilizing distributed algorithms, they are also a fundamental tool to transform an algorithm from one model to another [38, 64].

In Section 3.2 we give a self-stabilizing randomized algorithm with Byzantine nodes under the fair daemon, which converges in $O(\Delta n)$ rounds.

Then, in Section 3.3, we give a self-stabilizing randomized algorithm that finds a maximal independent set in an anonymous network, under the assumption of a distributed adversarial daemon (without Byzantine nodes). We show that our algorithm converges in $O(n^2)$ moves with high probability.

3.1 . State of the art

The maximal independent set (MIS) problem has been extensively studied in parallel and distributed settings, following the seminal works of [1, 49, 52]. Their idea is based on the fact that a node joins the "MIS under construction" S according to the neighbors: node v joins the set S if it has no neighbor in S, and it leaves the set S if at least one of its neighbors is in S. Most algorithms in the literature, including ours, are based on this approach.

The MIS problem has been extensively studied in the Local model, [33, 57, 14] for instance (a synchronous, message-passing model of distributed computing in which messages can be arbitrarily large) and in the Congest model [56] (synchronous model where messages are $O(\log n)$ bits long). In the Local model, Barenboim *et al.* [8] focus on identified system and give a self-stabilizing algorithm producing a MIS within $O(\Delta + \log^* n)$ rounds. Balliu *et al* [6] prove that the previous algorithm [8] is optimal for a wide range of parameters in the Local model. In the Congest model, Ghaffari *et al.* [34] prove that there exists a randomized distributed algorithm that computes a maximal independent set in $O(\log \Delta \cdot \log \log n + \log^6 \log n)$ rounds with high probability.

Self-stabilizing algorithms for maximal independent set have been designed in various models (anonymous network [59, 64, 63] or not [36, 42]). Up to our knowledge, Shukla *et al.* [59] present the first algorithm designed for finding a MIS in a

graph using self-stabilization paradigm for anonymous networks. Some other self-stabilizing works deal with this problem assuming identifiers: with a synchronous daemon [36] or distributed one [42]. These two works require $O(n^2)$ moves to converge. Turau [62] improves these results to $O(n)$ moves under the distributed daemon. Recently, some works improved the results in the synchronous model. For non-anonymous networks, Hedetniemi [40] designed a self-stabilization algorithm for solving the problem related to dominating sets in graphs in particular for a maximal independent set that stabilizes in $O(n)$ synchronous rounds. Moreover, for anonymous networks, Turau [63] designs some Randomized self-stabilizing algorithms for maximal independent set w.h.p. in $O(\log n)$ rounds. See the survey [39] for more details on MIS self-stabilizing algorithms.

Some variants of the maximal independent set problem have been investigated, for example the 1-maximal independent set problem [61, 58] or Maximal Distance-k Independent Set [10, 45]. Tanaka *et al* [61] designed a silent self-stabilizing 1-MIS algorithm under the weakly-fair distributed daemon for any identified network in $O(nD)$ rounds (where D is a diameter of the graph).

3.2 . With Byzantines Nodes under the Fair Daemon

In this section, we focus on the construction of a MIS handling both transient and Byzantine faults. On one side, transient faults can appear in the whole system, possibly impacting all nodes. However, these faults are not permanent, thus they stop at some point in the execution. Self-stabilization [23] is the classical paradigm to handle transient faults. Starting from any arbitrary configuration, a self-stabilizing algorithm eventually resumes a correct behavior without any external intervention. On the other side, (permanent) Byzantine faults [48] are located on some faulty nodes and so the faults only occur from them. However, these faults can be permanent, *i.e.*, they could never stop during the whole execution.

The algorithm presented in this section builds a maximal independent set represented by a local variable s. The approach of the state of the art is the following: when two nodes are candidates to be in the independent set, then a local election decides who will remain in the independent set. To perform a local election, the standard technique is to compare the identifiers of nodes. Unfortunately, this mechanism is not robust to the presence of Byzantine nodes.

Keeping with the approach outlined above, when a node u observes that its neighbors are not in (or trying to be in) the independent set , the non-Byzantine node decides to join it with a certain probability. Randomization helps to reduce the impact of Byzantine nodes. The choice of probability should reduce the impact of Byzantine nodes while maintaining the efficiency of the algorithm.

Algorithm 3.2.1. Any node u has two local variables $s_u \in \{\bot, \top\}$ and $x_u \in \mathbb{N}$

and may make a move according to one of the following rules:
(Refresh) $x_u \neq |N(u)| \rightarrow x_u := |N(u)| \quad (= deg(u))$
(Candidacy?) $(x_u = |N(u)|) \wedge (s_u = \bot) \wedge (\forall v \in N(u), s_v = \bot) \rightarrow$
if $Rand(\frac{1}{1+\max(\{x_v | v \in N[u]\})}) = 1$ then $s_u := \top$
(Withdrawal) $(x_u = |N(u)|) \wedge (s_u = \top) \wedge (\exists v \in N(u), s_v = \top) \rightarrow s_u := \bot$

Observe since we assume an anonymous setting, the only way to break symmetry is randomization. The value of the probabilities for changing the local variable s must carefully be chosen in order to reduce the impact of the Byzantine node.

A node joins the MIS with a probability $\frac{1}{1+\max(\{x_v | v \in N[u]\})}$. The idea to ask the neighbors about their own number of neighbors (through the use of the x variable) to choose the probability of a candidacy comes from the mathematical property $\forall k \in \mathbb{N}, (1 - \frac{1}{k+1})^k > e^{-1}$, which will allow to have a good lower bound for the probability of the event "some node made a successful candidacy, but none of its neighbors did".

3.2.1 . Specification

Since Byzantine nodes are not bound to follow the rules, we cannot hope for a correct solution in the entire graph. What we wish to do is to find a solution that works when we are far enough from the Byzantine nodes. One could think about a fixed containment radius around Byzantine nodes, but as we can see later this is not as simple, and it does not work with our approach.

Let us define on any configuration γ the following set of nodes, that represents the already built independent set:

$$I_\gamma = \{u \in V_1 | (s_u^\gamma = \top) \wedge \forall v \in N(u), s_v^\gamma = \bot\}$$

Definition 3.2.2. We say that a node is *locally alone* if it is candidate to be in the independent set (*i.e.* its s-value is $\top$) while none of its neighbors are.

In configuration γ, I_γ is the set of all locally alone nodes of V_1.

Definition 3.2.3. A configuration γ is said *legitimate* when I_γ is a maximal independent set of $V_2 \cup I_\gamma$.

3.2.2 . An example

Figure 3.1 gives an example of an execution of Algorithm 3.2.1. Figure 3.1a depicts a network in a given configuration. The symbol drawn above the node represents the local variable s. Every non-Byzantine node, represented by circles, is supposed to have already its degree as x-value. The only Byzantine node in the exemple is represented by a square.

In the initial configuration, nodes v_1 and v_2 are in the independent set, and **Withdrawal** is enabled on them. In the first step, the daemon activates v_1 (**Withdrawal**) and v_2 (**Withdrawal**) leading to configuration γ_1 (Fig. 3.1b). In the second step, the daemon activates v_1 (**Candidacy?**). Node v_1 randomly decides whether to set $s_{v_1} := \top$ leading to configuration γ_2 (Fig. 3.1c), or $s_{v_1} := \bot$ leading to configuration γ_1 (Fig. 3.1b). Assume that v_1 chooses $s_{v_1} := \top$. At this moment, node v_1 is "locally alone" in the independent set. In the third step, the daemon activates b and b makes a Byzantine move setting $s_b := \top$, leading to configuration γ_3 (Fig. 3.1d).

In the fourth step, the daemon activates v_1 (**Withdrawal**) and b that sets $s_b := \bot$. The configuration is now the same as configuration γ_1 (Fig. 3.1b). The daemon is assumed to be fair, thus v_2 and v_3 need to be activated before the execution can be called an infinite loop. These activations will prevent v_1 to alternate forever between in and out of the independent set, while the rest of the system remains out of it. In the fifth step, the daemon activates v_1 (**Candidacy?**), v_2 (**Candidacy?**) and v_3 (**Candidacy?**). They randomly decide to change their local variable s. Assume that v_1, v_2 and v_3 choose $s_{v_1} := \top$, $s_{v_2} := \bot$, and $s_{v_3} := \top$, leading to configuration γ_5 (Fig. 3.1e). At this moment, node v_3 is "locally alone" in the independent set. As v_3 is far enough from the Byzantine node it will remain in the independent set whatever b does.

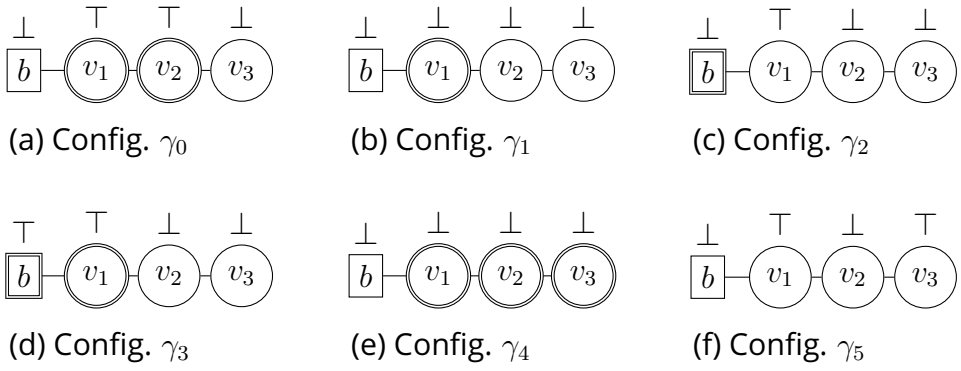


Figure 3.1: An example of execution. The square node is a Byzantine node, the double bordered nodes are those activated in the next transition.

About the specification

Our goal is to design an algorithm that builds a maximal independent set of the subgraph induced by a set of nodes where nodes "too close" to Byzantine nodes have been removed. The question now is to define what does "too close" mean. One could think about a fixed containment radius only excluding nodes at distance

at most 1 from Byzantine nodes. This set of nodes has been previously defined as V_1. Indeed, in Figure 3.2.2.(a), v_1 and v_2 belongs to V_1 and their local view of the system is correct, then they have no reason to change their states. Moreover, Byzantine nodes are too far away to change that: whatever the value of the state of v_0, the view of v_1 remains correct. Thus a containment radius of 1 could seem correct. However, in Figure 3.2.2.(b), if the Byzantine node does not make any move, then v_0 remains in the MIS while v_1 remains out of it. Thus, in this example, if we only consider nodes in V_1, the $\top$-valued nodes of V_1 are not a MIS of V_1. If V_1 is not always a good choice, neither is V_2. See Figure 3.2.2.(c), as one can see that all nodes in V_1 will never change their local state. The same can be said for V_k for any k, see example Figure 3.2.2.(d) for V_3. The solution is then to consider a set of nodes defined from a fixed containment radius to which we add locally alone neighboring nodes. The smallest containment radius that works with this approach is 3 (which corresponds to set V_2). Note that it depends on the current configuration and not only on the underlying graph.

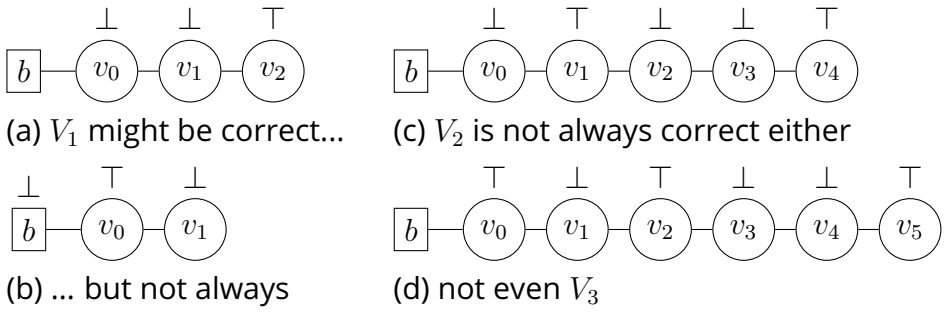


Figure 3.2: What is the good containment radius?

About the choice of probability to join the MIS

We could have gone with the same probability for every node, but that comes with the cost of making the algorithm very sensitive to the connectivity of the underlying graph. As we rely for convergence on the event where a node is candidate alone (*i.e.* switch to $\top$ without other node doing the same in its neighborhood), the probability of progress in a given number of rounds would then be exponentially decreasing with the degrees of the nodes.

We could have gone with something depending only on the degree of the node where the rule is applied. While it could have been an overall improvement over the uniform version above, the minoration of the probability of progress that can be made with a local scope is no better. We cannot exclude that a finer analysis would lead to a better overall improvement, but it would require to deal with far more complex math. On a smaller scale, we can also note that this choice

would introduce a bias toward small degree nodes, while we might not want that (depending on the application).

Then, we have chosen the version where nodes take into account their degree and what they know of the degree of their neighbors. On one hand, the first concern you could rise here would be the potential sabotage by Byzantine nodes. Here is the intuition of why this cannot be a problem here. If a node u is at distance at least 2 from any Byzantine node and if u is "locally alone" in the independent set, then whatever the Byzantine nodes do, u will forever remain in the independent set. To maximize the harm done, the Byzantine nodes have to prevent indirectly such a node to join the independent set. To do so it has to maximize the probability of its neighbors to be candidate to the independent set. But Byzantine nodes cannot lie efficiently in that direction, as the probability is upper-bounded by the degree values of both the node and its non-Byzantine neighbors. On the other hand, this choice allows us to address the problem that we had with the previous solution. Here, we can indeed frame the probability to be candidate alone between two constant bounds with a simple local analysis. Thus, we can ensure that the convergence speed does not depend on the connectivity of the underlying graph. Again, on a smaller scale, we also greatly reduce the bias toward small degree nodes compared to the previous option.

3.2.3 . The proof

To prove the convergence, and the speed at which Algorithm 3.2.1 converges, we first observe that the variable x quickly (in at most one round) reaches its final value for every non-Byzantine node. We use this observation to restrict our further considerations to configurations where it has already happened. Then, we prove that whenever a rule is enabled on a node that has no Byzantine neighbor, some event happens after at most one round. After that, we use the fact that those events happen to prove that the computed independent set probabilistically grows as long as it's not maximal on the subgraph where we expect the algorithm to converge. Finally, as we can iterate this probabilistic growth, we use a concentration inequality to bound the time it takes for the algorithm to converge.

Degree-stabilization

Definition 3.2.4. We say that in a configuration γ, a node u is *degree-stabilized* if rule **Refresh** is not enabled on it. A configuration is then said to be *degree-stabilized* if every non-Byzantine node is degree-stabilized.

Observe the two following facts about the degree-stabilization of a configuration:

Lemma 3.2.5. *Any reachable configuration from a degree-stabilized configuration is degree-stabilized.*

Proof. No rule can change the x value of a degree-stabilized non-Byzantine node. □

Lemma 3.2.6. *From any configuration γ, the configuration γ' after one round is degree-stabilized.*

Proof. Let u be a non-Byzantine node.

If $x_u^\gamma = deg(u)$, no activation of rule can change that thus $x_u^{\gamma'} = deg(u)$.

If $x_u^\gamma \neq deg(u)$, then u is activable in γ and remains so until it is activated. Rule **Refresh** is then executed on u in the first round, and since no rule can change the value of x_u afterward we have $x_u^{\gamma'} = deg(u)$. □

All locally alone nodes in V_1 (*i.e.* nodes of I_γ) remain locally alone during the whole execution.

Lemma 3.2.7. *If $\gamma \to \gamma'$, $I_\gamma \subseteq I_{\gamma'}$.*

Proof. Let's consider $u \in I_\gamma$. The only rules that may be enabled on u in γ is **Refresh** since **Candidacy?** can't be executed on u because $s_u^\gamma = \top$, and **Withdrawal** can't be executed on u because $\forall v \in N(u), s_u^\gamma = \bot$. We have then $s_u^{\gamma'} = \top$.

Now let's consider $v \in N(u)$. By definition of I_γ, $s_v^\gamma = \bot$, and v has u as a neighbor that has value $\top$ in γ. Then the only rule that may be enabled on v in γ is **Refresh**, and we have $s_v^{\gamma'} = \bot$.

Thus, $u \in I_{\gamma'}$. □

When a rule is enabled, something will happen

We now focus on properties on the local evolution around a node on which a rune is enabled after one round. These properties will be later of use to prove the progression of the algorithm. We start with the **Withdrawal** rule. When the **Withdrawal** rule is enabled on a node $u \in V_1$, it is a conflict between u and its neighbor about who should be in the independent set. It is solved by either making u locally alone or setting $s_u = \bot$.

Lemma 3.2.8. *If γ is a degree-stabilized configuration and if **Withdrawal** is enabled on $u \in V_1$ then after one round either **Withdrawal** has been executed on u or in the resulting configuration γ' we have $u \in I_{\gamma'}$.*

Proof. Since γ is degree-stabilized, no **Refresh** move can be executed in any future transition. Then, since $s_u^\gamma = \top$, only **Withdrawal** can be executed on u or any of its non-Byzantine neighbors until u has been activated. Since $u \in V_1$, it is in fact true for every neighbor of u.

Since u is activable in γ, we have two cases. If u has performed a **Withdrawal** move in the next round there is nothing left to prove. If it is not the case, u must have been unactivated by fairness hypothesis, which means that each neighbor $v \in N(u)$ that had s-value $\top$ in γ has been activated. By the above, they must have performed a **Withdrawal** move that changed their s-value to $\bot$. Also, u is supposed not to have performed any rule, so it keeps s-value $\top$ in the whole round: its neighbors -that are non-Byzantine since $u \in V_1$- cannot perform any **Candidacy** move. As such, in the configuration γ' at the end of the round, every neighbor of u has s-value $\bot$, and u has s-value $\top$. Since $u \in V_1$ that means that $u \in I_{\gamma'}$. □

When **Candidacy?** rule is enabled on a node $u \in V_1$, then **Candidacy?** is executed on $v \in N[u]$ within one round.

Lemma 3.2.9. *Suppose that in γ degree-stabilized **Candidacy?** is enabled on $u \in V_1$ (i.e., $s_u^\gamma = \bot$ and $\forall v \in N(u), s_v^\gamma = \bot$).*

*After one round **Candidacy?** have either been executed on u, or on at least one neighbor of u.*

Proof. Since γ is degree-stabilized, no **Refresh** move can be executed in any future transition. Then, until u or one of its neighbors has been activated, only **Candidacy?** can be executed on them since it's the only rule that can be activated on a node with s-value $\bot$.

Since u is activable in γ, by fairness, we have two cases:

- If u is activated before the end of the round, the only rule that it could have performed for its first activation is the **Candidacy?** rule since its s-value in γ is $\bot$ and the configuration is supposed degree-stabilized.
- If not, it has been unactivated, which means that at least one neighbor $v \in N(u)$ has been activated. As $u \in V_1$, v cannot be Byzantine. The only rule that it could have performed for its first activation is the **Candidacy?** rule since its s-value in γ is $\bot$ and the configuration is supposed degree-stabilized.

□

Probability of creating a new locally alone node

If node $u \in V_1$ executes **Candidacy?** rule, then u becomes a locally alone node with a certain probability in the next configuration. So it implies that set I grows.

Fact 3.2.9.1. $\forall k \in \mathbb{N}, (1 - \frac{1}{k+1})^k > e^{-1}$

Proof. For $k = 0$ it is true ($(1 - \frac{1}{0+1})^0 = 1 > \frac{1}{e}$). Suppose now that $k \geq 1$. A basic inequality about $\ln$ is that $\forall x \in \mathbb{R}^{+*}, \ln(x) \geq 1 - x$, with equality only when $x = 1$. For $x = \frac{k}{k+1}$, it gives us $\ln\left(\frac{k}{k+1}\right) \geq 1 - \frac{k+1}{k} = -\frac{1}{k}$, with equality only when $\frac{k}{k+1} = 1$. But since $\frac{k}{k+1}$ cannot have value 1 for any value of k, we have then $\ln\left(\frac{k}{k+1}\right) > -\frac{1}{k}$.

Then, by multiplying by k on each side, we have $k \ln\left(\frac{k}{k+1}\right) > -1$, and thus, taking the exponential:

$$\left(1 - \frac{1}{k+1}\right)^k = e^{k \ln\left(\frac{k}{k+1}\right)} > e^{-1}$$

□

Lemma 3.2.10. *If γ is a degree-stabilized configuration, and in the next transition rule **Candidacy?** is executed on a node $u \in V_1$*

Proof. For any node y, we write $\varphi(y) = \frac{1}{1+max(\{deg(v)|v \in N[y]\})}$.

Since **Candidacy?** can be executed on u in γ, we know that $\forall v \in N[u], s_v^\gamma = \perp$. Since γ is degree-stabilized and no neighbor of u can be Byzantine by definition of V_1, we have $\forall v \in N[u], x_v^\gamma = deg(v)$. The probability of $s_u^{\gamma'} = \top$ knowing that the rule has been executed is then $\varphi(u)$.

Then, for a given $v \in N(u)$, node v is not Byzantine since $u \in V_1$, and the probability that $s_v^{\gamma'} = \top$ is either 0 (if **Candidacy?** has not been executed on v in the transition) or $\frac{1}{1+\max(\{x_w^\gamma | w \in N[v]\})}$. Since $deg(u) = x_u \leq \max(\{x_w^\gamma | w \in N[v]\})$, that probability is then at most $\frac{1}{1+deg(u)}$.

Thus (since those events are independent), the probability for u to be candidate in γ' without candidate neighbor is at least:

$$p = \varphi(u) \prod_{v \in N(u)} \left(1 - \frac{1}{1+deg(u)}\right) \geq \frac{1}{\Delta+1} \left(1 - \frac{1}{1+deg(u)}\right)^{deg(u)}$$

Then, as $\forall k \in \mathbb{N}, (1 - \frac{1}{k+1})^k > e^{-1}$ (see Fact 3.2.9.1), $p > \frac{1}{\Delta+1} \times \frac{1}{e}$ and the lemma holds.

□

Lemma 3.2.11. *If γ is a degree-stabilized configuration such that I_γ is not a maximal independent set of $V_2 \cup I_\gamma$, then after at most one round one of the following events happens:*

1. *Rule **Candidacy?** is executed on a node of V_1*
2. *A configuration γ' such that $I_\gamma \subsetneq I_{\gamma'}$ is reached.*

3. *A configuration γ' such that rule **Candidacy?** is enabled on a node of V_2 in γ' is reached.*

Proof. Consider γ a degree-stabilized configuration such that $I_\gamma \cup V_2$ is not a maximal independent set of V_2. As γ is supposed degree-stabilized, we will only consider the possibility of moves that are not **Refresh** moves.

- If **Candidacy?** is enabled on a node of V_2 in γ, Condition 3 holds.
- If it is not the case, then there exists $u \in V_2$ that has at least one neighbor $v \in V_1$ such that $s_u^\gamma = s_v^\gamma = \top$ (otherwise I_γ would be a maximal independent set of $V_2 \cup I_\gamma$).

In the first case, there is nothing left to prove. In the second case, **Withdrawal** is enabled on $u \in V_2$ in γ and from Lemma 3.2.8, we have two possible cases:

- If γ' is the configuration after one round, $u \in I_{\gamma'}$ and Condition 2 holds.
- u performs a **Withdrawal** move before the end of the round.

In the first case, there is nothing left to prove. Suppose now that we are in the second case and that u is activated only once before the end of the round, without loss of generality since Condition 1 would hold otherwise as its second activation would be a **Candidacy?** move and $u \in V_1$. Then:

- If within a round a configuration γ' is reached where a node $w \in N[u]$ is such that $s_w^{\gamma'} = \top$ and w is not activable we have: $w \notin I_\gamma$ (as u is $\top$-valued in γ) which gives $I_\gamma \subsetneq I_\gamma \cup \{w\} \subseteq I_{\gamma'}$ by Lemma 3.2.7, thus Condition 2 holds.
- If we suppose then that no such event happens until the end of the round in configuration γ', we are in either of those cases: (i) Every neighbor of u has value $\bot$ in γ' and $s_u^{\gamma'} = \bot$ (as we would be in the previous case if it was $\top$), thus **Candidacy?** is enabled on u in γ' and Condition 3 holds. (ii) A **Candidacy?** has been performed within the round on a neighbor of u and Condition 1 holds.

Thus, in every possible case, one of the three conditions holds. □

Every 2 rounds, the set of locally alone nodes strictly grows with some probability.

Lemma 3.2.12. *If γ is degree-stabilized, with I_γ not being a maximal independent set of $V_2 \cup I_\gamma$, then after two rounds the probability for the new configuration γ' to be such that $I_\gamma \subsetneq I_{\gamma'}$ is at least $\frac{1}{(\Delta+1)e}$.*

Proof. From Lemma 3.2.11, we have three possibilities after one round.

- If we are in Case 1 of Lemma 3.2.11, let us denote by γ'' the resulting configuration after the transition where the said **Candidacy?** move has been executed on $u \in V_1$. Then using Lemma 3.2.10, we have $u \in I_{\gamma''}$ with probability at least $\frac{1}{(\Delta+1)e}$. Since we have $u \notin I_\gamma$ (if u was in I_γ, **Candidacy?** could not have been executed on u after configuration γ) and by Lemma 3.2.7, we have then $I_\gamma \subsetneq I_{\gamma'}$ with probability at least $\frac{1}{(\Delta+1)e}$.
- If we are in Case 2 of Lemma 3.2.11, there is nothing left to prove.
- If we are in Case 3 of Lemma 3.2.11, let us denote by γ'' the first configuration where **Candidacy?** is enabled on some $u \in V_2$. Then using Lemma 3.2.9 after at most one more round **Candidacy?** will be executed on $v \in N[u]$. Let us denote by γ'' the resulting configuration after the transition where the said **Candidacy?** move has been executed on v. Since $u \in V_2$ we have $v \in V_1$ and using Lemma 3.2.10 $v \in I_{\gamma'''}$ with probability at least $\frac{1}{(\Delta+1)e}$ and $v \notin I_\gamma$ by the same argument as above. Thus, since $I_{\gamma'''} \subseteq I_{\gamma'}$, the probability that $I_\gamma \subsetneq I_{\gamma'}$ is at least $\frac{1}{(\Delta+1)e}$.

In every case, the property is true, thus the lemma holds. □

Bounding the iterations with high probability

We will use the notation $\alpha = \frac{1}{(\Delta+1)e}$ to simplify the formulas in the remainder of the chapter.

We know that after one round I gets a chance to grow with probability at least α. Then, if I is still not an independent set $V_2 \cup I$, we can repeat the same argument, and so on. We will use a concentration inequality (Azuma's inequality) to give a probabilistic bound on the number of rounds it takes to reach a configuration where I is still not an independent set $V_2 \cup I$.

Lemma 3.2.13. *From any degree-stabilized configuration γ, the algorithm is self-stabilizing for a configuration γ' where $I_{\gamma'}$ is a maximal independent set of $V_2 \cup I_{\gamma'}$.*

The time for this to happen is less than $\max\left(-\alpha^{-2}\ln p, \frac{\sqrt{2}}{\sqrt{2}-1}\frac{n}{\alpha}\right)$ rounds with probability at least $1-p$, for any value of $p \in [0,1[$.

Proof. Consider a degree-stabilized configuration γ_0, and Ω the set of all complete executions of the algorithm starting in configuration γ_0. The probability measure $\mathbb{P}$ is the one induced by the daemon strategy (as we bound the probability without knowledge of the daemon strategy, it is valid whatever the strategy of the daemon).

As we know that every round we get a minimal chance to increase I from Lemma 3.2.12 as long as it is not a maximal independent set, we want to apply a concentration inequality to bound the length of the whole execution. Here, we will use Azuma's inequality, and to do so we need to introduce martingales that depict the progression of I.

Consider for $i \in \mathbb{N}$ the random variable X_i that denotes the configuration after the i-th round has ended (X_0 is the constant random variable of value γ_0).

Consider $(\mathcal{F}_i)_{i\in\mathbb{N}}$ the natural filtration associated with X_i

To make the reading easier we introduce the function $f : \gamma \mapsto |I_\gamma|$.

$Y_i = \mathbb{1}_{f(X_i)-f(X_{i-1})>0}$ is the random variable with value 1 if the size of I increased in the i-th round, else 0. (1 means that I grows in the i-th round, 0 that it does not.)

Consider the stopping time τ (random variable describing the number of rounds the algorithm takes to stabilize on V_1) defined by:

$$\tau(\omega) = \inf \{n \in \mathbb{N} | I \text{ does not change after round } n \text{ in execution } \omega\}$$

As Y_i has values in $\{0; 1\}$, we have $\mathbb{P}(Y_i = 1|\mathcal{F}_{i-1}) = \mathbb{E}[Y_i|\mathcal{F}_{i-1}]$. Also, from Lemma 3.2.12 we get $\mathbb{P}(Y_i = 1|\mathcal{F}_{i-1}) \geq \alpha \cdot \mathbb{1}_{\tau\geq i-1}$. Thus combining the two relations we get:

$$\mathbb{E}[Y_i|\mathcal{F}_{i-1}] \geq \alpha \cdot \mathbb{1}_{\tau\geq i-1} \tag{3.1}$$

Consider $S_i = \sum_{k=1}^{i} Y_k$ the random variable representing the number of rounds where there has been an increment. Since this cannot happen more times than there are nodes, we get:

$$S_i \leq n \tag{3.2}$$

Consider $A_i = \sum_{k=1}^{i} \mathbb{E}[Y_k|\mathcal{F}_{k-1}]$ the random variable representing the sum of the expected values of the increments at each step.

When $\tau > i$, every for very value of $k \in [\![1, i]\!]$ we have $\mathbb{1}_{\tau\geq k-1} = 1$. Then using (3.1) we get:

$$\tau > i \Rightarrow A_i \geq i\alpha \tag{3.3}$$

Consider then the random variable $M_i = \sum_{k=1}^{i} Y_k - \mathbb{E}[Y_k|\mathcal{F}_{k-1}]$ (do note

that it is the same as the difference $S_i - A_i$).

$$\begin{aligned}\mathbb{E}\left[M_{i+1}|\mathcal{F}_i\right] &= \mathbb{E}\left[\sum_{k=1}^{i+1} Y_k - \mathbb{E}[Y_k|\mathcal{F}_{k-1}]\middle|\mathcal{F}_i\right] \\ &= \mathbb{E}\left[M_i + Y_{i+1} - \mathbb{E}[Y_{i+1}|\mathcal{F}_i]|\mathcal{F}_i\right] \\ &= \mathbb{E}\left[M_i|\mathcal{F}_i\right] + \mathbb{E}\left[Y_{i+1}|\mathcal{F}_i\right] - \mathbb{E}\left[\mathbb{E}[Y_{i+1}|\mathcal{F}_i]|\mathcal{F}_i\right] \\ &= M_i + \mathbb{E}\left[Y_{i+1}|\mathcal{F}_i\right] - \mathbb{E}\left[Y_{i+1}|\mathcal{F}_i\right] \\ &= M_i\end{aligned}$$

Thus $(M_i)_{i\in\mathbb{N}}$ is a martingale with respect to the filtration $(\mathcal{F}_i)_{i\in\mathbb{N}}$.

We also have:

$$|M_{i+1} - M_i| = |Y_{i+1} - \mathbb{E}[Y_{i+1}|\mathcal{F}_i]| \leq \max(Y_{i+1}, \mathbb{E}[Y_{i+1}|\mathcal{F}_i]) \leq 1$$

Thus, by Azuma's inequality:

$$\forall \beta \leq 0, \mathbb{P}(M_i \leq \beta) \leq e^{-\frac{2\beta^2}{i}} \tag{3.4}$$

Then using (3.2) and (3.3) we get $\tau > i \Rightarrow S_i - A_i \leq n - i\alpha$, *i.e.* $\tau > i \Rightarrow M_i \leq n - i\alpha$

Thus we have $\mathbb{P}(\tau > i) \leq \mathbb{P}(M_i \leq n - i\alpha)$ and for $i \geq \frac{n}{\alpha}$ we can apply (3.4) to get :

$\mathbb{P}(\tau > i) \leq e^{-\frac{2(n-i\alpha)^2}{i}}$

For $i \geq \frac{\sqrt{2}}{\sqrt{2}-1}\frac{n}{\alpha}$ (it implies that $i \geq \frac{n}{\alpha}$) we have $\frac{1}{2}(i\alpha)^2 \leq (n - i\alpha)^2$, which give for such i :

$\mathbb{P}(\tau > i) \leq e^{-i\alpha^2}$

For $i \geq -\alpha^{-2}\ln p$, we have $e^{-i\alpha^2} \leq p$

Mixing the two above inequalities, when $i \geq \max\left(-\alpha^{-2}\ln p, \frac{\sqrt{2}}{\sqrt{2}-1}\frac{n}{\alpha}\right)$, we get:

$\mathbb{P}(\tau > i) \leq p$

Which concludes the proof. □

Theorem 3.2.14. *For any $p \in [0, 1[$. From any configuration γ, the algorithm is self-stabilizing for a configuration γ' where $I_{\gamma'}$ is a maximal independent set of $V_2 \cup I_{\gamma'}$, and reach such a configuration in $1 + \max\left(-\alpha^{-2}\ln p, \frac{\sqrt{2}}{\sqrt{2}-1}\frac{n}{\alpha}\right)$ rounds or less with probability $1 - p$.*

Proof. From Lemma 3.2.6 we reach a degree-stabilized configuration after at most one round. Then from that configuration, we apply Lemma 3.2.13. □

3.3 . In an Anonymous System under the Adversary Daemon

In this section, we remove the possibility of Byzantine faults, but at the same time remove the constraint of fairness on the daemon. We could have used the previous algorithm as fairness was only needed to contain Byzantine influence regarding convergence. But the complexity in number of moves would have been something proportional to Δn^2, and as we will prove, we can do better than that.

As in Algorithm 3.2.1, this new algorithm builds a maximal independent set represented by a local variable s. We keep the idea of having nodes making candidacy and then withdrawing if the candidate's situation is not correct. We still do not have identifiers, and so we do need a probabilistic tie-break. But contrary to the byzantine case, we move the probabilities to the **Withdrawal ?** rule: a non-candidate node with no candidate neighbor will always become a candidate when activated, but a candidate node with a candidate neighbor will only withdraw with probability $\frac{1}{2}$ when activated. As the probability does not depend on the degree of the nodes, there is no need for the variable representing the degree we used in the previous section.

Algorithm 3.3.1. Any node u has a single local variable $s_u \in \{\bot, \top\}$ and may make a move according to one of the following rules:
(Candidacy) $(s_u = \bot) \wedge (\forall v \in N(u), s_v = \bot) \rightarrow s_u := \top$
(Withdrawal?) $(s_u = \top) \wedge (\exists v \in N(u), s_v = \top) \rightarrow$ if $Rand(\frac{1}{2}) = 1$ then $s_u := \bot$

Given a configuration γ, we define:

$$\beta(\gamma) = \{u \in V \mid s_u{=}\top \wedge \forall v \in N(u), s_v{=}\bot\}$$

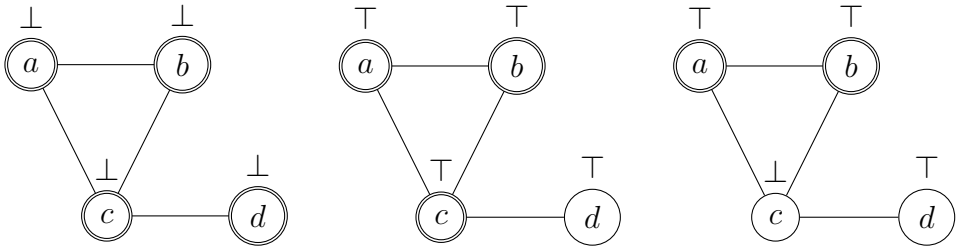
Note that $\beta(\gamma)$ is always an independent set since two distinct members cannot be neighbors (as they have both s-value $\top$).

3.3.1 . An example

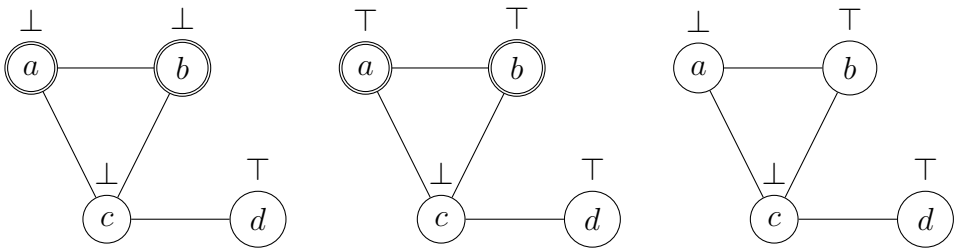
The aim of the algorithm is to build an independent set represented by $\beta(\gamma)$. The approach of the algorithm is the following: when a node is in the set independent it remains so throughout the execution.

Below is an execution of the algorithm under the adversarial distributed daemon. Figure 3.3.1a shows the initial configuration γ_0 of the execution. Node identifiers are indicated inside the circles. The symbols $\bot$ and $\top$ show the content of the local variable s. Double bordered nodes are those activated by the daemon in the next transition.

Consider the initial configuration γ_0 (Figure 3.3.1a) in which all local variables are equal to $\bot$. In this configuration the **Candidacy** rule is enabled on all nodes.



(a) Config. γ_0 (b) Config. γ_1 (c) Config. γ_2



(d) Config. γ_3 (e) Config. γ_4 (f) Config. γ_5

During the first transition $\gamma_0 \to \gamma_1$, the demon activates all the nodes. The resulting configuration γ_1 where every node has s-value $\top$ is drawn in Figure 3.3.1b.

In γ_1, **Whithdrawal** is enabled on all nodes. In the second transition, the daemon activates a,b and c. Those three nodes change independently their s-value with probability $\frac{1}{2}$, which results in confuguration γ_2 where a and b did not change their s-value but c did (see Figure 3.3.1c). Observe that node d is an element of $\beta(\gamma_2)$. It will remain in β until the end of the execution as its only neighbor whose s-value is $\bot$ cannot execute any rule while having a neighbor with s-value $\top$.

In γ_2, only a and b are enabled for rule **Withdrawal?**. In the next transition, both are activated by the daemon. Both change their s-value with probability $\frac{1}{2}$, resulting in configuration γ_3 where both get their s-value changed to $\bot$ (see Figure 3.3.1d).

In γ_3, only a and b are enabled for rule **Candidacy**. In the next transition, both are activated by the daemon, and both change their s-value to $\top$, resulting in configuration γ_3 (see Figure 3.3.1e).

Note that $\gamma_4 = \gamma_2$. In next transition a and b are again activated by the daemon and execute **Withdrawal?**. This time, the random change in s-value results in configuration γ_5, where a get its s-value changed to $\bot$ but not b (see Figure 3.3.1f).

The γ_5 configuration is stable since no node is enabled. $\beta(\gamma_5) = \{b, d\}$ is a maximal independent set of the underlying graph.

3.3.2 . The proof

The following lemma guarantees the correction of the algorithm.

Lemma 3.3.2. *A configuration γ is stable if and only if $\beta(\gamma)$ is a maximal independent set of G.*

Proof. Suppose γ is not stable.

- If **Withdrawal?** is enabled on a node u, it means that $u \notin \beta(\gamma)$ as u has a neighbor with s-value $\top$. But as u has s-value $\top$ it also means that none of its neighbors is in $\beta(\gamma)$, thus $\beta(\gamma) \cup \{u\}$ is an independent set greater than $\beta(\gamma)$.
- Else, **Candidacy** is enabled on a node u. It means that $u \notin \beta(\gamma)$ as $s_u = \bot$. It also means that none of its neighbors is in $\beta(\gamma)$ since they have all s-value $\bot$, thus $\beta(\gamma) \cup \{u\}$ is an independent set greater than $\beta(\gamma)$.

Thus, by contraposition, if $\beta(\gamma)$ is a maximal independent set of G, then γ is stable.

Suppose now that γ is stable. As **Withdrawal?** cannot be enabled on any node, any node with s-value $\top$ is in $\beta(\gamma)$. But then as **Withdrawal?** cannot be enabled on any node, every node has a node with s-value $\top$, and thus a node of $\beta(\gamma)$ in its closed neighborhood. $\beta(\gamma)$ is then a maximal independent set of G. □

Now that we have correction, we want to prove the convergence of our algorithm. We begin by proving that β is non-decreasing, thus is remains to prove that it does grow at some point. Then, we prove that any execution is equivalent to another execution that follows some restrictions, which allow us to work under the assumption that these restrictions are verified in our further considerations. After that, we focus on connected components with β-value $\top$. We prove that every time such a connected component "disappear", it makes β grow with probability $\frac{2}{3}$, and then observe how they appear, shrink and disappear in any execution. We use those facts to prove that any execution ends with probability 1. As it implies that every of those vanishing happens in finite time with probability 1. It finally allows us to use the repetition of those vanishings with a concentration inequality to bound the length of the execution.

β is non-decreasing

Looking at the guards of the rules of Algorithm 3.3.1, we observe the following fact:

Fact 3.3.2.1. *In a configuration γ, when **Withdrawal?** is enabled on a node, **Candidacy** is not enabled on any of its neighbors.*

Lemma 3.3.3. *If $\gamma \to \gamma'$, then $\beta(\gamma) \subseteq \beta(\gamma')$.*

Proof. For every $u \in \beta(\gamma)$, we have by definition:

1. $s_u^\gamma = \top$,
2. for every neighbor u of v, $s_u^\gamma = \bot$.

Point 1 implies that **Candidacy** is not enabled on any vertex of $N[u]$, and Point 2 implies that **Withdrawal?** is not enabled on any vertex of $N[u]$.

For every vertex $v \in N[u]$, since no rule is enabled on v, we have $s_v^\gamma = s_v^{\gamma'}$. Thus, $u \in \beta(\gamma')$. □

Simplifying the execution

Now that we know that β cannot shrink, it remains to prove that it does grow, and within a reasonable amount of time. We are introducing the following concepts to this end:

Definition 3.3.4. A set of nodes $\mathcal{X} \subseteq V$ is said to be a *candidate set* of a configuration γ if $\forall u \in \mathcal{X}, (s_u^\gamma = \top) \wedge \forall v \in N(u), s_v^\gamma = \top \Rightarrow v \in \mathcal{X}$. We will say that it is a *connected* candidate set when $\mathcal{X}$ is a connected component of G.

First, we transform all executions into executions having suitable properties, using the two following lemmas for the simplicity of the proof.

Lemma 3.3.5. *Consider γ_1 a configuration and t a valid set of moves in γ_1. Let us write $t_w \subset t$ (resp. $t_c \subset t$) the set of **Withdrawal?** moves (resp **Candidacy** moves) of t.*

Then making two successive transitions using the set of moves t_w and then t_c is equivalent to make a transition with the set of moves t (in the sense that the distribution of probability on the values of nodes variables is the same).

Proof. Consider γ_2 the random variable describing the state of the configuration attained from γ_1 by executing the set of moves t.

Let W (resp. C) be the set of the nodes that appear in a **Withdrawal?** (resp. **Candidacy**) move of t.

There is only one possible γ' such that $\gamma_1 \xrightarrow{t_c} \gamma'$ since the rule **Candidacy** is deterministic, consider this configuration γ'.

Due to Fact 3.3.2.1, no node in W has a node of C as neighbor. Thus, nodes in W and their neighbors did not change their local variable in the transition $\gamma_1 \xrightarrow{t_c} \gamma'$. Then, **Withdrawal?** is enabled on every node of W in γ' and t_w is a valid set of moves in γ'.

Consider then the random variable γ'_2 such that $\gamma' \xrightarrow{t_w} \gamma'_2$.

- For $u \in C$, the transitions are deterministic and we have $s_u^{\gamma'_2} = \top = s_u^{\gamma_2}$.
- For $u \in W$, note that as long as **Withdrawal?** is executed, its results always follow the same probability law. Thus, as nodes that execute **Withdrawal ?** are the same in $\gamma' \xrightarrow{t_w} \gamma'_2$ and $\gamma_1 \xrightarrow{t} \gamma_2$, the probability distribution of s_u is the same in γ_2 and γ'_2.
- For nodes that are neither in C, nor in W, they did not execute any rule in the considered transitions and thus they have the same s-value in every configuration we considered.

Thus, γ_2 has the same probability distribution as γ'_2, and hence the result. □

Lemma 3.3.6. *Let γ_1 be a configuration, t a valid set of moves in γ_1 containing only **Withdrawal?** moves, and A a candidate set of γ_1. Let also $t_A \subset t$ the set of moves on node of A in t.*

Then making two successive transitions using the set of moves t_A and then $t \setminus t_A$ is equivalent to make a transition with the set of moves t (in the sense that the distribution of probability on the values of nodes variables is the same).

i.e. we may replace $\gamma_1 \xrightarrow{t} \gamma_2$ with $\gamma_1 \xrightarrow{t_A} \gamma' \xrightarrow{t \setminus t_A} \gamma_2$.

Proof. The proof follows the same pattern as the one of Lemma 3.3.6. It relies on the fact that a rule -here **Withdrawal ?**- applied on a node u induces the same distribution of probability on the s-value of u independently of context.

The only thing left to prove is that $t \setminus t_A$ is a valid set of moves in any γ' such that $\gamma_1 \xrightarrow{t_A} \gamma'$.

Let's denote by B the set of the nodes that appear in moves of $t \setminus t_A$.

Consider $u \in B$. As **Withdrawal?** is enabled on u in γ_1 we know that $s_u^{\gamma_1} = \top$. Then, by definition of a candidate set, u cannot be a neighbor of a node in A, and as a result no node of $N[u]$ is activated in the transition $\gamma_1 \xrightarrow{t_A} \gamma'$. The closed neighborhood of u having the same s-values in γ_1 and γ', **Withdrawal?** must be enabled on u in γ'.

It is then the case for every node of B, and thus $t \setminus t_A$ is a valid set of moves in γ'. □

Using those lemmas, we will now only consider executions satisfying the two following properties:

- Each transition is composed of only one move type,
- For each transition $\gamma_{i-1} \xrightarrow{t_i} \gamma_i$ containing **Withdrawal?** moves, only nodes of the same connected candidate set in γ_{i-1} execute a move.

Using Lemmas 3.3.5 and 3.3.6, all executions can be transformed into executions satisfying the two last property without increasing the number of moves.

From now, all the executions we consider satisfy the two previous properties.

Candidate sets

Let $\mathcal{E}$ be an execution $\gamma_0 \xrightarrow{t_1} \gamma_1 \cdots \gamma_{i-1} \xrightarrow{t_i} \gamma_i \cdots$.

Definition 3.3.7. We say that a candidate set $\mathcal{X}$ of γ is *alive* in γ if **Withdrawal?** is enabled on at least a node of $\mathcal{X}$. Note that for connected candidate sets it is equivalent to having cardinality at least 2.

Definition 3.3.8. In a transition $\gamma \xrightarrow{t} \gamma'$, we say that an alive candidate set $\mathcal{X}$ of γ *vanishes* if $\forall u \in \mathcal{X}$, $s_u^{\gamma'} = \bot$ or $u \in \beta_{\gamma'}$.

Lemma 3.3.9. *Suppose that $\mathcal{X}$ is an alive connected candidate set of γ, and that t is a valid set of moves in γ that involve nodes of $\mathcal{X}$ (recall that we transformed our execution to make withdrawal transitions only act on a given connected candidate set).*

Then, if γ' is the random variable describing the state of the system after performing the set of moves t from γ, the probability that $\beta(\gamma') \setminus \beta(\gamma) \neq \emptyset$ knowing that $\mathcal{X}$ vanishes is at least $\frac{2}{3}$.

Proof. If there is one node u in $\mathcal{X}$ that is not activated in the transition, we know that $s_u^{\gamma'} = s_u^{\gamma} = \top$. As $\mathcal{X}$ is supposed to vanish in the transition, it implies that $u \in \beta(\gamma')$.

Else, every node of $\mathcal{X}$ has performed **Withdrawal ?** in the transition.

Consider the distribution of the s-values after performing the set of moves t in configuration γ (without the condition on the candidate set vanishing). Let's write $k = |\mathcal{X}|$. As every node of $\mathcal{X}$ flips a coin independently, every outcome has the same probability $\left(\frac{1}{2}\right)^k$. In particular,

- The event where all nodes of $\mathcal{X}$ change their s-value to $\bot$ in the transition.

- Each of the k events where all the nodes of $\mathcal{X}$ change their s-value to $\bot$ in the transition except exactly one.

Those events are compatible with the constraint of the candidate set vanishing. Every other event is either incompatible with the constraint, or a situation where at least two nodes of $\mathcal{X}$ are in $\beta(\gamma')$, as at least two nodes have s-value $\top$ in γ', and $\mathcal{X}$ is supposed to vanish in the transition.

Thus, if we denote by λ the sum of the probability of the events that are compatible with the constraint of the candidate set vanishing, we can say that $\lambda \geq (k+1)\left(\frac{1}{2}\right)^k$, and thus the probability to have $\beta(\gamma_{j^*}) \setminus \beta(\gamma_{j^*-1}) \neq \emptyset$ with the condition is $1 - \frac{\left(\frac{1}{2}\right)^k}{\lambda} \geq 1 - \frac{\left(\frac{1}{2}\right)^k}{(k+1)\left(\frac{1}{2}\right)^k} = 1 - \frac{1}{k+1} = \frac{k}{k+1}$.

Moreover, $\mathcal{X}$ is not empty by hypothesis. Consider $u \in \mathcal{X}$, it has at least a neighbor v such that $s_v^\gamma = \top$ as otherwise we would have $u \in \beta(\gamma)$. But then we have $v \in \mathcal{X}$, as $\mathcal{X}$ is a candidate set. Thus $k \geq 2$.

Thus the wanted probability is at least $\frac{2}{3}$. □

Now that we know that every time that an alive connected candidate set vanishes β progresses with good probability, it remains to guarantee that it ever happens. To do so, we will need the following lemmas about life and death of candidate sets.

Lemma 3.3.10. *If $\gamma \xrightarrow{t} \gamma'$ is a transition with t containing only **Candidacy** moves, either $\beta(\gamma) \subsetneq \beta(\gamma')$ or $\exists \mathcal{X}$ an alive connected candidate set of γ' such that $\forall u \in \mathcal{X}, s_u^\gamma = \bot$.*

Proof. As a transition must contain at least one move, there is at least one node u appearing in t.

If no neighbor of u appear in t, we know that no neighbor of u changes its s-value in the transition. But since **Candidacy** is enabled on u in γ we know that their s-value in γ is $\bot$. Thus, $s_u^{\gamma'} = \top$ (since u executed **Candidacy** in the transition), and every neighbor of u has s-value $\bot$ in γ', which is the definition for $u \in \beta(\gamma')$. And since **Candidacy** is enabled on u in γ, $s_u^\gamma = \bot$, thus $u \notin \beta(\gamma')$. Thus $\beta(\gamma) \subsetneq \beta(\gamma')$ (recall that, from Lemma 3.3.3, β cannot loose nodes in a transition).

Suppose now that at least a neighbor of u appears in t. Then consider the set $\mathcal{X}$ of the nodes that are accessible from u in the subgraph induced by the nodes appearing in t. Every node $v \in \mathcal{X}$ executes **Candidacy** in the transition, thus $s_v^\gamma = \bot$ and $s_v^{\gamma'} = \top$. Moreover, for every neighbor of a node of $\mathcal{X}$ not in $\mathcal{X}$, the condition of **Candidacy** implies that they have s-value $\bot$ in γ, and thus in γ' too since they did not perform a rule in the transition. Thus $\mathcal{X}$ is a

candidate set of γ', connected by construction, and of cardinality at least 2 by hypothesis thus alive. □

Lemma 3.3.11. *If $\gamma \xrightarrow{t} \gamma'$ is a transition and $\mathcal{X}$ is an alive connected candidate set of γ, one of those is true:*

- *$\mathcal{X}$ vanishes in the transition,*
- *$\exists \mathcal{X}'$ an alive connected candidate set of γ' such that $\mathcal{X}' \subseteq \mathcal{X}$.*

Proof. If t is a set of **Candidacy** moves, or a set of **Withdrawal?** moves on a connected candidate set different from $\mathcal{X}$, then the transition does not affect neither nodes of $\mathcal{X}$ nor their neighbors. Thus $\mathcal{X} \subseteq \mathcal{X}$ is still an alive connected candidate set of γ'.

Else, t is a set of **Withdrawal?** moves on nodes of $\mathcal{X}$. Suppose $\mathcal{X}$ does not vanish in the transition. By definition of vanishing, it means that it exists a node $u \in \mathcal{X}$ such that $s_u^{\gamma'} = \top$ and $u \notin \beta(\gamma')$. It implies that there exists $v \in N(u)$ such that $s_v^{\gamma'} = \top$. But then, as t contains only **Withdrawal?** moves, $s_v^{\gamma} = \top$. By definition of a candidate set, we must have $v \in \mathcal{X}$. Now consider $\mathcal{X}'$ the set of nodes in the connected component of u in the subgraph of G induced by the nodes with s-value $\top$ in γ'. As only **Withdrawal?** moves have been executed in the transition, this implies that $\mathcal{X}'$ is a subset of the set of nodes in the connected component of u in the subgraph of G induced by the nodes with s-value $\top$ in γ, which is exactly $\mathcal{X}$. Moreover, if $w \notin \mathcal{X}'$ is a neighbor of a node in $\mathcal{X}'$, by definition of $\mathcal{X}'$ it is such that $s_w^{\gamma'} = \bot$ (it would be part of $\mathcal{X}'$ otherwise). Hence, cX' is a connected candidate set of γ', alive since of cardinality at least 2. □

The execution ends

There cannot be more alive candidates sets than there is space in the underlying graph.

Lemma 3.3.12. *A configuration γ contains at most $\frac{n}{2} - |\beta(\gamma)|$ distinct alive connected candidate sets.*

Proof. An alive candidate set must have at least two nodes. Two candidate sets that share a node must be the same candidate set as the definition of a candidate set prevents from having s-value $\top$ in the neighborhood of a candidate set. □

This property allows us to guarantee that some candidate sets do vanish in the execution, and subsequently that the execution ends.

Lemma 3.3.13. *Any execution ends with probability* 1.

Proof. Every transition is either a transition with only **Withdrawal ?** moves, or only **Candidacy** moves.

Note that from Lemma 3.3.11 the number of alive connected components cannot shrink without one vanishing, and shrinks by exactly one when it happens.

Every transition with **Withdrawal ?** moves that do not make an alive connected candidate set vanish makes an alive connected candidate set shrink with probability at least $\frac{1}{2}$. Moreover when a transition makes one of those disappear the size of β increases with probability at least $\frac{2}{3}$.

Every transition with **Candidacy** moves either makes a new alive candidate set appear, or increases the size of β. But as there can not be more than $\frac{n}{2} - |\beta|$ candidate sets alive, there cannot be a transition that makes a new alive candidate set appear while the number of alive candidate sets is maximal, and thus one must vanish before it becomes possible again (which will increase the size of β with probability at least $\frac{2}{3}$).

By induction on the number of alive connected candidate sets and their size, as long as **Candidacy** or **Withdrawal ?** is enabled, β will grow within a finite time. As its size cannot be greater than the number of nodes, the execution must end. □

Convergence speed

Now, as we know that the execution ends with probability 1, we may suppose that every alive candidate set vanishes in finite time. We can then use Lemma 3.3.9 with a concentration inequality (as in Lemma 3.2.13) to bound the number of vanishings in an execution.

Lemma 3.3.14. *For* $p \in]0, 1]$*, any execution has at most* $\max\left(-\frac{9}{4}\ln p, \frac{\sqrt{2}}{\sqrt{2}-1}\frac{3n}{2}\right)$ *transitions in which a connected candidate set vanishes with probability at least* $1 - p$.

Proof. It is the same as the proof of Lemma 3.2.13, but the event we track is the vanishing of connected candidate set (which happens in finite time with probability 1 from Lemma 3.3.13), the lower bound on the probability of the interesting event to happen when such a vanishing happens is $\frac{2}{3}$ from Lemma 3.3.9. □

It only remains to count the number of moves associated with such an execution to prove our theorem.

Theorem 3.3.15. *Consider $p \in]0,1]$. From any configuration, the algorithm is self-stabilizing for a configuration γ in which $\beta(\gamma)$ is a maximal independent set, with at most $\mathcal{O}(n^2)$ moves with probability at least $1-p$.*

Proof. Consider $p' \in]0,1]$, $p' = \frac{p}{2}$, and $\lambda = \max\left(-\frac{9}{4}\ln p', \frac{\sqrt{2}}{\sqrt{2}-1}\frac{3n}{2}\right)$. We denote by x the number of vanishing of connected candidate sets in the execution. Using Lemma 3.3.14, we know that $x > \lambda$ with probability at most p'.

First, we count the number of **Candidacy** moves. For each transition in which a connected candidate set vanishes, those nodes have had their s-value set to $\top$ by the last transition containing **Candidacy** moves on the said candidate set when it exists. When it does not, they had s-value $\top$ in the initial state. This makes for at most n **Candidacy** move for every connected candidate set that vanishes, thus xn in total.

To count for **Withdrawal?** moves, we will first count the number of successful ones,*i.e.* those that actually succeed in changing the s-value of the node that executes it. For every connected candidate set that vanishes, as they may only shrink from the time they appear due to Lemma 3.3.11, it takes at most n successful **Withdrawal?** moves to make it disappear. Note that any **Withdrawal?** move acts on an alive connected candidate set, thus every **Withdrawal?** participates to the shrinking of an alive connected candidate set. Then, in total, we need at most xn successful successful **Withdrawal?** moves to have x vanishings of connected candidate sets.

Each **Withdrawal?** move has probability $\frac{1}{2}$ to be successful, a probability which does not depend on anything but the fact that rule **Withdrawal?** is applied, and are thus the outcome of different **Withdrawal?** moves are independent. The number of successful **Withdrawal?** is then connected to the total number of **Withdrawal?** by a binomial law of parameter λ and $\frac{1}{2}$, where λ is the number of **Withdrawal?** moves. Using the standard bound obtained by Hoeffding inequality for the binomial law, the probability to have $\lambda n - 1$ or less successful **Withdrawal?** after $2(\lambda n + \sqrt{-\lambda n \log p'} - 1)$ **Withdrawal?** moves is at most p. This means that the probability to not have finished the execution after $2(xn + \sqrt{-xn \log p'} - 1)$ **Withdrawal?** moves is at most p'.

Then, using the union bound, the probability for x to be such that $x > \lambda$ or the number of **Withdrawal?** moves in the execution to be greater than $2(\lambda n + \sqrt{-\lambda n \log p'} - 1)$ is at most $2p' = p$.

Thus, the probability to converge in less than $2(\lambda n + \sqrt{-\lambda n \log p'} - 1) + \lambda n = O(n^2)$ moves in total is at least $1-p$, which concludes the proof. □

3.4 . Conclusion

In this chapter, we presented two algorithms to find a MIS. The first one acting under the (distributed) fair daemon with Byzantine nodes finds a MIS of an induced subgraph containing at least every node without Byzantine neighbor in $O(\Delta n)$ rounds. The second one acting under the adversarial daemon finds a MIS of the whole graph in $O(n^2)$ moves.

A notion that generalizes the notion of maximal independent set is the notion of $(k, k-1)$-ruling set. We study the problem of finding such a ruling set in a distributed system in the next chapter, under the Gouda daemon.

4 - Ruling Set

The greedy approach is often considered to solve a problem: is it possible to build up a solution step by step by completing a partial solution? For example, in graph theory, one can consider the MIS problem where you want to select a set of nodes such that no two selected nodes are adjacent and any unselected node is a neighbor of a selected one. To produce a MIS, a simple algorithm is to select a node, reject all its neighbors, and then repeat this operation until there is no node left. Another classical greedy algorithm is the one that produces a $(\Delta + 1)$-coloring of a graph, where Δ is the maximum degree in the graph. Each time a node is considered, as it has at most Δ different colors in its neighborhood, we can always choose a different color for it to extend the current partial solution. Observe that most of the graphs admit a Δ-coloring, which cannot be found with this kind of greedy heuristic. We can also notice that the size of a MIS can be arbitrarily smaller than the size of a *maximum* independent set. *Greedy problems* are problems that can be solved using greedy algorithms.

Mendable problems have recently been introduced in [5] as a generalization of greedy problems. In these problems, a solution can be found sequentially, by producing the output of each node one after another as it is the case in greedy problems. However, here, for each chosen node, it is possible to change the output of its neighborhood, but only up to some distance. The set of mendable problems is larger than the set of greedy ones. For instance, the 4-coloring of the grid is a mendable problem, but it cannot be solved greedily, as its maximal degree Δ is equal to 4.

A more generalized way to consider MIS is *ruling sets*. Given a graph $G = (V, E)$, a (a, b)-*ruling set* is a subset $S \subset V$ such that the distance between any two nodes in S is at least a, and any node in V is at distance at most b from some node in S. In what follows, the concept of *ball* will play an important role. Formally, the *ball* of radius i and center s, $\mathcal{B}(s, i)$, is the set of nodes that are at distance at most i from s. Observe that a ball of radius $a - 1$ centered in a node of the ruling set S contains only one node in S.

In particular, a $(2, 1)$-ruling set is an MIS of G. A $(k, k - 1)$-ruling set S is a maximal independent set at distance k: all the elements of S are at distance at least k from each other, every other node is at distance at most $k - 1$ from S, and thus cannot be added. Note that it is an MIS of G^{k-1} (the graph with the same vertices as G, and with edges between two vertices if there are at distance $k - 1$ or less from each other in G), and this problem can be greedily solved. A $(k, k - 1)$-ruling set can also be seen as a maximal distance-k independent set where a distance-k independent set is a subset of nodes at distance at least k from

each other.

A *distance-K coloring* of a graph $G = (V, E)$ is a mapping $\mathcal{C} : V \to \mathbb{N}$ such that $\forall u \neq v \in V^2$, $dist(u, v) \leq K \Rightarrow \mathcal{C}(u) \neq \mathcal{C}(v)$. A way to produce a distance-K coloring is to partition V into sets of nodes at distance at least $k > K$ from each other, *i.e.* distance-k independent sets, each one representing a color. One can construct such a partition sequentially by constructing a partition into $\ell \geq \Delta^k$ distance-k independent sets $\{S^{(i)}\}_{i \leq \ell}$, where $S^{(i)}$ is a distance-k independent set of G maximal under the constraint that every node of the independent must be in $V \setminus \bigcup_{j<i} S^{(j)}$. These distance-$k$ independent sets can be computed in a very similar way as $(k, k-1)$-ruling sets. Distance-K coloring is a way to simulate Local algorithms: the colors can be used as constant size identifiers, as long as the simulated algorithm does run in less than K rounds on this range of identifiers.

Our Contribution and structure of the chapter

In this chapter, we provide the self-stabilizing algorithm that computes a $(k, k-1)$-ruling set in an anonymous network under the Gouda daemon (Section 4.2). The algorithm detects when a leader can be added or two leaders are too close. To that end, each node computes its distance to the leaders. If a node and its neighbors are at distance at least $k-1$ from the leaders, that node can try to add itself to the ruling set. If two leaders are too close, thanks to a clock system that consists of a mosaic of local synchronizers beta from [2], a node in the middle of the path will eventually detect the problem and initiate the removal of the leaders from the set. Thanks to the Gouda daemon, we ensure that not too many nodes will try to add themselves simultaneously and that the clock system will eventually detect collisions. On the other hand, we prove that a stable configuration can always be reached, and the Gouda daemon ensures that it eventually happens (Section 4.3).

In Section 4.4, by combining this algorithm Δ^k times, we partition the graph into distance-k independent sets, which correspond to a distance-K coloring for any $K < k$. This coloring allows us to consider nodes of each set sequentially to compute a solution to some greedy problem. In Section 4.5, we present a solution allowing us to solve any T-mendable problem, where T is a constant corresponding to the radius up to which we are allowed to change the output of a node. To that end, we use the fact that a Local algorithm runs in r rounds, for some constant r, when a distance-$(2T+1)$ coloring is given. To do that, we compute a distance-$(2T+1)$ and a distance-$(2r+1)$ coloring. That way, each node will be able to have access to its neighborhood at the right distance and compute the output the Local algorithm would have given in that situation.

4.1 . State of the art

The notion of *checking locally* and its relationship with the notion of *solving locally* have been introduced by Naor and Stockmeyer in [54]. This work, along with Cole and Vishkin's algorithm that efficiently computes a 3-coloring of a ring [17], leads to the notion of *Locally Checkable Labelling problems* (Lcl) and the Local model. Locally checkable problems are problems such that when the output is locally correct for each node, the global output is guaranteed to be correct too. Coloring and MIS belong to that field. Ruling Set problems are also Lcl problems: to check locally that the solution is right, the distance to the set must be given in the output. The Local model (see [60] for a survey) is a synchronous model that requires unique identifiers but does not impose any restriction on communication bandwidth or computation complexity. The goal is to find sublinear time algorithms. An adaptation of the Local model, the Slocal model [35] considers algorithms that are executed on nodes one after another, only one time each, but are allowed to see the state of every node up to some distance when they do. In particular, locally greedy problems are solved with constant distance of sight in this model.

Bitton *et al.* [11] designed a self-stabilizing transformer for Local problems. Their probabilistic transformer converts a given fault-free synchronous algorithm for Lcl problems into a self-stabilizing synchronous algorithm for the same problem in anonymous networks. The overheads of this transformation in terms of message complexity and average time complexity are upper bounded: the produced algorithms stabilize in time proportional to $\log(\alpha + \Delta)$ in expectation, where α is the number of faulty nodes.

Awerbuch *et al.* [3] introduced the ruling set as a tool for decomposing the graph into small-diameter connected components. As for the seminal work, the Ruling Set problems have been used as a sub-routine function in order to solve some other distributed problems (network decompositions [3, 9], colorings [55], shortest paths [41]).

The MIS problem has been extensively studied in the Local model, [33, 57, 14] for instance and in the Congest model [56] (synchronous model where messages are $O(\log n)$ bits long). In the Local model, Barenboim *et al.* [8] focus on systems with unique identifiers and gave a self-stabilizing algorithm producing an MIS within $O(\Delta + \log^* n)$ rounds. Balliu *et al.* [7] prove that the previous algorithm [8] is optimal for a wide range of parameters in the Local model. In the Congest model, Ghaffari *et al.* [34] prove that there exists a randomized distributed algorithm that computes a maximal independent set in $O(\log \Delta \cdot \log \log n + \log^6 \log n)$ rounds with high probability. Considering the problem (α, β)-ruling set in a more general way, Balliu *et al.* [4] give some lower bound for computing a $(2, \beta)$-ruling set in the Local model: any deterministic algorithm requires $\Omega\left(\min\left\{\frac{\log \Delta}{\beta \log \log \Delta}, \log n\right\}\right)$

rounds.

Up to our knowledge, no self-stabilizing algorithm has been designed for computing $(k, k-1)$-ruling sets where $k > 2$ under the Gouda daemon. Self-stabilizing algorithms for maximal independent set have been designed in various models (anonymous network [59, 64, 63] or not [36, 42, 62]). Shukla *et al.* [59] present the first self-stabilization algorithm designed for finding an MIS for anonymous networks. Turau [62] gives the best known result with $O(n)$ moves under the distributed daemon. Recently, some works improved the results in the synchronous model. For non-anonymous networks, Hedetniemi [40] designed a self-stabilization algorithm that stabilizes in $O(n)$ synchronous rounds. Moreover, for anonymous networks, Turau [63] designs some randomized self-stabilizing algorithms for maximal independent set that stabilizes in $O(\log n)$ rounds with high probability. See the survey [39] for more details on MIS self-stabilizing algorithms.

4.2 . Self-Stabilizing Algorithm for Computing a $(k, k-1)$-Ruling Set

4.2.1 . General Overview

As we want to compute a $(k, k-1)$-ruling set, a node needs to detect when it is currently "too far" from the nodes pretending to be in the ruling set. When $k = 2$, a $(2, 1)$-ruling set is a MIS, and some self-stabilization algorithms are designed for finding a MIS [59, 64, 63]. For the remaining of the document, we assume $k > 2$.

To this aim, the local variable d represents the distance at which the node thinks it is from the ruling set. In particular, a d-value of 0 indicates that a node is (or thinks it is) in the ruling set, and we denote by $S(\gamma)$ the set of those nodes in a given configuration γ. Any other value of d_u is meant to represent the distance to $S(\gamma)$ (actually, the minimum between $k-1$ and the said distance). This will not be true in every configuration as the information needs to spread, which is enforced by the rule **Update distance**, which has the highest priority. When a node u has its local variable d_u equal to $k-1$ and is surrounded by nodes of d-value $k-1$, it "knows" that it is far enough from $S(\gamma)$ to be added to it. Node u can then execute **Become Leader** to do so. Update of d-values will then spread from the new member of $S(\gamma)$ through the execution of **Update distance**.

The way to insert new nodes into $S(\gamma)$ cannot avoid the fact that two new members of $S(\gamma)$ may be too close. A way to detect those problems is needed to guarantee that we will not let those nodes in $S(\gamma)$.

If they are close enough (distance 2 or less), it can be directly detected by a node (either a common neighbor if they are at distance 2, or one of them if they are at distance 1). The rule **Two Heads** is here to detect this.

When problematic nodes are too far away, no node can detect locally this

————– Attributes of the nodes

$\mathbf{d}_u \in [\![0, k-1]\!]$
$\mathbf{err}_u \in \{0, 1\}$
For every $i \in [\![1, \lfloor \frac{k}{2} \rfloor - 1]\!]$: $\mathbf{c}_{i,u} \in \mathbb{Z}/4\mathbb{Z}$ and $\mathbf{b}_{i,u} \in \{\uparrow, \downarrow\}$

————– Predicates

well_defined$(u) \equiv err_u = 0 \wedge \forall v \in N(u), |d_u - d_v| \leq 1 \wedge (d_u > 0 \Rightarrow (\exists v \in N(u), d_v = d_u - 1))$
leader_down$(u) \equiv d_u = 0 \Rightarrow \forall i \in [\![1, \lfloor \frac{k}{2} \rfloor - 1]\!], b_{i,u} = \downarrow$
branch_coherence_up$(u, i) \equiv$
$\forall v \in N(u), d_v = d_u - 1 \Rightarrow (b_{i,u}, b_{i,v}, c_{i,v}) \in \{(\uparrow, \uparrow, c_{i,u}), (\uparrow, \downarrow, c_{i,u}), (\uparrow, \downarrow, c_{i,u} + 1), (\downarrow, \downarrow, c_{i,u})\}$
branch_coherence_down$(u, i) \equiv$
$\forall v \in N(u), d_v = d_u + 1 \Rightarrow (b_{i,u}, b_{i,v}, c_{i,v}) \in \{(\uparrow, \uparrow, c_{i,u}), (\downarrow, \uparrow, c_{i,u}), (\downarrow, \uparrow, c_{i,u} - 1), (\downarrow, \downarrow, c_{i,u})\}$
branch_coherence$(u) \equiv d_u \geq \lfloor \frac{k}{2} \rfloor \vee \big(branch_coherence_up(u, d_u) \ \wedge$
$\forall i \in [\![d_u + 1, \lfloor \frac{k}{2} \rfloor - 1]\!], branch\_coherence\_up(u, i) \wedge branch\_coherence\_down(u, i)\big)$

————– Rules

Incr Leader:: (priority 2)
if well_defined$(u) \wedge (d_u = 0) \wedge (\exists i \in [\![1, \lfloor \frac{k}{2} \rfloor - 1]\!], \forall v \in N(u), d_v = 1 \wedge c_{i,u} - c_{i,v} = 0)$
then For all such i, $c_{i,u} := c_{i,u} + 1$

Sync 1 down:: (priority 2)
if well_defined$(u) \wedge \exists! v \in N(u), \exists i \in [\![1, \lfloor \frac{k}{2} \rfloor - 1]\!], d_u = 1 \wedge d_v = 0 \wedge c_{i,u} = c_{i,v} - 1 \wedge b_{i,u} = \uparrow)$
then For all such i, $c_{i,u} := c_{i,v}$; $b_{i,u} := \downarrow$

Sync 2+ down:: (priority 2)
if well_defined$(u) \wedge 1 < d_u < \lfloor \frac{k}{2} \rfloor$
$\wedge(\exists i \in [\![d_u, \lfloor \frac{k}{2} \rfloor - 1]\!], b_{i,u} = \uparrow \wedge \forall v \in N(u), d_v = d_u - 1 \Rightarrow (c_{i,u} = c_{i,v} - 1 \wedge b_{i,v} = \downarrow))$
then For all such i, $c_{i,u} := c_{i,v}$; $b_{i,u} := \downarrow$

Sync 1+ up:: (priority 2)
if well_defined$(u) \wedge 0 < d_u < \lfloor \frac{k}{2} \rfloor$
$\wedge(\exists i \in [\![d_u + 1, \lfloor \frac{k}{2} \rfloor - 1]\!], b_{i,u} = \downarrow \wedge \forall v \in N(u), d_v = d_u + 1 \Rightarrow (c_{i,u} = c_{i,v} \wedge b_{i,v} = \uparrow))$
then For all such i, $b_{i,u} := \uparrow$

Sync end-of-chain:: (priority 2)
if well_defined$(u) \wedge 0 < d_u < \lfloor \frac{k}{2} \rfloor \wedge \forall v \in N(u), d_v = d_u - 1 \Rightarrow (c_{d_u,u} = c_{d_u,v} - 1 \wedge b_{i,v} = \downarrow))$
then $b_{d_u,u} := \uparrow$; $c_{d_u,u} := c_{i,v}$

Update distance :: (priority 0)
if $(d_u \neq 0) \wedge d_u \neq \min(\min\{d_v | v \in N(u)\} + 1, k - 1)$
then $d_u := \min(\min\{d_v | v \in N(u)\} + 1, k - 1)$
If $d_u < \lfloor \frac{k}{2} \rfloor$: Let $v :=$ choose$(\{w \in N(u) | d_w = d_u - 1\})$
For each $i \in [\![d_u, \lfloor \frac{k}{2} \rfloor - 1]\!], c_{i,u} := c_{i,v}$; $b_{i,u} := b_{i,v}$

Become Leader :: (priority 2)
if $err_u = 0 \wedge (d_u = k - 1) \wedge \forall v \in N(u), d_v = k - 1$
then $d_u := 0$, For each $i \in [\![1, \lfloor \frac{k}{2} \rfloor - 1]\!], c_{i,u} := 0, b_{i,u} := \downarrow$

Leader down :: (priority 1)
if $well\_defined(u) \wedge d_u = 0 \wedge \exists i \in [\![1, \lfloor \frac{k}{2} \rfloor - 1]\!], b_{i,u} = \uparrow$
then For each $i \in [\![1, \lfloor \frac{k}{2} \rfloor - 1]\!]$, $b_{i,u} := \downarrow$

Two Heads:: (priority 1)
if $err_u = 0 \wedge \exists v, v' \in (N(u) \cup \{u\})^2, v \neq v' \wedge d_v = d_{v'} = 0)$
then $err_u := 1$

Branch incoherence:: (priority 1)
if $err_u = 0 \wedge \neg branch\_coherence(u)$
then $err_u := 1$

Error Spread :: (priority 2)
if $err_u = 0 \wedge (d_u \leq \lfloor \frac{k}{2} \rfloor - 1) \wedge (\exists v \in N(u), err_v = 1 \wedge d_u < d_v)$
then $err_u := 1$

Reset Error :: (priority 2)
if $(err_u = 1) \wedge ([d_u > \lfloor \frac{k}{2} \rfloor] \vee [\forall v \in N(u), d_v \geq d_u \vee err_v = 1])$
then $err_u := 0$, If $d_u = 0$, $d_u := 1$, For each i, $c_{i,u} := 0$, $b_{i,u} :=\uparrow$

Figure 4.1: Algorithm for the $(k, k-1)$-Ruling Set

problem. To remedy this, each node maintains a synchronized clock system around each node of $S(\gamma)$ by executing the *stationary rules*. For this reason, we split the set of rules into two groups:

- The *stationary* rules are the rules **Incr Leader**, **Sync 1 down**, **Sync 2+ down**, **Sync 1+ up**, and **Sync end-of-chain**;
- The *convergence* rules are the rules **Remote Collision**, **Two Heads**, **Branch Incoherence**, **Update Distance**, **Become Leader**, **Error Spread**, **Reset Error**, and **Leader down**.

We say that a node in $S(\gamma)$ is the *leader* of the nodes under its influence, corresponding to the nodes closest to it than to any other node of $S(\gamma)$. Assuming d-value has already been spread, the clock of index i of nodes that gave the same leader will always be either equal or out-of-sync by 1. Thus, a node detects that two nodes in $S(\gamma)$ are too close when it sees in its neighborhood two nodes with clocks out-of-sync by 2. It will raise an error when activated by executing **Remote Collision**. The error is then propagated toward the problematic members of $S(\gamma)$ by the rule **Error Spread**.

In both previous cases, the problematic nodes of $S(\gamma)$ end up having err-value 1, which makes them leave $S(\gamma)$ by executing **Reset Error**. Afterward, the rule **Update distance** will, over time, update the d-values of the nodes at distance up to k to that node.

The approach of our algorithm makes sure that when a node is inserted in $S(\gamma)$ and no node gets added at distance at most $k-1$ away, it remains in $S(\gamma)$ forever. Note that when it is executed, the rule **Update distance** setup the clock values and arrows (variables c and b) so that the newly updated node is synchronized to

its "parent" (the node it takes as a reference to update its d-value).

The target configuration is **not** a *stable* configuration, and from it, all the nodes can only execute *stationary rules*. In this configuration, $S(\gamma)$ is guaranteed to be a $(k, k-1)$-ruling set of the underlying graph. Note that the predicate $well\_defined$ appears in the guard of every stationary rule. The predicate guarantees that the considered node neither is in error-detection mode nor has some incorrect d-values in its neighborhood before executing any clock-related rule.

4.2.2 . The Clock System

Now, we describe the clock system used to detect that two leader nodes in $S(\gamma)$ are at a distance less than k. The leaders are the nodes that update the clock value c_i and propagate it to its "children" and so on. For a given clock index i, when every neighbor of a leader s has the same clock c_i and their corresponding arrow b_i pointed up, node s increments its clock value by 1 by executing rule **Incr Leader**.

After that, the clock value is propagated downward (toward nodes of greater d-value) by the rules **Sync 1 down** and **Sync 2+ down**. Note that it's performed locally by layers: one node of a given d-value cannot update its clock value and arrow before every neighbor with a smaller d-value does so. This is necessary to guarantee the global synchronization of the clock.

There are two ways for the propagation to reach the limit of the area it should spread in: either it has reached nodes with d-value i, or there is no node having a greater d-value to spread the clock further.

- In the first case, the rule **Sync end-of-chain** flips the arrow b_i.
- In the second case, the nodes execute rule **Sync 1+ up** to flip b_i.

In both cases, it allows the rule **Sync 1+ up** to propagate upward (toward smaller d-values) with the b_i-value switching to $\uparrow$ from the nodes to their parents. Note that it is done locally by layers: one node of a given d-value may not update its clock value and arrow before every neighbor with a greater d-value has done so.

When the propagation reaches the neighbors of s, node s "detects" that its current clock value has been successfully propagated. It can then again execute **Incr Leader** to increase it.

The point of this clock system is that two nodes under the same leader cannot have clock values out-of-sync by 2, but two nodes that have different leaders may. It allows to detect a "collision" (*i.e.* two nodes of $S(\gamma)$ too close from each other) when the d-values of two such nodes are too small (smaller than $\lfloor \frac{k}{2} \rfloor$). Observe that the clock of index i is only reliable for detecting collision between nodes of $S(\gamma)$ that are at distance $2i$ or $2i+1$ from each other. For smaller distances, this clock may be forcefully synchronized between two nodes of $S(\gamma)$ by the layer-by-

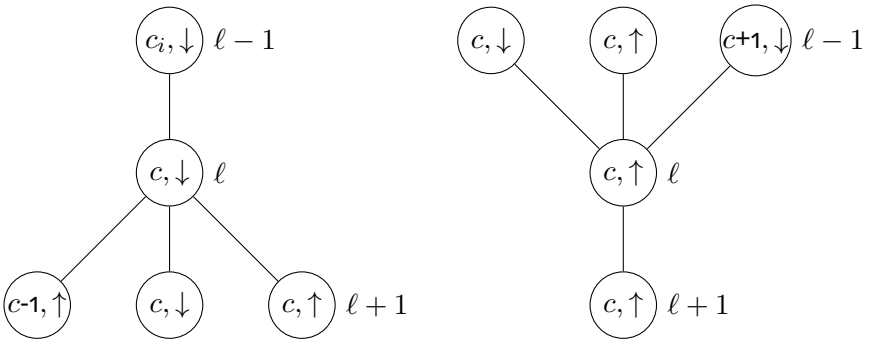


Figure 4.2: Branch coherence condition. The couples $(c, \downarrow)$ or $(c, \uparrow)$ represent the local variables (c_i, b_i) of the nodes. The value on the right of the nodes represents their distance to the leader node (*i.e.* their d-value). The central node in both figures is the reference, and the other nodes represent the possible couples for its neighbors with different d-value.

layer updating, and for greater distances, no node may detect an out-of-sync from it. Thus, we have $\lfloor \frac{k}{2} \rfloor - 1$ parallel clock systems to capture every possible distance of collision.

The Gouda daemon ensures that, if two nodes of $S(\gamma)$ are too close from each other, this will not be the case forever. Indeed, the clock system will eventually detect it and propagate an error.

4.2.3 . Handling Initial and Perturbed Configurations

The rules **Leader Down** and **Branch Incoherence** are only executed to solve problems coming from the initial configuration, or after a perturbation has occurred. The rule **Leader Down** is executed when a leader has some of its arrows b_i in the wrong direction. The rule **Branch Incoherence** is executed when some "impossible" patterns are produced in the clock systems due to wrong clock values and arrows in the initial state. Normal patterns are shown in Figure 4.2, any other pattern will make an activated node execute **Branch Incoherence**.

4.3 . Proof of the Algorithm

4.3.1 . Stability of Legitimate Configurations

The ruling set algorithm presented in this section uses the state model. It constructs the set of vertices whose d-value is 0. We will prove that this set is a ruling set in legitimate configurations. Formally, we require the following

specification for the legitimate configurations:

Definition 4.3.1. Let $S(\gamma)$ be the set of nodes s such that $d_s = 0$ in a given configuration γ. Configuration γ is said to be *legitimate* if:

1. All the nodes u are such that $well\_defined(u)$, $leader\_down(u)$ and $branch\_coherence(u)$ hold;
2. For any two u and v distinct nodes of $S(\gamma)$, we have $dist(u, v) \geq k$.

Theorem 4.3.2. *From any legitimate configuration γ, every reachable configuration is legitimate, and in such a configuration every node has the same d-values as in γ.*

Thanks to Theorem 4.3.2, we know that, from a legitimate configuration, we keep the same set of leaders $S(\gamma)$, which forms a $(k, k-1)$-ruling set. Hence, under the Gouda daemon, the set of leaders will eventually be a stable $(k, k-1)$-ruling set.

The goal of the following lemmas will be to prove Theorem 4.3.2. Lemma 4.3.3 ensures that $S(\gamma)$ forms a ruling set when the values of all the local variables are correct.

Lemma 4.3.3. *Let γ be a legitimate configuration.*

- *For any node u, $d_u = dist(u, S(\gamma))$,*
- *$S(\gamma)$ is a $(k, k-1)$-ruling set of the underlying graph.*

Proof. By definition of the predicate $well\_defined(u)$, if $d_u > 0$, there exists some $v \in N(v)$ such that $d_v = d_u - 1$. Let's prove by induction on $i < k$ that "For any node u, $d_u \leq i \Leftrightarrow dist(u, S(\gamma)) = d_u$":

- If $d_u = 0$, we have $u \in S(\gamma)$, and $dist(u, S(\gamma)) = dist(u, u) = 0$. Conversely, if $dist(u, S(\gamma)) = 0$, then $u \in S(\gamma)$ and $d_u = 0$ by definition of $S(\gamma)$.
- Let's assume that it is true for $i < k - 1$. Let u be a node such that $d_u = i + 1$. Since predicate $well\_defined(u)$ is satisfied in γ, there exists a node $v \in N(u)$ such that $d_v = d_u - 1 = i$. Using induction hypothesis, $dist(v, S(\gamma)) = i$, *i.e.* there exists node s in $S(\gamma)$ such that $dist(s, v) = i$. Hence, $dist(s, u) \leq i + 1$. Using induction hypothesis, $dist(u, S(\gamma)) \leq i$ would imply $d_u = dist(u, S(\gamma)) \leq i$, thus by contraposition $dist(u, S(\gamma)) > i$. Hence $dist(u, S(c)) = i + 1$.

  Conversely, let's assume that $dist(u, S(\gamma)) = i + 1$. By using a shortest path from u to $S(\gamma)$, we can get a node $v \in N(u)$ such that

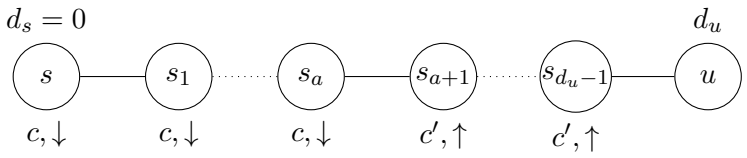


Figure 4.3: Node s propagates its clock value along a shortest path from s to u where $c' \in \{c, c-1\}$.

$dist(v, S(\gamma)) = i$. Hence, by induction, $d_v = i$. As $well\_defined(u)$ is true, we have $|d_u - d_v| \leq 1$ and hence $i - 1 \leq d_u \leq i + 1$. As $dist(u, S(\gamma)) = i + 1 \notin [\![0, i]\!]$, by induction hypothesis we get $d_u > i$. Hence $d_u = i + 1$.

By induction, we proved the first item of the lemma.

Let's prove now that $S(\gamma)$ is a $(k, k-1)$-ruling set. Since every node u has the value of local variable d_u less than $k-1$, the first item of this lemma states that $dist(u, S(\gamma)) \leq k - 1$. Hence, we only need that for any pair $(u, v) \in S(\gamma)^2$ of distinct nodes, $dist(u, v) \geq k$. This is true by definition of a legitimate configuration. Thus, the lemma holds. □

Now we focus on the clock system. We prove the following property on the ruling set to run the clock system.

Lemma 4.3.4. *Let γ be a legitimate configuration and s be a node in $S(\gamma)$. For every node u, $dist(u, s) \leq \lfloor \frac{k}{2} \rfloor$ implies that $d_u = dist(u, s)$.*

Proof. Suppose node u is such that $dist(u, s) \leq \lfloor \frac{k}{2} \rfloor$.

Let's take $s' \in S(\gamma)$ such that $d(u, S(\gamma)) = d(u, s')$. We have $dist(u, s) \leq dist(u, s')$. Then by triangular inequality we have $dist(s, s') \leq dist(s, u) + dist(u, s') \leq 2\, dist(s, u) \leq 2\lfloor \frac{k}{2} \rfloor \leq k - 1$. As $S(\gamma)$ is a $(k, k-1)$-ruling set from Lemma 4.3.3, this means that $s = s'$. □

This property allows us to deduce that a node u such that $dist(u, s) \leq \lfloor \frac{k}{2} \rfloor$ has only one node s of $S(\gamma)$ in its ball at distance $\lfloor \frac{k}{2} \rfloor$. Thus, all the nodes in $\mathcal{B}(s, \lfloor \frac{k}{2} \rfloor - 1)$ must be synchronized with s. Now we explain how the values representing the clock of the local variable of nodes with d-value smaller than $\lfloor \frac{k}{2} \rfloor$ are spread from their leader. Figure 4.3 illustrates how the pairs (c_i, b_i) go from nodes in $S(\gamma)$.

Lemma 4.3.5. *Let γ be a legitimate configuration and s a node in $S(\gamma)$. For every node u such that $dist(u, s) \leq \lfloor \frac{k}{2} \rfloor - 1$, every shortest path $(s_0, s_1, \cdots, s_{d_u})$ from*

s to u satisfies the following property in γ:
For every clock index $i \in [\![d_u + 1, \lfloor \frac{k}{2} \rfloor - 1]\!]$, there exists some integer $a \in [\![0, d_u]\!]$ such that:

1. $\forall \ell \in [\![0, a]\!]$, $(b_{i,s_\ell}, c_{i,s_\ell}) = (\downarrow, c_{i,s})$,
2. $\exists c' \in \{c_{i,s} - 1, c_{i,s}\}, \forall \ell \in [\![a + 1, d_u]\!]$, $(b_{i,s_\ell}, c_{i,s_\ell}) = (\uparrow, c')$.

Proof. For a given integer $\alpha \in [\![0, \lfloor \frac{k}{2} \rfloor - 1]\!]$ we denote by $\mathcal{H}(\alpha)$ the property of the lemma for every node whose distance to s is α.

For $\alpha = 0$, the only path to consider only contains s itself. Consider any clock index $i \in [\![1, \lfloor \frac{k}{2} \rfloor - 1]\!]$. The only possible value for a is 0.

1. By definition of the legitimate configuration, $b_{i,s} = \downarrow$. Thus $(b_{i,s}, c_{i,s}) = (\downarrow, c_{i,s})$, and Point 1 of property $\mathcal{H}(0)$ holds.
2. Point 2 of property $\mathcal{H}(0)$ holds trivially, since $a + 1 > d_u = 0$.

Thus $\mathcal{H}(0)$ holds.

We now suppose $\mathcal{H}(\alpha)$ true for any $\alpha < \delta$ for some $\delta \leq \lfloor \frac{k}{2} \rfloor - 1$. Let then $P = (s = s_0, s_1, \cdots, s_\delta = u)$ be a shortest path from node s to some node u such that $dist(s, u) = \delta$. Since γ is legitimate, $d(u, S(\gamma)) = d(u, s)$ by Lemma 4.3.4 since $\delta \leq \lfloor \frac{k}{2} \rfloor$. Lemma 4.3.3 gives then $d_u = dist(s, u) = \delta$.

By definition of a shortest path, $s_{\delta-1}$ is at distance $dist(s, u) - 1 = \delta - 1$ from s. By the same argument as for u, Lemma 4.3.3 gives $d_{s_{\delta-1}} = \delta - 1$, thus $d_{s_{\delta-1}} < \delta$. Let also i be an integer in $[\![d_u + 1, \lfloor \frac{k}{2} \rfloor - 1]\!]$ be a clock index. Since $dist(s, s_{\delta-1}) < \delta$, we apply the induction hypothesis: the shortest path $(s_0, s_1, \cdots, s_{\delta-1})$ from s to $s_{\delta-1}$ satisfies that there exists some $a \in [0, \delta - 1]$ such that:

1. $\forall \ell \in [\![0, a]\!]$, $(b_{i,s_\ell}, c_{i,s_\ell}) = (\downarrow, c_{i,s})$;
2. $\exists c' \in \{c_{i,s} - 1, c_{i,s}\}, \forall \ell \in [\![a + 1, \delta - 1]\!]$, $(b_{i,s_\ell}, c_{i,s_\ell}) = (\uparrow, c')$.

We treat the cases $a = \delta - 1$ and $a < \delta - 1$ separately, using the same argument that predicate $branch\_coherence\_down(s_{\delta-1}, i)$ is true (as $\delta - 1 < \delta \leq i < \frac{k}{2}$, and γ is legitimate).

Suppose that $a < \delta - 1$. We have $(b_{i,s_{\delta-1}}, c_{i,s_{\delta-1}}) = (\uparrow, c')$ for some $c' \in \{c_{i,s} - 1, c_{i,s}\}$. Since $branch\_coherence\_down(s_{\delta-1}, i)$ is true, $b_{i,s_{\delta-1}} = \uparrow$ implies that $(b_{i,s_\delta}, c_{i,s_\delta}) = (\uparrow, c')$. Thus, $P = (s_0, s_1, \cdots, s_\delta)$ respects the property of the lemma.

Suppose that $a = \delta - 1$. It implies that $\forall \ell \in [\![0, \delta - 1]\!]$, $(b_{i,s_\ell}, c_{i,s_\ell}) = (\downarrow, c_{i,s})$. Since $branch\_coherence\_down(s_{\delta-1}, i)$ is true, we have three possibilities:

- If $(b_{i,u}, c_{i,u}) = (\downarrow, c_{i,s_{\delta-1}})$, we have $(b_{i,u}, c_{i,u}) = (\downarrow, c_{i,s})$. Then $P = (s_0, s_1, \cdots, s_\delta)$ respects the wanted property for $a = \delta$. Note that the second part of the property is trivially true with this value of a for any value c', for the "for all" quantifier acts on the empty set.

- If $(b_{i,u}, c_{i,u}) = (\uparrow, c_{i,s_{\delta-1}})$, we have $(b_{i,u}, c_{i,u}) = (\uparrow, c_{i,s})$. Then, $P = (s_0, s_1, \cdots, s_\delta)$ respects the wanted property for $a = \delta - 1$, taking $c' = c_{i,s}$ for the second part.

- Else, we must have $(b_{i,u}, c_{i,u}) = (\uparrow, c_{i,s_{\delta-1}} - 1)$, and we have $(b_{i,u}, c_{i,u}) = (\uparrow, c_{i,s})$. Then, $P = (s_0, s_1, \cdots, s_\delta)$ respects the wanted property for $a = \delta - 1$, taking $c' = c_{i,s} - 1$ for the second part.

Thus, every shortest path $(s_0, s_1, \cdots, s_\delta)$ from s to every such node u satisfies the property for every possible clock index $i \in [\![d_u + 1, \lfloor \frac{k}{2} \rfloor - 1]\!]$, and $\mathcal{H}(\delta)$ holds.

Thus, by induction, the lemma holds.

□

Lemma 4.3.6 proves that only rules to update clocks can be executed from legitimate configurations:

Lemma 4.3.6. *Let γ be a legitimate configuration. Let u be a node. Node u only executes stationary rules from γ.*

Proof. Let γ be a legitimate configuration. Since every node satisfies predicates $well\_defined$, $branch\_coherence$ and $leader\_down$, none has its err-value equals to 1, and rules **Leader Down**, **Error Spread**, **Reset Error** and **Branch incoherence** cannot be executed.

From Lemma 4.3.3, $S(\gamma)$ is a $(k, k-1)$-ruling set of the underlying graph: each element in $S(\gamma)$ is at distance at least k from another. Thus, there are no two neighboring nodes that have their local variables d equal to 0 and rule **Two Heads** cannot be executed in γ.

Let u be a node. We have $d_u = \min\{dist(u, s) \mid s \in S(\gamma)\}$ by the first point of Lemma 4.3.3. Observe that, by definition of local variable d_u, we have $0 \leq d_u \leq k - 1$.

From predicate $well\_defined$, we know that if $d_u > 0$, u has a neighbor v such that $d_v = d_u - 1$, and that any neighbor w of u are such that $d_u - 1 \leq d_w \leq d_u + 1$. Hence, $d_u = \min(\min\{d_v \mid v \in N(u)\} + 1, k - 1)$. We deduce that rule **Update distance** cannot be executed in γ.

Observe that, if $d_u = k - 1$, u has at least a neighbor v is such that $d_v = d_u - 1 = k - 2$. This implies that no node can activate rule **Become Leader**.

Let u be a node that executes rule **Remote Collision**. We have $d_u \leq \frac{k-1}{2}$, the existence of two nodes v and v' in $N(u) \cup \{u\}$ with $d_v = d_{v'}$, and $|c_{d_v,v} - c_{d_v,v'}| = 2$ (as local variable c_{d_v} is defined, we have $d_v = d_{v'} \leq \lfloor \frac{k}{2} \rfloor - 1$). It means that there exist two nodes s, s' in $S(\gamma)$ such that $dist(v,s) = d_v$ and $dist(u,s') = d_u \leq \frac{k-1}{2}$. As v is in the closed neighborhood of u, we have:

$$dist(s,s') \leq dist(s,v)+1+dist(u,s') \leq d_v+1+d_u \leq (\lfloor \frac{k}{2} \rfloor -1)+1+\frac{k-1}{2} < 2\frac{k}{2}$$

As $S(\gamma)$ is a $(k, k-1)$-ruling set, we get that $s = s'$. By the same reasoning, we get that v' is also at distance $d_{v'}$ to s.

By applying Lemma 4.3.5 for both v and v', we get that $c_{d_v,v}, c_{d_{v'},v'}$ are in $\{c_{d_v,s} - 1, c_{d_v,s}\}$. Thus it implies that $|c_{d_v,v} - c_{d_v,v'}| \leq 1$, which contradicts the execution of rule **Remote Collision**.

This concludes the proof.

□

Once the execution reaches a legitimate configuration γ, we have proved that only stationary rules can be executed. The goal is to use that result and the previous lemmas to prove that only legitimate configurations can be reached from γ. This result will lead to the proof of Theorem 4.3.2.

Proof. of Theorem 4.3.2. Let γ and γ' be two configurations such that γ is legitimate and $\gamma \to \gamma'$. By enumerating cases, we prove that γ' is also legitimate.

Based on the stationary rules, if $d_u = 0$, for any $i < \lfloor \frac{k}{2} \rfloor$, we always have $b_{i,u} = \downarrow$, and if $c_{i,u}$ changes it is to be incremented by 1. If $0 < d_u < \lfloor \frac{k}{2} \rfloor$, for any $i \geq d_u$, if $b_{i,u} = \downarrow$, the only state change can be that $b_{i,u}$ becomes $\uparrow$. If $b_{i,u} = \uparrow$, the only state change can be that $b_{i,u}$ becomes $\downarrow$ and $c_{i,u}$ gets incremented by 1. Thus, we can consider the only possibility for a state change of a given pair $(b_{i,u}, c_{i,u})$. We exhaustively look at all the possible transitions in $N(u) \cup \{u\}$ to check that we still have $branch\_coherence(u)$ and $well\_defined(u)$.

- Since we can only apply stationary rules from γ (Lemma 4.3.3), all the local variables d_u remain constant: $\forall u,\ d_u^{\gamma} = d_u^{\gamma'}$. Hence, for all nodes u, predicate $well\_defined(u)$ is true in γ', and $S(\gamma) = S(\gamma')$.
- Let u be a node. We need to prove that in γ', $branch\_coherence(u)$ is true. As we will consider separately the different clock indexes in our reasoning, when a given clock index i is considered in the context, and x is a node, we will call $(b_{i,x}, c_{i,x})$ the *state* of x. It will make the discussion smoother in the remainder of the proof.
  - If $d_u \geq \lfloor \frac{k}{2} \rfloor$, $branch\_coherence(u)$ is always true in γ', as $d_u^{\gamma'} = d_u^{\gamma}$.

- If $d_u = 0$, let $i \geq 0$. We need to prove $branch\_coherence\_down(u, i)$ in γ'. We know, by the fact that $d_u = 0$ and $leader\_down$, that $b_{i,u} = \downarrow$. Let $v \in N(v)$. Since predicate $well\_defined(u)$ is true, we have $d_v = 1$. By $branch\_coherence\_down(u, i)$, we have $(b_{i,v}, c_{i,v}) \in \{(\uparrow, c_{i,u}), (\uparrow, c_{i,u} - 1), (\downarrow, c_{i,u})\}$ in γ.

  If $c^{\gamma'}_{i,u} = c^{\gamma}_{i,u} + 1$, rule **Incr Leader** has been activated. It means that $\forall v \in N(u)$, $b^{\gamma}_{i,v} = \uparrow$ and $c^{\gamma}_{i,v} = c_{i,v}$. Rule **Sync 1 Down** could not have been activated on v for i as they had the same $c_{i,u}$. Hence $b^{\gamma'}_{i,v} = b^{\gamma}_{i,v} = \uparrow$, and $c^{\gamma'}_{i,v} = c^{\gamma}_{i,v} = c^{\gamma}_{i,u} = c^{\gamma'}_{i,u} - 1$. Predicate $branch\_coherence\_down(u, i)$ is true in γ'.

  If $c^{\gamma'}_{i,u} = c^{\gamma}_{i,u}$, let us enumerate what can have happened to the neighbor v depending on their values of the couple $(b_{i,v}, c_{i,v})$ in γ:

  * $(b_{i,v}, c_{i,v}) = (\uparrow, c_{i,u})$: the states do not change after in the transition, as u's value of local variable $c_{i,u}$ should have been one more.
  * $(b_{i,v}, c_{i,v}) = (\uparrow, c_{i,u} - 1)$: either v is not activated and the state has not changed, either v was activated, and Rule **Sync 1 down** was applied. In that case, the new state in γ' is $(\downarrow, c_{i,u})$.
  * $(b_{i,v}, c_{i,v}) = (\downarrow, c_{i,u})$: if the state of v has changed, it means that Rule **Sync 1+ up** has been applied. In that case, the new state of v in γ' is $(\uparrow, c_{i,u})$.

  In every case, the possibilities in γ' for v are compatible with the state of u.

- If $0 < d_u \leq k/2$, let $i \geq d_u$. We have four possibilities, depending on if $b_{i,u}$ is $\uparrow$ or $\downarrow$, and if u changes its state or not:

  * $b_{i,u} = \uparrow$, and u does not change its state.
    Let $v \in N(u)$ be a node such that $d_v = d_u - 1$. It can only be in states in $\{(\uparrow, c_{i,u}), (\downarrow, c_{i,u}), (\downarrow, c_{i,u} + 1)\}$. In the two first possibilities, if v changes its state, it is still compatible with the state of u. In the third case, v could not change its state because the state of u is not allowing it.
    Let $v \in N(u)$ be a node such that $d_v = d_u + 1$. It can only be in state $(\uparrow, c_{i,u})$. It cannot change its state, because the state of u is not allowing it.
  * $b_{i,u} = \uparrow$, and u changes its state to $(\downarrow, c_{i,u} + 1)$.
    Let $v \in N(u)$ be a node such that $d_v = d_u - 1$. It can only be in state $(\downarrow, c_{i,u} + 1)$, as otherwise u could not have changed. In that situation, the state of u prevents v from changing its own.
    Let $v \in N(u)$ be a node such that $d_v = d_u + 1$. It can only be

in state $(\uparrow, c_{i,u})$. It cannot change its state, because the state of u is not allowing it.

* $b_{i,u} = \downarrow$, and u does not change its state.
  Let $v \in N(u)$ be a node such that $d_v = d_u - 1$. It can only be in states $(\downarrow, c_{i,u})$. It cannot change its state, because the state of u is not allowing it.
  Let $v \in N(u)$ be a node such that $d_v = d_u + 1$. It can only be in states in $\{(\uparrow, c_{i,u} - 1), (\downarrow, c_{i,u}), (\uparrow, c_{i,u})\}$. In the two first possibilities, if v changes its state, it is still compatible with the state of u. In the third case, v could not change its state because the state of u is not allowing it.
* $b_{i,u} = \downarrow$, and u changes its state.
  Let $v \in N(u)$ be a node such that $d_v = d_u - 1$. It can only be in state $(\downarrow, c_{i,u})$. It cannot change its state, because the state of u is not allowing it.
  Let $v \in N(u)$ be a node such that $d_v = d_u + 1$. It can only be in state $(\uparrow, c_{i,u})$, as otherwise u could not have changed. In that situation, the state of u prevents v from changing its own state.

□

4.3.2 . Reaching a Legitimate Configuration

The goal of the following lemmas is to prove that, from any configuration γ, we can reach a configuration γ' that is legitimate. The Gouda daemon's property concludes that a legitimate configuration will always eventually be reached. Indeed, let γ be a configuration that is infinitely often reached during an execution. Under the Gouda daemon, as a legitimate configuration γ' is reachable from configuration γ, γ' will also be reached infinitely often.

To that end, we introduce the notion of *locally legitimate* node for leaders satisfying conditions close to the legitimate ones in their ball of radius $k-1$. We prove that if a node s is locally legitimate then it will remain so forever (Lemma 4.3.10).

We explain how to make locally legitimate a node that does not have any leader at distance smaller than k to it in Lemmas 4.3.12 and 4.3.15. We explain how, when some leaders are too close from each other, we can reach a configuration where none of the remaining ones are at distance smaller than k from another (Lemma 4.3.15).

From here, we can conclude with the proof of the following theorem:

Theorem 4.3.7. *Under the Gouda daemon, any execution eventually reaches a legitimate configuration.*

We first introduce the notion we will use in this section for nodes in $S(\gamma)$:

Definition 4.3.8. Let γ be a configuration. A node s in $S(\gamma)$ is *locally legitimate* when:

1. $\forall u \in \mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$, $d_u = dist(u, s)$, and $well\_defined(u)$, $leader\_down(u)$ and $branch\_coherence(u)$ are true;
2. $\forall u \in \mathcal{B}(s, k-1) \setminus \mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$, $k - dist(u, s) \leq d_u \leq dist(u, s)$.

We denote $\mathcal{LL}(\gamma)$ the set of those nodes in γ.

Let s be a locally legitimate node. The first property means that in its neighborhood at distance at most $\lfloor \frac{k}{2} \rfloor$, nodes behave like in a legitimate configuration. Therefore, they cannot detect errors. The second property implies that all nodes in $\mathcal{B}(s, k-1)$ have coherent d-values according to s and to potential leaders that are at distance at least k from s. A direct observation is the following:

Lemma 4.3.9. *Let $s \in \mathcal{LL}(\gamma)$. We have $\mathcal{B}(s, k-1) \cap S(\gamma) = \{s\}$.*

Proof. From the definition of local legitimacy, every node in $\mathcal{B}(s, k-1) \setminus \{s\}$ has positive d-value. □

Combining Lemma 4.3.9 and the first property of the legitimated node, we can deduce that once a node is legitimate, it remains legitimate during the rest of the execution.

Lemma 4.3.10. *Let γ, γ' be two configurations such that $\gamma \to \gamma'$, $\mathcal{LL}(\gamma) \subset \mathcal{LL}(\gamma')$.*

Proof. Suppose that node s is locally legitimate in γ, let's prove that if $\gamma \to \gamma'$ then s is also locally legitimate in γ'.

- Since s is locally legitimate in γ, the configuration obtained by restricting γ to nodes $\mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$ is a legitimate configuration. By Lemma 4.3.6, only stationary rules can be applied to those nodes in that restricted configuration. Since nodes in $\mathcal{B}(s, \lfloor \frac{k}{2} \rfloor - 1)$ can only see nodes in $\mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$, the rules enabled for those nodes are the same in the restricted configuration and in γ, thus only stationary rules can be enabled on $\mathcal{B}(s, \lfloor \frac{k}{2} \rfloor - 1)$ in γ.
- For nodes at distance $\lfloor \frac{k}{2} \rfloor$ from s, since their d-value is exactly $\lfloor \frac{k}{2} \rfloor$, they cannot execute any of the rules **Error Spread**, **Become Leader**, **Two Heads**, **Branch incoherence** nor any stationary rule. They also have at least one neighbor at distance $\lfloor \frac{k}{2} \rfloor - 1$ from s with d-value $\lfloor \frac{k}{2} \rfloor - 1$. Moreover, we have these two following points:

1. The neighbors of those nodes in $\mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$ have d-value equal to their distance to s, *i.e.* $\lfloor \frac{k}{2} \rfloor$ or $\lfloor \frac{k}{2} \rfloor - 1$;
2. The neighbors of those nodes in $\mathcal{B}(s, k-1) \setminus \mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$ have distance to s at least $\lfloor \frac{k}{2} \rfloor + 1$ and from local legitimacy their d-value is at least $k - (\lfloor \frac{k}{2} \rfloor + 1)$ which is $\lfloor \frac{k}{2} \rfloor - 1$ when k is even, and $\lfloor \frac{k}{2} \rfloor$ when k is odd.

From these facts:

- Rule **Update distance** is not enabled on those nodes;
- For rule **Remote Collision**, the only possibility for it to be enabled would be the first case in the guard when k is odd. Suppose then k is odd and u is a node at distance $\lfloor \frac{k}{2} \rfloor$ from s, with two distinct neighbors v, v' such that $d_v = d_{v'} = d_u - 1 = \lfloor \frac{k}{2} \rfloor - 1$. As k is odd, neighbors of u in $\mathcal{B}(s, k-1) \setminus \mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$ have d-value at least $\lfloor \frac{k}{2} \rfloor$ (see Point 2 above). Hence, nodes v, v' should be in $\mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$. But local legitimacy guarantees predicate *branch_coherence* on every node in $\mathcal{B}(s, \lfloor \frac{k}{2} \rfloor - 1)$ which guarantees that the clock $c_{\lfloor \frac{k}{2} \rfloor - 1, \cdot}$ of two such nodes could not be out of sync by 2. Thus, **Remote Collision** is not enabled on u.

And finally, since those nodes are supposed well-defined, **Reset error** is not enabled on them. Those nodes are then not activable.

- For nodes in $\mathcal{B}(s, k-1) \setminus \mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$, the condition on their d-value prevents them from executing any stationary rule, on any of the rules **Remote Collision**, **Two Heads**, **Branch incoherence**, and **Become leader**. The only potentially enabled rules are **Update distance**, **Error Spread**, and **Reset Error**.

Then, we focus on properties in configuration γ'.

- As it was done before to analyze the enabled rules in $\mathcal{B}(s, \lfloor \frac{k}{2} \rfloor - 1)$, we consider the legitimate configuration obtained by restricting γ to nodes $\mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$. By Theorem 4.3.2, any transition from the restricted configuration would still be legitimate. Since nodes in $\mathcal{B}(s, \lfloor \frac{k}{2} \rfloor - 1)$ can only see nodes in $\mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$, and since nodes of $\mathcal{B}(s, \lfloor \frac{k}{2} \rfloor) \setminus \mathcal{B}(s, \lfloor \frac{k}{2} \rfloor - 1)$ were not activated in the transition, the nodes changed state in the transition from γ to γ' as they would have from the restricted configuration. Thus the restriction of γ' to $\mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$ is a legitimate configuration, and $well\_defined(.)$ and $branch\_coherence(.)$ are still true on nodes of $\mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$ in γ'.

- Consider $u \in \mathcal{B}(s, k-1) \setminus \mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$. If u has not changed its d-value in the transition, there is nothing to prove. If the d-value of u changed in $\gamma \to \gamma'$, it must have performed rules **Update distance** or **Become Leader** (**Reset Error** may only change d-value of nodes that have an original d-value of 0).

  Rule **Become Leader** could not have been applied, as u has a neighbor v closer to s by 1, which implies that $d_v \leq dist(v, s) < k - 1$.

  Let's prove that rule **Update distance** could not have been applied either. Every $v \in \mathcal{B}(s, k-1) \setminus \mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$ is such that $k - dist(u, s) \leq d_v \leq dist(v, s)$ in γ from local legitimacy (see Definition 4.3.8), and the same can be true for the nodes exactly at distance $\lfloor \frac{k}{2} \rfloor$. Then, as neighbors of u are at distance at least $dist(u, s) - 1$ at most $dist(u, s) + 1$ from s, the minimum d-value in the neighborhood of u in γ is in $[\![k - (dist(u, s) + 1), dist(u, s) - 1]\!]$. Then if u has executed **Update distance** in the transition, its new d-value must be at least $k - (dist(u, s) + 1) + 1 = k - dist(u, s)$, and at most $dist(u, s) - 1 + 1 = dist(u, s)$, thus $\lfloor \frac{k}{2} \rfloor < d_u \leq dist(u, s)$ in γ'.

□

We focus now on how to create locally legitimate nodes. First of all, we can make sure that the d-values of all the nodes are coherent with regards to their distance to $S(\gamma)$:

Lemma 4.3.11. *For any configuration γ, we can reach a configuration γ' such that $S(\gamma) = S(\gamma')$, and $d_u = \min(dist(u, S(\gamma')), k - 1)$ for every node u, and there is no node with err-value 1 among nodes with d-value greater than $\lfloor \frac{k}{2} \rfloor$.*

Proof. Let's prove the following property by induction on $i < k - 1$:

From any configuration γ, we can reach a configuration γ' where $S(\gamma) = S(\gamma')$, and, for all nodes $u \in V$ we have $dist(u, S(\gamma')) \leq i \iff d_u \leq i \iff d_u = dist(u, S(\gamma'))$.

- For the case $i = 0$, this is true by definition of $S(\gamma')$.
- Let's assume that the property is true for some $i \leq k - 3$.

  Let u be a node such that $dist(u, S(\gamma')) = i + 1$. In its neighborhood, there is a node v such that $dist(v, S(\gamma')) = i$. By the induction hypothesis, $d_v = dist(v, S(\gamma'))$. Moreover, by the induction hypothesis, no node w in $N(u)$ is such that $d_w < d_v$. If it was the case, u would be at distance at most $d_w + 1$ from $S(\gamma')$, but $d_w + 1 < d_v + 1 = dist(u, S(\gamma'))$.

  Hence, if $d_u \neq dist(u, S(\gamma'))$, by activating u, rule **Update distance** can be executed, and d_u would become $d_v + 1$ as $d_v < k - 1$.

Let u be a node such that $d_u = i + 1$ and $dist(u, S(\gamma')) > i + 1$. Since all of its neighbors are at distance greater than i to $S(\gamma')$, the induction hypothesis implies that their d-value is at least $i + 1$. Thus if u gets activated, it will execute rule **Update distance**. After that, we have $d_u > i + 1$.

From the current configuration γ, by activating all the nodes u at distance $i + 1$ to $S(\gamma)$ such that $d_u \neq i + 1$ and all the nodes at distance greater than $i+1$ such that $d_u = i+1$, we reach a configuration γ' where the induction property is true for $i + 1$.

Note that we never activate any node u such that $d_u = k - 1$ and for all $v \in N(u)$, $d_v = k - 1$. Hence, $S(\gamma') = S(\gamma)$.

The induction being verified, we can reach a configuration γ' where, for all nodes, $d_u < k - 1 \iff d_u = dist(u, S(\gamma'))$. Hence, in this configuration, all nodes at distance at least $k - 1$ have $d_u = k - 1$. □

Let s be a node at distance at least k from $S(\gamma)$. We explain how to make that node locally legitimate:

Lemma 4.3.12. *Let γ be a configuration where there exists a node s such that $dist(s, S(\gamma)) \geq k$ then a configuration γ' can be reached from γ such that $s \in \mathcal{LL}(\gamma')$.*

Proof. By applying Lemma 4.3.11, we can reach a configuration γ'' where, for each node $u \in V$, $d_u = \min(dist(u, S(\gamma'')), k - 1)$. In particular, it means that $d_s = k - 1$, and it is also the case for its neighbors.

Observe that all nodes u in $\mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$ have its local value d_u greater that $\lfloor \frac{k}{2} \rfloor$. Then, in γ'', if one of these nodes has err-value equal to 1, it can execute rule **Reset Error** which does not change its d-value. We can reach a configuration with the same property as γ'' on d-values, without a node having $err_u = 1$ in $\mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$. Now, we can apply rule **Become Leader** to s.

For any integer i from 1 to $k - 2$, we do a transition where we activate nodes at distance i from s, to reach some configuration γ'.

For the first $\lfloor \frac{k}{2} \rfloor$ steps, the nodes activated have their d-value inferior to $\lfloor \frac{k}{2} \rfloor$ before their activation (otherwise, we would have $d_s < k - 1$). For those nodes, rule **Update distance** is executed. We get, for each node u activated at step i, $d_u = i = dist(u, s)$ after the transition corresponding to step i. For each $j \in [\![d_u, \frac{k}{2} - 1]\!], c_{j,u} := 0$; $b_{i,u} :=\downarrow$. For all $i \leq \lfloor \frac{k}{2} \rfloor - 1$, Property 1 of Definition 4.3.8 is satisfied.

Let u be a node at distance $\lfloor \frac{k}{2} \rfloor$ from s. Let v be a neighbor of u such that $d_v = \lfloor \frac{k}{2} \rfloor - 1$. Note that u is at distance d_v from s (otherwise, s would have

been at distance $< k$ from a node in $S(\gamma)\setminus\{s\}$). Hence u also satisfies Property 1 of Definition 4.3.8.

Let $u \in \mathcal{B}(s, k-1) \setminus \mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$. By direct induction, we can see that after step $dist(u,s)-1$, u has a neighbor v closer to s such that $d_v \leq dist(v,s) = dist(u,s)-1$. After applying rule **Update distance** in step $dist(u,s)$, we get that $d_u \leq d_v + 1 \leq dist(u,s)$. Moreover, we need to prove that $d_u \geq k - dist(u,s)$. In γ'', as each node is such that its d equals its distance to $S(\gamma'')$, no node v in $\mathcal{B}(s, k-1) \setminus \mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$ is such that $d_v < k - dist(v,s)$, otherwise $dist(s, S(\gamma))$ would be smaller than k, which contradicts the premise of the lemma.

Let $i \leq k-2$ be the first step where a node u updates its distance such that $d_u < k - dist(u,s)$. It would mean that it has a node v in its neighborhood such that $d_v < k - dist(u,s) - 1$ in the previous step. We have $dist(v,s) \leq dist(u,s)+1$. Moreover, v cannot be at a closer distance to s, otherwise i would not be minimal. Hence, v was not updated in the steps from 1 to $i-1$, meaning that $d_v = dist(v, S(\gamma''))$. As $dist(v,s) \geq \lfloor \frac{k}{2} \rfloor$, there exists some $s' \in S(\gamma'')$ different from s such that $dist(v,s') = d_v$. We obtain that $dist(s,s') \leq dist(v,s) + dist(v,s') < dist(u,s) + 1 + k - dist(u,s) - 1 < k$, which contradicts the premise of the lemma.

This concludes that s is locally legitimate in the last configuration after the $k-1$ steps.

□

Now, we need to deal with leaders that are too close from each other. To do this, we introduce the function that measures the number of nodes in this situation in a configuration, and Lemma 4.3.14 shows how to decrease it.

Definition 4.3.13. When γ is a configuration, we define $\phi(\gamma)$ as the set of leaders in γ having a conflict with another one due to being at distance less than k to each other, *i.e.* $\phi(\gamma) = \{u \in S(\gamma) \mid \exists v \in S(\gamma) \setminus \{u\}, dist(u,v) < k\}$.

Lemma 4.3.14. *Let γ be a configuration such that $\phi(\gamma) \neq \emptyset$. There exists a node u in $\phi(\gamma)$ and a configuration γ' such that we can reach γ' from γ with $S(\gamma') = S(\gamma) \setminus \{u\}$.*

Proof. Using Lemma 4.3.11, we can reach a configuration γ^* from γ where for each node $u \in V$, $d_u = \min(dist(u, S(\gamma^*)), k-1)$. From now, we suppose that each node $u \in V$ is such that $d_u = \min(dist(u, S(\gamma)), k-1)$ in γ.

Let u be a node in $\phi(\gamma)$. The definition of $\phi(\gamma)$ states that a node v in $S(\gamma) \setminus \{u\}$ is at distance at most k from u. Thus, v is also in $\phi(\gamma)$, and $\phi(\gamma)$ have at least two elements.

Let $u, v \in \phi(\gamma)$ such that $dist(u, v)$ is minimum among the pairs of distinct nodes of $\phi(\gamma)$, and let us denote $\delta = dist(u, v)$. From definition of function ϕ, $dist(u, v) = \delta < k$, and we have $\lceil \frac{\delta}{2} \rceil \leq \lfloor \frac{k}{2} \rfloor$ with equality when $\delta = k - 1$.

Since v is the closest leader to u in $\phi(\gamma)$, every node in $\mathcal{B}(u, \lceil \frac{\delta}{2} \rceil)$ has its distance to u as d-value. Symmetrically, we can apply the same argument for nodes in $\mathcal{B}(v, \lceil \frac{\delta}{2} \rceil)$ relatively to v.

If a node w in $\mathcal{B}(u, \lceil \frac{\delta}{2} \rceil)$ is such that $err_w = 1$, then all its neighbors with a smaller d-value can execute rule **Error Spread**. By following a shortest path from w to u, we can reach a configuration where $err_u = 1$. When we reach a configuration where $err_u = 1$, u can execute rule **Reset Error**, and after its execution, we can reach a configuration γ' where $d_u = 1$, thus $S(\gamma') = S(\gamma) \setminus \{u\}$.

Symmetrically, using the same argument as previously, if a node w in $\mathcal{B}(v, \lceil \frac{\delta}{2} \rceil)$ is such that $err_w = 1$, we can reach a configuration γ' such that $S(\gamma') = S(\gamma) \setminus \{v\}$.

Let us then suppose that there is no node in $\mathcal{B}(u, \lceil \frac{\delta}{2} \rceil) \cup \mathcal{B}(v, \lceil \frac{\delta}{2} \rceil)$ such that it err-value is equal to 1.

Consider a shortest path P from u to v. We consider two cases according to the parity of its length.

- When $\delta = 2i$ with $i \in \mathbb{N}$, $P = (u = u_0, u_1, .., u_i = v_i, .., v_1, v_0 = v)$. Nodes in $\mathcal{B}(u, i) \setminus \mathcal{B}(u, i-1)$ have d-value i and have err-value 0 by hypothesis. Thus, executing stationary rules, we can make the clock with index i go to 0 for every node of $\mathcal{B}(u, i-1)$. Symmetrically, executing stationary rules, we can make the clock with index $i-1$ go to 2 for every node of $\mathcal{B}(v, i-1)$.

- When $\delta = 2i+1$ with $i \in \mathbb{N}$, $P = (u = u_0, u_1, .., u_i, v_i, .., v_1, v_0 = v)$. Note that nodes in $\mathcal{B}(u, i+1) \setminus \mathcal{B}(u, i)$ have either their d-value equal to $i+1$ or i, since it would otherwise imply that another node of $S(\gamma)$ is closer to u than v, they also have err-value 0 by hypothesis. Then, executing stationary rules, we can make the clock with index in i go to 0 for every node of $\mathcal{B}(u, i)$. Symmetrically, executing stationary rules, we can make the clock with index in i go to 2 for every node of $\mathcal{B}(v, i)$.

In both cases, nodes u_i can execute rule **Remote Collision** and then we can make the error propagate toward u executing rule **Error Spread** and reach a configuration γ' where $S(\gamma') = S(\gamma) \setminus \{u\}$. □

Thanks to this result, we prove that we can reach a configuration γ such that the set of conflicting nodes is empty:

Lemma 4.3.15. *From any configuration γ, we can reach a configuration γ' such that $\phi(\gamma') = 0$.*

Proof. Lemma 4.3.14 states that, from any configuration γ, we can reach a configuration γ' where $|\phi(\gamma')| < |\phi(\gamma)|$. This lemma can be proved by applying Lemma 4.3.14 at most $|\phi(\gamma)|$ times. □

Now we focus on how to make leaders locally legitimate if they do not have any other leaders at distance smaller than k from them.

Lemma 4.3.16. *Let γ and s be a configuration and a node such that $\mathcal{B}(s, k-1) \cap S(\gamma) = \{s\}$. We can reach a configuration γ' such that $s \in \mathcal{LL}(\gamma')$.*

Proof. First, let's apply Lemma 4.3.11, to ensure that all nodes have their distance to $S(\gamma)$ up to date.

We can assume that $s \notin \mathcal{LL}(\gamma)$, as otherwise we take $\gamma' = \gamma$. The goal in the following proof is, in each considered case, to reach a configuration γ'' such that $s \notin S(\gamma'')$ (and $S(\gamma) = S(\gamma'') \cup \{s\}$). Then, by applying Lemma 4.3.12 on γ'', we can reach a configuration in which node s is locally legitimate. This actually means that we need to reach a configuration where $d_s > 0$, while no other node u changes its d_u.

As $s \notin \mathcal{LL}(\gamma)$, by definition of $\mathcal{LL}$, we have three possible scenarios:

- There exists a node u in $\mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$ such that $\neg well\_defined(u)$. As all the distances to $S(\gamma)$ are correct, it means that $err_u = 1$. Let's choose such a node u that minimizes its distance to s.

  If $u = s$, we can apply rule **Reset Error**, which removes directly s from $S(\gamma)$.

  Otherwise, there is a path $(u = u_0, u_1, \dots u_{d_u} = s)$ from u to s. By activating u_i for i from 1 to d_u, rule **Error Spread** will be applied each time, putting all those nodes in an error state. After that, we activate again s, which will remove it from $S(\gamma)$.

- There exists a node u in $\mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$ such that $\neg branch\_coherence(u)$. We activate that node, which will make an error appear with rule **Branch incoherence**, and we go back to the previous case.

- There exists a node u in $\mathcal{B}(s, k-1) \setminus \mathcal{B}(s, \lfloor \frac{k}{2} \rfloor)$ such that $d_u < k - dist(u, s)$ or $d_u > dist(u, s)$. We prove that it cannot actually happen, as we have $d_u = dist(u, S(\gamma))$ and $\mathcal{B}(s, k) \cap S(\gamma) = \{s\}$. We cannot have $d_u > dist(u, s)$, as $d_u = dist(u, S(\gamma)) \leq dist(u, s) \leq k - 1$. Suppose that we have $k - dist(u, s) > d_u$. It implies, as $dist(u, s) > \lfloor \frac{k}{2} \rfloor$, that $d_u \leq \lfloor \frac{k}{2} \rfloor <$

$dist(u,s)$, which means that there exists some $s' \neq s$ in $S(\gamma)$ such that $dist(u,s') = d_u$. We have

$$dist(s,s') \leq dist(u,s) + dist(u,s') < dist(u,s) + k - dist(u,s) < k.$$

Thus $dist(s,s') \leq k-1$, which is a direct contradiction with $\mathcal{B}(s,k-1) \cap S(\gamma) = \{s\}$.

Hence, in the previous scenarios, we managed to reach a configuration γ' such that $S(\gamma'') = S(\gamma) \setminus \{s\}$. From this configuration, using Lemma 4.3.12, we can reach a configuration γ' such that $s \in \mathcal{LL}(\gamma')$.

We proved that we can always reach a configuration where s joins $\mathcal{LL}(\gamma)$. □

Now, we can prove that the number of legitimate nodes increases during the execution until we converge to a legitimate configuration:

Lemma 4.3.17. *Let γ be a configuration. From γ, we can reach a configuration γ' such that either $\mathcal{LL}(\gamma) \subsetneq \mathcal{LL}(\gamma')$, or γ' is legitimate.*

Proof. Using Lemma 4.3.15, from γ, we can reach a configuration γ''' such that $\phi(\gamma''') = 0$. Then, using Lemma 4.3.11 on γ''', we can reach γ'' such that $S(\gamma'') = S(\gamma''')$ (hence $\phi(\gamma'') = \phi(\gamma''') = \emptyset$), and every node satisfies $d_u = \min(dist(u, S(\gamma'')), k-1)$. If $\mathcal{LL}(\gamma'')$ is a $(k,k-1)$-ruling set, we can reach a legitimate configuration γ' using Lemma 4.3.11. Let's then suppose it's not the case.

- If $S(\gamma'') \setminus \mathcal{LL}(\gamma'') \neq \emptyset$, let us consider $u \in S(\gamma'') \setminus \mathcal{LL}(\gamma'')$. Since $\phi(\gamma'') = \emptyset$, we know that $\mathcal{B}(u,k-1) \cap S(\gamma'') = \{u\}$. Using Lemma 4.3.16 we can reach a configuration γ' such that $u \in \mathcal{LL}(\gamma')$.
- Else, as the nodes of $S(\gamma'') = \mathcal{LL}(\gamma'')$ do not form a $(k,k-1)$-ruling set, there exists a node $u \notin S(\gamma'')$ such that $dist(u, S(\gamma'')) \geq k$. We can then apply Lemma 4.3.12 to reach a configuration γ' where $u \in \mathcal{LL}(\gamma')$.

In both cases, since $u \notin \mathcal{LL}(\gamma'')$, we know that $u \notin \mathcal{LL}(\gamma)$ as local legitimacy cannot be lost from Lemma 4.3.10. Thus $\mathcal{LL}(\gamma) \subsetneq \mathcal{LL}(\gamma')$. □

This last lemma allows us to conclude with the proof of Theorem 4.3.7.

Proof. of Theorem 4.3.7. Let γ be a configuration that is reached infinitely often under the Gouda daemon. We prove that γ is legitimate.

Indeed, by applying Lemma 4.3.17, either γ is legitimate, or we can reach a configuration γ' such that $\mathcal{LL}(\gamma) \subsetneq \mathcal{LL}(\gamma')$. In the second case, by the Gouda

daemon, γ' is reached infinitely often. By Lemma 4.3.10, $\mathcal{LL}(\gamma')$ can only increase from γ'. Hence, we will no longer be able to reach γ, which means that γ is not reached infinitely often.

Hence, γ is legitimate. □

4.4 . From Ruling Sets to Distance-K Colorings

In this section, we focus on the *distance-K coloring* problem. A distance-K coloring is a coloring such that any pair of nodes cannot share a color unless they are at distance greater than K. If the nodes having the same color form a $(K+1, K)$-ruling set, then those nodes respect the coloring constraint.

Let choose $k > K$ for our $(k, k-1)$-ruling sets. We split the set of nodes into pairwise disjoint sets such that each set corresponds to nodes of the same color. We partition the nodes into sets $S^{(i)}$ we build one after another. Each of these sets is a distance-k independent set of the graph, which is maximal among the nodes of $V \setminus \bigcup_{j<i} S^{(j)}(\gamma)$. These sets will be built by composing an adaptation of our $(k, k-1)$-ruling set algorithm. Since the maximum degree of the graph is Δ, any ball of radius $k-1$ contains at most $\Delta^{k-1}+1$ nodes. Hence we can partition the nodes into Δ^k ruling sets (we use this upper bound to simplify the reading of the following proofs).

For this reason, the distance K-coloring algorithm is composed of Δ^k parallel algorithms, each one of them computing an adapted $(k, k-1)$-ruling set. For Algorithm i and configuration γ, we note $S^{(i)}(\gamma)$ (or $S^{(i)}$ if there is no ambiguity) the corresponding set $S(\gamma)$. Each time a node u is activated, it applies a rule (if it can) for each ruling set algorithm.

It is necessary to manage that a node must belong to only one ruling set. To perform this, we number the ruling set algorithms: we denote by $d_u^{(j)}$ the local variable d_u of u of the j-th algorithm. By convention, we assume that u belongs to the j-th ruling set (or it has color j) if $j = \min\left\{1 \leq p \leq \Delta^k \mid d_u^{(p)} = 0\right\}$. To form a partition with the sets, we need to reach a configuration where, for each node u, $|\left\{i \leq \Delta^k \mid d_u^{(i)} = 0\right\}| = 1$. To achieve this, we modify rule **Become Leader** and add a rule to detect if a node is a leader in different layers (for Algorithm j).

Become Leader$^{(j)}$ :: (priority 1)
if $err_u^{(j)} = 0 \wedge (d_u^{(j)} = k-1) \wedge \forall v \in N(u), d_v^{(j)} = k-1 \wedge \forall p < j : d_u^{(p)} > 0$
then $d_u^{(j)} := 0$
$\forall i \in [\![1, \lfloor \frac{k}{2} \rfloor - 1]\!], c_{i,u}^{(j)} := 0, b_{i,u}^{(j)} :=\downarrow$

Belong To Two ruling sets$^{(j)}$ :: (priority 0)
if $d_u^{(j)} = 0 \wedge \exists p < j : d_u^{(p)} = 0$

then $d_u^{(j)} := 1$

We also modify the predicate $well\_defined$ (for Algorithm j) as follows, which impacts the definition of legitimate configuration. In particular, now, a node u such that $d_u^{(j)} = k-1$ is allowed not to have a neighbor closer to a leader if there exists some $i < j$ such that $d_u^{(i)} = 0$.

well_defined$^{(j)}(u) \equiv err_u^{(j)} = 0 \wedge \forall v \in N(u), |d_u^{(j)} - d_v^{(j)}| \leq 1 \wedge$
$((\forall p \leq j, d_u^{(p)} > 0) \vee d_u^{(j)} < k-1 \Rightarrow (\exists v \in N(u), d_v^{(j)} = d_u^{(j)} - 1)) \wedge$
$(d_u^{(j)} = 0 \Rightarrow \forall p < j, d_u^{(p)} > 0)$

We give a new definition of *legitimate configuration*:

Definition 4.4.1. Let $j \leq \Delta^k$. A configuration γ is said to be *legitimate for Algorithm j* if, for all $i \leq j$:

1. all the nodes u are such that $well\_defined^{(i)}(u)$, $leader\_down^{(i)}(u)$ and $branch\_coherence^{(i)}(u)$ hold;
2. for any $u \neq v$ in $S^{(i)}(\gamma)^2$, we have $dist(u,v) \geq k$.

From this, we get the following adaptation of Lemma 4.3.3. The proof remains slightly the same, with the exception that in the case of $d_u^{(j)} = k-1$, only nodes that do not have a variable $d_u^{(i)} = 0$ for some $i < j$ are considered.

Lemma 4.4.2. *Let γ be a legitimate configuration for Algorithm j.*

- *For any node u, if for all $i < j$, $d_u^{(i)} > 0$, we have $d_u^{(j)} = dist(u, S^{(j)}(\gamma))$;*
- *$S^{(j)}(\gamma)$ is a $(k, k-1)$-ruling set of $V \setminus \bigcup_{i<j} S^{(i)}(\gamma)$.*

With these modifications, we have the following adaptation of Theorem 4.3.2:

Theorem 4.4.3. *For all $j \leq \Delta^k$, the set of legitimate configurations for Algorithm j is closed. Moreover, from a legitimate configuration γ for Algorithm j, all the $d^{(j)}$-value do not change.*

Proof. We prove this theorem by induction on j. The base case $j = 1$ is proved by Theorem 4.3.2. Suppose that the property is true for some $j < \Delta^k$, and we have a legitimate configuration γ for the first $j+1$ algorithms. By induction, we know that the configurations we can reach from γ do not change the $d^{(i)}$-values for $i \leq j$, and they are legitimate for Algorithm i.

The rule **Belong To Two ruling sets**$^{(i)}$ cannot be applied for $i \leq j+1$, as we have $well\_defined^{(i)}$. The only change that can happen is about rule **Become Leader**$^{(i)}$, which can only happen less often.

The difference from the proofs of the previous section is that we have nodes with a $d^{(j+1)}$-value that is $k-1$ without a $k-2$ in their neighborhood. We use the fact that this happens only if their $d^{(i)}$-value is equal to o for some $i \leq j$. As this value cannot change, by the induction, $d^{(j)}$ will not change either.

Hence, the set of legitimate configurations for Algorithm $j+1$ is closed, concluding the proof.

□

The proof to reach a legitimate configuration for Algorithm Δ^k works in the same way as the proof of Theorem 4.3.7. We need to do it one algorithm after another, from 1 to Δ^k. The main difference is that we only consider nodes that are not a leader in a smaller algorithm when we increase the set of locally legitimate nodes. This leads to the result:

Theorem 4.4.4. *Under the Gouda daemon, any execution eventually reaches a legitimate configuration in Algorithm Δ^k.*

These two theorems lead to the main result of distance-K coloring:

Theorem 4.4.5. *Let k and K be two integers such that $k > K$. Under the Gouda daemon, any execution eventually reaches a configuration γ such that*

- $S^{(i)}(\gamma) = \left\{u \mid d_u^{(i)} = 0\right\}$ *forms a distance-k maximal independent set of $V \setminus \bigcup_{j<i} S^{(j)}(\gamma)$ in G.*
- *The sets $S^{(1)}(\gamma), \dots S^{(\Delta^k)}(\gamma)$ form a distance-K coloring.*
- *Every configuration in any execution starting in γ verifies the two above properties with the same sets as γ.*

4.5 . Solving Mendable Problems

In this section, we want to solve a generalization of *Greedy Problems*: $O(1)$-*Mendable Problems*, introduced in [5]. Greedy problems, such as $\Delta+1$-coloring and Maximal Independent Set, have the property that, if some of the nodes have chosen an output that is locally valid (no pair of neighbors sharing a color, no adjacent nodes selected in the set), then any single node can choose an output that will keep the global solution locally valid. In a distributed setting, we cannot do this process sequentially one node after another, but we can do it in parallel:

if a set of nodes that are far enough from each other choose their output at each step, the solution can be completed. If we repeat this process until all nodes have chosen an output, the global solution is valid.

To that end, we first introduce some definitions.

4.5.1 . Definitions

We call a *Locally Checkable Problem* (Lcl) Π a problem where each node can check locally that its output is compatible with its neighbors. Let $\mathcal{O}$ be the set of outputs. The output $\Gamma : V \to \mathcal{O}$ is good if and only if, for all $u \in V$, $\Gamma(u)$ is compatible with the multiset $\{\Gamma(v) \mid v \in N(u)\}$. For example, in the case of Maximal Independent Set, with $\mathcal{O} = \{0, 1\}$, 1 is compatible with $\left\{0^k \mid k \leq \Delta\right\}$, and 0 is compatible with $\{11^x0^y \mid x + y < \Delta\}$. Note that we can consider radius-r neighborhood for the compatibility in the general case, which we will not do here out of simplicity. Our results can be adapted to the general version.

Let $\mathcal{O}$ be the set of outputs, and $\Gamma^* : V \to \mathcal{O} \cup \{\perp\}$. We say that Γ^* is a partial solution if, for any $u \in V$ such that $\Gamma^*(u) \neq \perp$, we can complete the labels of the neighbors v of u (i.e. give an output to the nodes v such that $\Gamma^*(v) = \perp$) to make u compatible with it neighbors.

A problem is T-*mendable* if, from any partial solution Γ^* and any $u \in V$ such that $\Gamma^*(v) = \perp$, there exists a partial solution Γ' such that:

- $\Gamma'(v) \neq \perp$
- $\forall u \neq v$, $\Gamma'(v) = \perp \Leftrightarrow \Gamma^*(v) = \perp$
- $\forall u \in V$, $dist(u, v) > T \Rightarrow \Gamma'(u) = \Gamma^*(u)$

Intuitively, we can change the output of nodes at distance at most T from a node v when we select the output of v.

The Local *model* is a synchronous model where each node is given a unique identifier. As there is no limit on the size of the messages for communication, after r rounds, each node knows the topology of its neighborhood at distance r.

We use Theorem 6.2 from [5], that states:

Theorem 4.5.1. *Let Π be a T-mendable* Lcl *problem. Π can be solved in $O(T\Delta^{2T})$ rounds in the* Local *model if we are given a distance-$(2T + 1)$ coloring.*

An important fact in the Local model to solve an Lcl problem is that unique identifiers are not necessary, as long as nodes do not see twice the same identifier during their run. If we know that an algorithm runs on a graph of size at most n in $r(n) = o(\log n)$ rounds, then we can have it run on any graph of size at least n with a distance-$r(n)$ coloring, using those colors as the new identifiers.

The algorithm will not notice that the identifiers are not unique, producing correct output. This technique has been used, for example, in [5, 12].

Hence, for a constant T, we can produce a distance-$r(T)$ coloring to then use the algorithm of Theorem 4.5.1.

4.5.2 . Solving Greedy and Mendable Problems

The goal now is to use distance-k colorings to solve other problems. Let us say that we want to solve Π on some out-value. To that end, we will have to couple a self-stabilizing version of that algorithm on a distance-k coloring (for some well-chosen k) to the algorithm computing the distance-k coloring described in Section 4.4. To ensure that the coloring is solved before we start solving Π, when a node u executes a rule of the ruling set algorithms, out_u is reset to $\perp$. If a node realizes that its out-value is not compatible with its neighbors, it returns its out-value to $\perp$.

As a first example to show how the technique works, we show how to produce, a $(\Delta+1)$-coloring and a MIS from a distance-k coloring for $k \geq 2$. The idea is to go through each color class one after another:

Proposition 4.5.2. *Let γ be a configuration where each node u has a color c_u corresponding to a distance-k coloring for $k \geq 2$ and outputs $out_u = \perp$. From this configuration, under the Gouda daemon, we will reach a configuration γ' where each node outputs a color $\leq \Delta+1$. Moreover, the nodes of color 1 form a Maximal Independent Set.*

Proof. The algorithm only uses the following rule:

Select Color ::
if $out_u = \perp \wedge \forall v \in N(u), (c_v > c_u \vee out_v \neq \perp)$
then $out_u := \min(\mathbb{N}^* \setminus \{c_v \mid v \in N(u)\})$

Each node only needs to change its state following rule **Select Color**. We are sure that no pair of neighbors will select their color simultaneously, as one of them has a color greater than the other. Moreover, as each node selects the smallest available color, we are sure that each node selects color 1 or has color 1 in its neighborhood.

Finally, only nodes with minimal color among their neighbors that did not output yet can be activated. Once activated, they produce their output, and rule **Select Color** will no longer be applied to them. Hence, under the Gouda daemon, local minima will eventually be activated. The only configurations where no node can be activated are those where they all have produced an output.

□

To simulate r rounds in the Local model, we need, to compute the topology of the graph at distance r for each node. If we have beforehand a distance-$(2r+1)$ coloring, each node has at most one node of some given color in its neighborhood at distance r. Hence, each node can know its neighborhood at distance r. In the beginning, each node knows its neighborhood at distance 0. If all the neighbors of a node u know their mapping at distance i, u can deduce its own topology up to distance $i+1$. Note that we consider only cases where r does not depend on the number of nodes in the graph. Hence, for a fixed Δ, there is a finite number of balls of radius r using Δ^{2r+1} colors.

Lemma 4.5.3. *Let γ be a configuration where each node u has a color c_u corresponding to a distance-$(2r+1)$ coloring and outputs $out_u = \perp$. From this configuration, under the Gouda daemon, we will reach a configuration γ' where each node outputs a mapping of their neighborhood at distance r.*

Proof. The algorithm uses the two following rules:

Init ::
if $out_u = \perp$
then $out_u := (0, G_0(c_u))$ where $G_0(x)$ is a graph of a single node colored x

Merge Neighbors ::
if $out_u = (i, G_u)$ with $i < r \wedge \forall v \in N(v), \exists j \geq i, out_v = (j, G_v)$
then $out_u := (i+1, merge_{i+1}(\{G_v \mid v \in N(u) \cup \{u\}\}))$

Where the process $merge_{i+1}(\{G_v \mid v \in N(u) \cup \{u\}\})$ consists in merging the mappings at distance i of u and the ones of its neighbors to produce the mapping at distance $i+1$. This can be done unambiguously as the distance-$(2r+1)$ coloring ensures that if several mappings have a node of color c, it corresponds to a single node of V.

Under the Gouda daemon, we can make sure that, for any $i \leq r$, we can reach a configuration where all nodes have computed their mapping at distance i.

□

With this lemma and Theorem 4.5.1, we can conclude the end result of this section:

Theorem 4.5.4. *Let Π be an* Lcl *problem with mending radius k, that can be solved in $r = O(k\Delta^{2k})$ rounds in the* Local *model. Let γ be a configuration where each node u has a color c_u corresponding to a distance-$(2k+1)$ coloring, a color c'_u corresponding to a distance-$(2r+1)$ coloring, and outputs $out_u = \perp$. From this configuration, under the Gouda daemon, we will reach a configuration γ' where each node outputs a solution to Π.*

4.6 . Conclusion

This chapter provides a self-stabilizing algorithm to compute a $(k, k-1)$-ruling set under the Gouda daemon. This construction generalizes well to probabilistic daemons if stationary rules and rule **Become Leader** have some probability smaller than 1 to be activated. The question holds on whether it is possible to produce this kind of algorithm with even more restrictive daemons. This algorithm permits building up distance-k colorings, which helps solve greedy and mendable problems by simulating the Local model. As we know now that it is possible to solve these problems, the question of complexity might arise.

5 - Minimal Clique Decomposition

The (Vertex) Clique cover problem is a well-known NP-complete problem among Karp's 21 in its minimization version [46]. It has, among other things, been used to approximate Vertex Cover [21].

Definition 5.0.1 (Clique cover)**.** Consider a graph $G = (V, E)$, a *clique cover* $\mathcal{X}$ = $\mathcal{X}_1, \mathcal{X}_2, \dots \mathcal{X}_\ell$ of G is such that:

- ***clique:*** $\mathcal{X}_i$ is a clique of G (see 2.2.4),
- ***cover:*** $\bigcup_{i \in [\![1,\ell]\!]} \mathcal{X}_i = V$.

In centralized computing, it is equivalent to the problem of Clique Decomposition where you expect the cover to be also a partition of V. Any clique decomposition is a clique cover, and to find a clique decomposition it is enough to remove duplicate nodes from one clique until there is no such duplicate left. However, it is not the case in distributed computing, as choosing which duplicate to remove needs coordination. In this chapter, we will study the Minimal Clique Decomposition problem, specified as follows.

Definition 5.0.2 ($\mathcal{MCD}$ Specification)**.** Consider a graph $G = (V, E)$, a *minimal clique decomposition* $\mathcal{X}$ = $\{\mathcal{X}_1, \mathcal{X}_2, \dots \mathcal{X}_\ell\}$ of G is such that:

- ***clique:*** $\mathcal{X}_i$ is a clique of G (see 2.2.4),
- ***partition:*** $\mathcal{X}$ is a partition of V (see 2.1.1),
- ***minimal:*** For any i, j distinct in $[\![1, \ell]\!]$, $\mathcal{X}_i \cup \mathcal{X}_j$ is not a clique of G.

It is equivalent to the Coloring problem as it is enough to replace the edge set by its complement to go from one problem to the other. Or so is the case when you consider centralized computing. In distributed systems where you search for the solution of a graph problem on the underlying graph of the said system, the connectivity directly affects the computations. Here, we deal with the problem of finding a minimal clique decomposition of the graph of communications under the state model, with the possible presence of Byzantine nodes.

5.1 . State of the art

Searching for cliques already has a long history in distributed systems: building upon the work in parallel computing [20], Jennings and al. [44] give a distributed algorithm to find all maximal cliques in a graph, in the context of a message-passing model with message of size $O(\log n)$. It has been followed by many other works improving the performance of such a search (see [51, 65] for example).

Closer to Clique Decomposition, Ishii and Kakugawa [43] give in a self-stabilizing algorithm that operates in the state model under the unfair centralized daemon to compute multiple cliques of "maximal size" for each node under some constraints. In this work, there are agreement constraints between nodes, as if a node computes a given clique, every member of that clique must compute the same clique (among other cliques they may have computed), it is thus closer to the problem of finding a clique decomposition.

The first attempt to our knowledge to tackle the Minimal Clique Decomposition problem in a distributed self-stabilizing setting has been made by Delbot and al. [22]. Their algorithm operates under a fair distributed daemon in $O(n)$ rounds provided a spanning tree has been computed beforehand.

In this chapter, we follow these footsteps by proposing the first algorithm that tackles the Minimal Clique Decomposition problem while handling Byzantine faults.

Algorithm 1 Minimal Clique Decomposition Algorithm ($\mathcal{MCD}$)

Variables:
$\Omega_v \subseteq V$: the set of nodes supposed to be the clique v belongs to

$\mathcal{N}_v \subseteq V$: the (closed) neighborhood made apparent for the neighbors to read

$\beta_v \in N(v) \cup \{\perp\}$: the current merge target for the clique leader v

Funtions:

$$min(A) \equiv \begin{cases} \text{the smallest value of a set A} & \text{if } A \neq \emptyset \\ \perp & \text{otherwise} \end{cases}$$

$leader(v) \equiv min(\Omega_v)$
$merge\_candidate(v) \equiv$
$$\left\{ u \in N(v) | merge\_ready(u) \wedge \Omega_u \cap \Omega_v = \emptyset \wedge \left(\Omega_v \cup \Omega_u \subseteq \bigcap_{x \in \Omega_v \cup \Omega_u} \mathcal{N}_x \right) \right\}$$

$choose(A)$ is an element of the non-empty set A taken uniformly at random.

$\Omega^0(v) \equiv \{v\}$
$\Omega^{k+1}(v) \equiv \bigcup_{x \in \Omega^k(v)} \Omega_x$
$\Omega^*(v) \equiv \bigcup_{i \geq 0} \Omega^i(v)$: the Ω-closure of node v

Predicates:

$$merge\_ready(v) \equiv (leader(v) = v) \wedge Stab(v) \wedge coherent\_clique(v)$$

$$Stab(v) \equiv \forall x \in \Omega_v, \Omega_x = \Omega_v$$

$$coherent\_local(v) \equiv \mathcal{N}_v = N(v) \cup \{v\} \wedge \{v\} \subset \Omega_v \subset \mathcal{N}_v \wedge \beta_v \in \{\perp\} \cup \mathcal{N}_v \setminus \Omega_v$$

$coherent\_clique(v) \equiv$
$$\Omega^*(v) \subseteq \mathcal{N}_v \quad \wedge |\{x \in \Omega^*(v) | \beta_x \neq \perp\}| \leq 1$$
$$\wedge \forall x \in \Omega^*(v), \begin{cases} \Omega^*(v) \subseteq \mathcal{N}_x \\ \{x\} \subseteq \Omega_x \subseteq \Omega_{leader(x)} = \Omega^*(x) \\ \forall y \in \Omega_x, \Omega_y \subseteq \Omega_x \vee \Omega_y = \Omega^*(x) \\ \beta_x \neq \perp \Rightarrow (Stab(x) \wedge leader(x) = x \wedge \beta_x \in \mathcal{N}_x \setminus \Omega_x) \\ \min \Omega^*(v) \in \left\{ leader(leader(x)), \beta_{leader(x)} \right\} \end{cases}$$
$well\_defined(v) \equiv coherent\_local(v) \wedge coherent\_clique(v)$

Abandonment ▷ priority 5
if $(\beta_v = u \neq \bot) \wedge (\beta_u \notin \{\bot, v\} \vee u \notin merge\_candidate(v))$ **then**
$\beta_v = \bot$
end if

Mariage ▷ priority 4
if $leader(v) = v \wedge Stab(v) \wedge (\beta_v = \bot) \wedge (\exists u \in merge\_candidate(v), \beta_u = v)$ **then**
$\beta_v := choose(\{u \in merge\_candidate(v) | \beta_u = v\})$
end if

Seduction ▷ priority 4
if $leader(v) = v \wedge Stab(v) \wedge (\beta_v = \bot) \wedge (\forall u \in merge\_candidate(v), \beta_u \neq v) \wedge (\exists u \in merge\_candidate(v), \beta_u = \bot)$ **then**
$\beta_v := choose(\{u \in merge\_candidate(v) | \beta_u = \bot\})$
end if

Merge lead ▷ priority 4
if $leader(v) = v \wedge Stab(v) \wedge \beta_v = u \neq \bot \wedge u \in merge\_candidate(v) \wedge \beta_u = v \wedge v < u$ **then**
$\Omega_v := \Omega_v \cup \Omega_u$
$\beta_v := \bot$
end if

Merge follow ▷ priority 4
if $leader(v) = v \wedge Stab(v) \wedge \beta_v = u \neq \bot \wedge leader(u) = u \wedge v \in \Omega_u \wedge \Omega^*(u) \subseteq \mathcal{N}_v \wedge coherent\_clique(u)$ **then**
$\Omega_v := \Omega_u$
$\beta_v := \bot$
end if

Update ▷ priority 2
if $leader(v) = u \wedge \Omega_v \subsetneq \Omega_u$ **then**
$\Omega_v := \Omega_u$
end if

Reset ▷ priority 1
if $\neg well\_defined(v)$ **then**
$\Omega_v := \{v\}$
$\mathcal{N}_v := N(v) \cup \{v\}$
$\beta_v := \bot$
end if

5.2.1 . Local variables

Each node v has three variables $\mathcal{N}_v$, Ω_v and β_v:

- $\mathcal{N}_v$ represents its neighborhood,
- Ω_v corresponding to the clique it belongs to,
- β_v represents its current target for merging cliques if there is one. Else it has value $\perp$.

5.2.2 . About the Ω-closure, Ω^*

The Ω-closure of a node v, $\Omega^*(v)$, is the clique that v will have in its variable Ω_v when the potential merging process currently going is finished. Note that a locally coherent node v (such that $coherent\_local(v)$ is true) can always see -and thus read variables of- every node of its Ω-closure, as if it was not the case $\Omega^*(v)$ would not be a clique containing v. From the local coherence $\mathcal{N}_v = N(v) \cup \{v\}$ the node v can read the variables of the nodes of Ω_v. Then v can check whether the nodes of $\Omega^2(v)$ (which we can compute since v can read variables of Ω_v) have their Ω-values included in $\mathcal{N}_v$. Then v can do the same for $\Omega^3(v)$ and so on. Until $\Omega^*(v) \subseteq \mathcal{N}_v$ is proven false or v stops seeing new nodes, in which case we do have that $\Omega^*_v \subseteq \mathcal{N}_v$.

5.2.3 . How to merge two cliques

Each clique has a distinguished node called the leader corresponding to the one with the smallest identifier. Only the clique leader can decide with which clique it will merge. The leader v of a clique starts looking for a merge target as soon as all nodes in its clique have the same view: the predicate $Stab(v)$ is satisfied ($\forall x \in \Omega_v, \Omega_x = \Omega_v$). When it is the case, v seeks a suitable clique leader u to merge clique with. In order for v to merge with u, the local variables of all nodes in Ω_u must have the following properties:

- All nodes of this clique have the same view of this clique: the predicate $Stab(u)$ is satisfied (*i.e.* $\forall x \in \Omega_u, \Omega_x = \Omega_u$).
- All nodes of this clique and the clique of v form a clique in the graph: $\Omega_v \cup \Omega_u \subseteq \bigcap_{x \in \Omega_v \cup \Omega_u} \mathcal{N}_x$.
- The clique is not merging with another clique : $\beta_u = \perp$.

If all these properties are satisfied, v can propose clique merging. To do so, it changes the value of β_v to u, the leader of one of the cliques suitable for merging with, chosen at random. Then, u answers or does not respond positively to this proposal. Of course, it checks that the first two conditions above are verified

too. Note that it is basically applying locally the algorithm proposed by Kunne and al. [47] on the well-formed nodes, hence the borrowed names for the rules dedicated to this: **Seduction**, **Marriage** and **Abandonment**. Once the two leaders agree to merge ($\beta_v = u$ and $\beta_u = v$) the merging process begins. The one among u and v with the smallest identifier changes its variable Ω value to the union of the two cliques, and sets its variable β to $\perp$ by executing rule **Merge lead**. Then the other one updates its two local variables by executing rule **Merge follow**. All the other nodes refresh their variables by executing rule **Update** to complete the merging process.

5.2.4 . How to handle errors

Now we describe how the algorithm handles errors in the local variables and avoids creating new ones when cliques merge. This part of the explanation of the algorithm is the most technical.

Each time a node v is activated, it checks whether it detects any inconsistencies in its local variables (for example the local variable $\mathcal{N}_v$ must correspond to its closed neighborhood.) or those of its current clique nodes (for example the clique must be included in the displayed neighborhood $\mathcal{N}$ of every node of the clique). If it detects any inconsistency that way, it executes the rule **Reset**.

The predicate $coherent\_local$ allows a node v to detect that its local variables are well initialized : $\mathcal{N}_v = N(v) \cup \{v\}$ and $\{v\} \subseteq \Omega_v \subseteq \mathcal{N}_v$. Since variable β_v designates the clique's leader with whom the clique Ω_v must merge, β_v must not be in Ω_v. Note that $coherent\_local(v)$ can only be computed by v itself since it needs to know $N(v)$.

The predicate $coherent\_clique$ allows a node to check that the state of the (future) clique of a node v is coherent (assuming local coherence for all the nodes involved). Note that a locally coherent node u may only evaluate $coherent\_clique(v)$ when $\Omega^*(v) \subseteq \mathcal{N}_u$. As we have said above, it is always the case when $u = v$. It will otherwise only be evaluated when u considers v as a potential target to merge their cliques. In such a case, we will have checked that $\Omega^*(v) \subseteq \mathcal{N}_u$ beforehand, and there will be no problem. We structure the explanation of $coherent\_clique(v)$ by following the structure of the predicate for better intelligibility:

- $\Omega^*(v) \subseteq \mathcal{N}_v$: As $\Omega^*(v)$ is supposed to be either the future clique of v (or current if there is no merging in progress), this condition is necessary for $\Omega^*(v)$ to be a clique containing v.
- $|\{x \in \Omega^*(v) | \beta_x \neq \perp\}| \leq 1$: If multiple β-values were non-$\perp$ in $\Omega^*(v)$, it would mean that multiple merging processes are taking place at the same time. As the algorithm waits for a clique to have finished merging before making it merge again, we do not want that.

- Then v checks all nodes x of its Ω-closure for potential inconsistencies:
  - $\forall x \in \Omega^*(v), \Omega^*(v) \subseteq \mathcal{N}_x$: $\Omega^*(v)$ is supposed to be a clique, thus every node of $\Omega^*(v)$ must have every other node of $\Omega^*(v)$ in its neighborhood.
  - $\forall x \in \Omega^*(v), \{x\} \subseteq \Omega_x \subseteq \Omega_{leader(x)} = \Omega^*(x)$: As leaders are those that move first when merging, we have $\Omega_x \subseteq \Omega_{leader(x)}$. $\{x\} \subseteq \Omega_x$ is just a part of the local coherence of x that happens to be checkable by neighbors. As $x \in \Omega^*(v)$, we have $\Omega^*(x) \subseteq \Omega^*(v)$, and thus if a node can see every node of $\Omega^*(v)$, it can see every node of $\Omega^*(x)$. Observe that $\Omega^*(x)$ may be different from $\Omega^*(v)$ in the case of an x that is part of a clique whose leader has yet to execute **Merge follow**. Finally, there are two cases for the leader of x:
    * It has already the final value as Ω-value (it has already executed **Merge lead** or **Merge follow**) and in this case $\Omega^*(x) = \Omega^*(v)$.
    * It is waiting to execute **Merge follow** and thus still has its old clique value.

    In both cases $\Omega_{leader(x)} = \Omega^*(x)$.
  - $\forall x \in \Omega^*(v), \forall y \in \Omega_x, \Omega_y \subseteq \Omega_x \vee \Omega_y = \Omega^*(x)$: When a clique merging begins, every node in one of the old cliques have only two possible values: the old, and the new one. The old being included in the new one. Note that either $\Omega^*(x) = \Omega^*(v)$, or we are in a case where the leader of x is waiting to execute **Merge follow**.
  - $\forall x \in \Omega^*(v), \beta_x \neq \bot \Rightarrow (Stab(x) \wedge leader(x) = x \wedge \beta_x \in \mathcal{N}_x \setminus \Omega_x)$: Only a leader may have a non-$\bot$ β-value, and it should be in a valid shape to have such a value. It means that x should be its own leader, having every node in its clique having the same Ω-value, and having a merge target β_x that is not already in its clique. $\beta_x \neq \bot$ can either happen for $x = \min \Omega^*(v)$ if the clique of v is ready to merge, or to another node that would be the leader of the clique $\min \Omega^*(v)$ is merging with (which has yet to execute **Merge follow**).
  - $\forall x \in \Omega^*(v), \min \Omega^*(v) \in \{leader(leader(x)), \beta_{leader(x)}\}$: $\min \Omega^*(v)$ will be the leader of the future clique of v, $\Omega^*(v)$, when every node in it has updated its Ω-value. The case $\min \Omega^*(v) = leader(leader(x))$ corresponds to two types of situations. The first one is when either x has been part of the clique that started the merging process with **Merge lead** (it was already true before the merging in this case). The second is when x has been part of the the other clique involved in the merging and the leader of that second clique has already performed **Merge follow**. The case $\min \Omega^*(v) = \beta_{leader(x)}$ corresponds to situations where $leader(x)$ is the leader of that other clique, but has yet

to perform **Merge follow**. If $\min \Omega^*(v)$ is equal to neither of those two options, it means that the clique x belongs to is not really aware that it should be merging. It may happen as a result of errors in the starting configuration.

The predicate $coherent\_clique$ will be used to check for local coherence in the predicate $well\_defined$, and also to avoid merging with nodes that can be detected as not well defined (because $coherent\_clique$ is false on them).

5.3 . Convergence

To prove the convergence of our algorithm, we present our reasoning in five steps.

First, in Subsection 5.3.1, we observe that after at most 1 round the variable $\mathcal{N}$ of non-Byzantine nodes contains the closed neighborhood of the node (Lemma 5.3.2), and it cannot change afterward (Lemma 5.3.3). As such we can, without loss of generality, only consider $\mathcal{N}$-stabilized configurations in the remaining of the proof.

In Subsection 5.3.2, we discuss the properties of well-definedness (corresponding to the $well\_defined$ predicate). We use this to define V_1'' (Definition 5.3.5) a superset of V_1 (recall that V_1 is the set of the nodes that have no Byzantine neighbors) on which well-definedness is guaranteed (Lemma 5.3.6). We prove the convergence of our algorithm on V_1'', as it would not be possible on V_1. on which we can hope to define a convergence property for our algorithm.

In Subsection 5.3.3, we focus on the merging of two cliques. We prove that when a merging between two nodes of V_1'' has been started, they do not interact with nodes outside their cliques while the process is not finished (Lemma 5.3.10), and it ends after a few rounds (Lemmas 5.3.12 and 5.3.14). It allows us to focus on the events that lead nodes to begin such a merging.

In Subsection 5.3.4, we focus on the progression of the algorithm. To do so we define a notion corresponding to the "current" state of the clique decomposition (Definition 5.3.16). Using this, we then prove a succession of lemmas that draw a pattern by which the decomposition progresses probabilistically (summarized in Figure 5.1).

Finally, in Subsection 5.3.5 using a concentration inequality, we deduce that our algorithm converges, and ends within $O(\Delta n)$ rounds with high probability (Theorem 5.3.27).

5.3.1 . Neighborhood stabilization

Definition 5.3.1. We say that a configuration is *$\mathcal{N}$-stabilized* when every non-Byzantine node has its $\mathcal{N}$-value equal to its actual closed neighborhood (*i.e* $\forall x \in V_0, \mathcal{N}_x = N(x) \cup \{x\}$).

It is a condition needed for the rules to behave as intended, and as such we need to know when we can ensure that the condition is met.

Lemma 5.3.2. *Let γ be a configuration. The configuration γ' reached after one round from γ is $\mathcal{N}$-stabilized.*

Proof. **Reset** is the highest priority rule, and it is enabled on any node that does not have its closed neighborhood as $\mathcal{N}$-value. After it has been executed on a non-Byzantine node, this node will have its closed neighborhood as $\mathcal{N}$-value, and it is the only rule that may change a $\mathcal{N}$-value.

Then, after at most one round, every node that didn't have its closed neighborhood as $\mathcal{N}$-value has then been activated and performed **Reset**. As no node having its closed neighborhood as $\mathcal{N}$-values may change that property, it follows that γ' is $\mathcal{N}$-stabilized. □

Then, it is easy to see that a $\mathcal{N}$-stabilized configuration will stay $\mathcal{N}$-stabilized across a transition.

Lemma 5.3.3. *Let γ be a $\mathcal{N}$-stabilized configuration, and $\gamma \to \gamma'$ be a transition. Then γ' is $\mathcal{N}$-stabilized.*

Proof. No rule may change the $\mathcal{N}$-value of a node that has its $\mathcal{N}$-value match its actual closed neighborhood. □

From any configuration, after one round a $\mathcal{N}$-stabilized configuration is reached, and after that, in the execution, every transition will be $\mathcal{N}$-stabilized. Using this fact, in most of the lemmas, we will suppose that we start directly in a $\mathcal{N}$-stabilized configuration without loss of generality.

5.3.2 . Well-definedness

The $well\_defined$ predicate expresses a bunch of "good" properties that we would like to be true, in the sense that it would be always true if we started from a clean starting configuration without Byzantine nodes and where every node u would be such that $\Omega_u = \{u\}$, $\mathcal{N}_u = N(u) \cup \{u\}$ and $\beta_u = \bot$. As it is not the case, we cannot hope for it to be true everywhere and every time. But we can try to understand when it's the case.

First, we note that the $well\_defined$ property is inherited by nodes that are in the Ω-closure of a $well\_defined$ node.

Lemma 5.3.4. *Let γ be a $\mathcal{N}$-stabilized configuration and u, v two nodes such that $u \in \Omega^*(v)$. In γ, if v is well-defined, u is well-defined too.*

Proof. Let u and v be two nodes such that $u \in \Omega^*(v)$ and suppose that v is well-defined. Since v is supposed well-defined, we have either $\Omega^*(u) = \Omega^*(v)$ or $\beta_{leader(u)} = leader(v)$. Since γ is supposed $\mathcal{N}$-stabilized we have $\mathcal{N}_u = N(u) \cup \{u\}$, and $coherent\_clique(v)$ implies that $\{u\} \subseteq \mathcal{N}_u \subseteq \Omega^*(u)$ and $\Omega^*(v) \subseteq \mathcal{N}_u$. Thus $\{u\} \subseteq \Omega_u \subseteq \mathcal{N}_u$. It also implies that $\beta_u \in \mathcal{N}_u \setminus \Omega_u$. Thus, $coherent\_local(u)$ is true.

Let's then prove $coherent\_clique(u)$:

- Suppose $\Omega^*(u) = \Omega^*(v)$. As $coherent\_clique(v)$ is true, it only remains to prove $\Omega^*(u) \subseteq \mathcal{N}_u$, $\Omega_{leader(u)} = \Omega^*(u)$ and $|\{x \in \Omega^*(u) | \beta_x \neq \perp\}| \leq 1$. Using the hypothesis $\Omega^*(u) = \Omega^*(v)$ and the fact that $u \in \Omega^*(v)$, they are direct consequences of $coherent\_clique(v)$.
- Suppose now that $\Omega^*(u) \neq \Omega^*(v)$. It implies $\Omega^*(u) \subsetneq \Omega^*(v)$. Then, using the well-definedness of v we get:
  - $\Omega^*(u) \subsetneq \Omega^*(v) \subseteq \mathcal{N}_u$;
  - Since $\beta(u) \neq \perp$, from the fact that $|\{x \in \Omega^*(v) | \beta_x \neq \perp\}| \leq 1$, u is the only node in $\Omega^*(v)$ to have a non-$\perp$ value of β. Thus, as $\Omega^*(u) \subsetneq \Omega^*(v)$, we get $|\{x \in \Omega^*(u) | \beta_x \neq \perp\}| = 1 \leq 1$;
  - The first four properties stated in the quantified part does not depend on v, thus they are still true for every node of $\Omega^*(u) \subsetneq \Omega^*(v)$. The fifth one comes from the fact that as $\beta(u) \neq \perp$, we have $Stab(u)$ and $leader(u) = u$, thus $leader(leader(.))$ has value $u = min\Omega^*(u)$ for every node of $\Omega^*(u) = \Omega_u$.

  Thus, $coherent\_clique(u)$ is true.

Thus, in every case, u is well-defined. □

To contain the influence of Byzantine nodes, we must identify on which space we want our algorithm to converge. V_0 is out of the picture as a node neighbor to two Byzantine nodes could be fooled by those pretending to be neighbors of each other. The next set to naturally consider is V_1, but as nodes of V_1 may legitimately have some nodes of V_0 in their Ω-value it is not possible to take exactly V_1. We have to widen a bit V_1 in order to include those legitimately included V_0 nodes. As we just proved that $well\_defined(v)$ implies that all the nodes of $\Omega^*(v)$ are well-defined as well, we might want to consider $V_1' = \Omega^*(V_1)$.

As it is easier for us to handle well-defined nodes, we define the following:

Definition 5.3.5. $V_1'' = \Omega^*(\{u \in V_1 | well\_defined(u)\})$.

We will later on prove that after some time it must contain V_1. Do note that contrary to V_1, V_1'' depends on the variables' values of a configuration. We write $V_1''(\gamma)$ for "V_1'' in configuration γ" when there could be some ambiguity.

As we constructed it with this in mind, let's prove that every node of V_1'' is indeed well-defined.

Lemma 5.3.6. *Let γ be a $\mathcal{N}$-stabilized configuration. Every node of V_1'' is well-defined.*

Proof. Let u be a node of V_1''. By definition, it must be in $\Omega^*(v)$ for some well-defined $v \in V_1$. Then by Lemma 5.3.4, as v is well-defined, u must also be well-defined. □

To be a concept of use to express the convergence of an algorithm, we need V_1'' to be non-decreasing.

Lemma 5.3.7. *Let γ be a $\mathcal{N}$-stabilized configuration, and $\gamma \xrightarrow{t} \gamma'$ a transition. Consider $u \in V$, well-defined in γ, such that $leader(u) = u$ and Ω_u does not contain Byzantine nodes.*

1. *If $\neg Stab(u)$, u is well-defined in γ';*
2. *Else, if u does not perform a move in the transition, u is well-defined in γ';*
3. *Else, if $\beta_u = \perp$, u is well-defined in γ';*
4. *Else, if u performs **Abandonment** in the transition, u is well-defined in γ';*
5. *Else, if Ω_{β_u} does not contain Byzantine nodes, u is well-defined in γ'.*

Proof. Let γ be a $\mathcal{N}$-stabilized configuration, and $\gamma \xrightarrow{t} \gamma'$ a transition. Consider $u \in V$ such that u is well-defined in γ, $leader(u) = u$ and Ω_u does not contain Byzantine nodes. Let us first prove Point 1.

Proof of Point 1. Suppose $\neg Stab(u)$. With the well-definedness of u, it implies that none of **Mariage**, **Seduction**, **Abandonment** or **Merge lead** is enabled on any node of $\Omega^*(u) = \Omega_u$ in γ (recall that no node of Ω_u^γ is Byzantine by hypothesis).

Using one of these rules, the Ω-value of a node may only grow by setting it to the value of its current leader (or to the target of the merge in progress in the case of the only potential node being enabled for the rule **Merge follow**) which is already a subset of $\Omega^*(u)$ in γ. As the value of Ω_u does not change in the transition since u is its own leader, the value of $\Omega^*(u)$ does not change in the transition.

Then, let's prove what is needed for $coherent\_clique$ to be true in γ' (we cut the proof according to the structure of the predicate to make the reading easier):

- As $\Omega^*(u)$ does not change in the transition $\Omega^*(u) \subseteq \mathcal{N}_u$ is still true in γ'.
- As $|\{x \in \Omega^*(v) | \beta_x \neq \bot\}| \leq 1$ and no rule that may give a non-$\bot$ value of β is activable on any node of $\Omega^*(u)$ in γ, the same can be said in γ'.
- Let $x \in \Omega^*(u)$,
  - Neither $\Omega^*(u)$ nor $\mathcal{N}_x$ may change in the transition, thus as it was true in γ from well-definedness, $\Omega^*(u) \subseteq \mathcal{N}_x$ is still true in γ'.
  - As the Ω-value of a well-defined node may only change by taking the current Ω-value of its leader, $\{x\} \subseteq \Omega_x \subseteq \Omega_{leader(x)} = \Omega^*(x)$ in γ implies the same in γ'.
  - Let y be a node of Ω^γ_x, we have in γ that $\Omega_y \subseteq \Omega_x \vee \Omega_y = \Omega^*(x)$ in γ by well-definedness. If none of x and y are activated in the transition, or if $x = y$, there is nothing to prove, suppose then that they are distinct and at least one of them is activated.
    * Suppose $\Omega_y \subseteq \Omega_x$ in γ. If only x was activated this remains true in γ', and there is nothing to prove. We then suppose y is activated in the transition.
      · If $\Omega^*(x) = \Omega^*(y)$ in γ, y must have performed **Update** in the transition (as $\Omega_y \subsetneq \Omega_x$ implies $\neg Stab(y)$). As well-definedness implies that $\Omega_x = \Omega^*(x)$ and $\Omega_{leader(y)} = \Omega^*(x) = \Omega_x$ in γ, we have $\Omega^{\gamma'}_y = \Omega^\gamma_x \subseteq \Omega^{\gamma'}_x$.
      · Else, we have $\Omega^*(y) \subsetneq \Omega^*(x) \subseteq \Omega^*(v)$. Well-definedness implies $\min(\Omega^*(v)) \in \{leader(leader(y)), \beta_{leader(y)}\}$ in γ. The previous inequality implies that $leader(leader(y)) \neq \min(\Omega^*(v))$ in γ, thus we have $\beta_{leader(y)} = \min(\Omega^*(v))$ in γ. Well-definedness implies $Stab(leader(y))$, and thus since we supposed that y is activated in the transition we must have $y = leader(y)$, and y performed **Merge follow** in the transition. We can also deduce that $\Omega^*(x) = \Omega^*(v)$ from the fact that it would otherwise imply that $leader(x) \neq y$ has a non-$\bot$ value of β which would contradict well-definedness. Thus, in γ', $\Omega_y = \Omega_x = \Omega^*(v)$.
    * Else, we have $\Omega_y = \Omega^*(x)$. If $\Omega^*(x) = \Omega^*(v)$, there is nothing to prove as Ω-values may only grow in the transition and $\Omega^*(v)$ does not change. Suppose then that $\Omega^*(x) \neq \Omega^*(v)$, *i.e.* $\Omega^*(x) \subsetneq \Omega^*(v)$. This implies by well-definedness that

$\beta_{leader(x)} = \min(\Omega^*(v))$, and $Stab(leader(x))$. Then, only $leader(x)$ is activable (for the **Merge follow** rule) among the nodes of $\Omega_{leader(x)}$. As x and y are in $\Omega_{leader(x)} = \Omega^*(x)$, one of them must be $leader(x)$, and the other one is not activated in the transition. Then, if y performs **Merge follow** in the transition, we have $\Omega_y = \Omega^*(v) = \Omega^*(x)$ in γ'. Else, x performs **Merge follow** in the transition, and $\Omega_y \subseteq \Omega^*(v) = \Omega_x = \Omega^*(x)$ in γ'.

- We have $\beta_x \neq \bot \Rightarrow (Stab(x) \wedge leader(x) = x \wedge \beta_x \in \mathcal{N}_x \setminus \Omega_x)$ in γ. If $\beta_x^{\gamma'} = \bot$ there is nothing to prove. Let's then focus on the other case: $\beta_x \neq \bot$ in γ'. In this case we have $\beta_x \neq \bot$ in γ too as no rule that could give a non-$\bot$ β-value is enabled on $\Omega^*(v)$ in γ. Thus, by well-definedness, $Stab(x)$ and $leader(x) = x$ in γ and only x may be activable among nodes of Ω_x in γ, for the rule **Merge follow**. As $\beta_x \neq \bot$, it executes **Merge follow** in the transition, and no node of Ω_x was activated. Thus $Stab(x)$ and $leader(x) = x$ are still true in γ', and as no variable of x changes value, $\beta_x \in \mathcal{N}_x \setminus \Omega_x$ in γ' too.
- By well-definedness, $\min(\Omega^*(v)) \in \{leader(leader(x)), \beta_{leader(x)}\}$ in γ. As $\Omega^*(v)$ does not change in the transition, so does its minimum $\min(\Omega^*(v))$. As Ω-values may only grow in the transition, if $leader(leader(x)) = \min(\Omega^*(v))$ in γ, the same is also true in γ'. Suppose now that $leader(leader(x)) \neq \min(\Omega^*(v))$ in γ, which means that $\beta_{leader(x)} = \min(\Omega^*(v))$ in γ. Using well-definedness we have $Stab(leader(x))$ and $leader(leader(x)) = leader(x) \neq \min(\Omega^*(v))$. If $leader(x)$ is not activated in the transition there is nothing to prove. Else, it performs **Merge follow** and we have $\Omega^{\gamma'}_{leader(x)} = \Omega^{\gamma}_{\min(\Omega^*(v))} = \Omega^*(v)$. Thus, in γ', $leader(leader(x)) = \min(\Omega^*(v))$.

Thus, when $\neg Stab(u)$ in γ, u is well-defined in γ', which proves Point 1. □

Now suppose $Stab(u)$ in γ. As u is well-defined and there is no Byzantine node in Ω_u by hypothesis, no rule is enabled on $\Omega_u \setminus \{u\}$.

Proof of Point 2. If u does not execute any rule in the transition, no node of $\Omega_u = \Omega^*(u)$ does, and u is well-defined in γ' which proves Point 2. □

Suppose then that u executes a rule in the transition.

Proof of Point 3. Suppose $\beta_u^{\gamma} = \bot$. As u is well-defined, the only rules it can perform in the transition are **Seduction** and **Mariage**. From the guards of

those rules $\beta^{\gamma'}_{leader(u)}$ is a node in $merge\_candidate(leader(u))$ in γ, which ensures that $\beta^{\gamma'}_{leader(u)} \notin \Omega^{\gamma}_{leader(u)} = \Omega^{\gamma'}_{leader(u)}$. The guards of both rules imply that $Stab(u)$ is true in γ. This gives by well-definedness that no node of Ω^{γ}_{u} had β-value non-$\perp$ in γ, and that in Ω^{γ}_{u} only u is activable in γ. As no Ω-value in Ω^{γ}_{u} changes in the transition, we have that u is well-defined in γ'.

□

Suppose now that $\beta^{\gamma}_{u} \neq \perp$, u must have performed either **Abandonment**, **Merge lead**, or **Merge follow**.

Proof of Point 4. Suppose u performed **Abandonment** in the transition. As only u was activable in γ among nodes of Ω^{γ}_{u}, it's easy to see that u is still well-defined in γ' which proves Point 4. □

Suppose now that u did not perform **Abandonment** in the transition. It means that either **Merge follow** or **Merge lead** has been executed by u.

Proof of Point 5. Let's write $v = \beta^{\gamma}_{u}$, and suppose Ω_v does not contain Byzantine nodes. Then v is well-defined in γ, as $v \in merge\_candidate(u)$ (from the guard of both possible rules) and γ is $\mathcal{N}$-stabilized (by hypothesis).

- Suppose **Merge follow** was executed by u in the transition. Then we may apply Point 1 of the Lemma to v. Hence v is well-defined in γ'. From the guard of **Merge follow** we have $u \in \Omega^{\gamma}_{v}$, thus $u \in \Omega^{\gamma'}_{v}$ as v does not execute **Reset** in the transition. Then, applying Lemma 5.3.4, we get that u is well-defined.
- Suppose **Merge lead** was executed by u in the transition. We know from the guard of the rule that $v = \beta^{\gamma}_{u}$ is in $merge\_candidate(u)$ in γ, which implies $leader(v) = v$, $Stab(v)$, and $coherent\_neighborhood(v)$. Since the configuration is supposed $\mathcal{N}$-stabilized, v is then well-defined in γ. By hypothesis on u that there is no Byzantine node in Ω^{γ}_{v}. Since $Stab(v)$ is true, no node of $\Omega^{\gamma}_{v} \setminus \{v\}$ is activable in γ. Moreover, the guard of **Merge lead** also implies that $leader(u) < v$ and $\beta^{\gamma}_{v} = u$, which implies that v is not activable in γ. Thus u is the only node of $\Omega^{\gamma}_{leader(u)} \cup \Omega^{\gamma}_{v}$ that has been activated in the transition, and no other node of that set had the values of its variables changed. As $\Omega^{\gamma'}_{u} = \Omega^{\gamma}_{u} \cup \Omega^{\gamma}_{v}$ from **Merge lead** command, we get $\Omega^{*}(u)^{\gamma'} = \Omega^{\gamma'}_{u} = \Omega^{\gamma}_{u} \cup \Omega^{\gamma}_{v}$. Let's check that $coherent\_clique(u)$ is true in γ' (again we cut the proof according to the structure of the predicate to make the reading easier):
  - As only u was executed in the transition, we have $\Omega^{*}(u)^{\gamma'} = \Omega^{\gamma'}_{u} = \Omega^{\gamma}_{u} \cup \Omega^{\gamma}_{v}$. From the guard of **Merge lead**, $v \in merge\_candidate(u)$ in γ, hence $\Omega^{*}(u)^{\gamma'} = \Omega^{\gamma}_{u} \cup \Omega^{\gamma}_{v} \subseteq \mathcal{N}_u$

- As $\beta_u = v$ and $\beta_v = u$ in γ from the guard of **Merge lead**. By hypothesis u and v are well-defined in γ, thus u and v are the only nodes in $\Omega_u^\gamma \cup \Omega_v^\gamma$ to have non-$\perp$ β-value. Hence we have, in γ', $\{x \in \Omega^*(u) | \beta_x \neq \perp\} = \{v\}$, of size 1.
- Let $x \in \Omega^*(u)^{\gamma'}$,
  * $\Omega^*(u) \subseteq \mathcal{N}_x$ in γ' is a consequence of $v \in merge\_candidate(u)$ in γ.
  * As $Stab(u)$ and $Stab(v)$ in γ, we know that $leader(x)^\gamma$ is either u or v. If $x = u$ there is nothing to prove. Else x was not activated in the transition, and thus $\{x\} \subseteq \Omega_x \subseteq \Omega_{leader(x)}$ as the Ω-value of its leader may only have grown. Moreover, we still have $\Omega_v^{\gamma'} = \Omega^*(v)^{\gamma'}$ as no node of $\Omega_v^{\gamma'}$ is activated in the transition, and we've already seen that $\Omega_u^{\gamma'} = \Omega^*(u)^{\gamma'}$, thus $\Omega_{leader(x)} = \Omega^*(x)$.
  * Let y be a node of $\Omega_x^{\gamma'}$. If $x \in \Omega_u^\gamma$, either $y = u$, and then $\Omega_y^{\gamma'} = \Omega^*(u)^{\gamma'} = \Omega^*(x)^{\gamma'}$, or $\Omega_y^{\gamma'} = \Omega_u^\gamma \subseteq \Omega_x^{\gamma'}$. Else, $x \in \Omega_v^\gamma$, and $\Omega_x^{\gamma'} = \Omega_y^{\gamma'}$ as neither node was activated in the transition.
  * If $x \neq v$, $\beta_x^{\gamma'} = \perp$ and there is nothing to prove. Else, $x = v$, and as no node of Ω_v^γ is activated in the transition we still have $Stab(v), leader(v) = v$ and $\beta_v^\gamma \in \mathcal{N}_v \setminus \Omega_v$ in γ' .
  * If $x \in \Omega_u^\gamma$, we have $leader(x)^{\gamma'} = u$, thus $leader(leader(x)) = leader(u) = u$ in γ'. Else, $x \in \Omega_v^\gamma$, and then $leader(x)^{\gamma'} = v$ and thus $\beta_{leader(x)} = \beta_v = u$ in γ'.

Thus $coherent\_clique(v)$ is true in γ', and hence v is well-defined. Which proves Point 5. □

All five points have been proved, hence the result. □

Lemma 5.3.8. *Let γ be a $\mathcal{N}$-stabilized configuration, and $\gamma \xrightarrow{t} \gamma'$ a transition. Then $u \in V_1''(\gamma) \Rightarrow u \in V_1''(\gamma')$.*

Proof. As $u \in V_1''(\gamma)$, u is well-defined in γ from Lemma 5.3.6. By definition of V_1'', there is $x \in V_1$ well-defined in γ such that $u \in \Omega^*(x)$.

Let's write $u' = leader(leader(u))$. As $u' = leader(leader(u)) \in \Omega_{leader(u)}$ and $leader(u) \in \Omega_u$, we have $u' \in \Omega^*(x)$. Thus by definition of V_1'', $u' \in V_1''(\gamma)$ and u' is well-defined in γ from Lemma 5.3.6. As $\Omega_{leader(u)} = \Omega^*(u)^\gamma$ from well-definedness, we get $u' = leader(leader(u)) = \min \Omega^*(u)^\gamma$, and thus $leader(u') = u'$ in γ.

If $\neg Stab(u')$, u' does not perform a move in the transition, $\beta = \perp$, or u' executes **Abandonment**, we can apply Lemma 5.3.7 and thus u' is well-defined in γ'.

Suppose now that $Stab(u)$, $\beta^{\gamma}_{u'} \neq \bot$, and u performs a non-**Abandonment** move in the transition. By well-definedness we know that $\Omega^{\gamma}_{u'} = \Omega^*(u')^{\gamma}$

Let's write $v = \beta^{\gamma}_{u'}$. Given the conditions, the rule executed by u' in the transition is either **Merge lead** or **Merge follow**.

- Suppose it is **Merge follow**. As $u' \in \Omega^{\gamma}_v$ from the guard of **Merge follow**, $x \in \Omega^*(v)^{\gamma}$ by definition of Ω^*. Since we have $coherent\_clique(v)$ and $leader(v) = v$ in γ, we get $\Omega^{\gamma}_v = \Omega^*(v)^{\gamma}$. Thus, using again the fact that $coherent\_clique(v)$ is true in γ, we get $\Omega^{\gamma}_v = \Omega^*(v)^{\gamma} \subseteq \mathcal{N}_x$.

- Suppose now that it is **Merge lead**. The guard of the rule implies $v \in merge\_candidate(u')$, which leads to $\Omega_v \subseteq \mathcal{N}_x$.

In both cases, we have $\Omega_v \subseteq \mathcal{N}_x$. As $x \in V_1$ and the configuration is supposed $\mathcal{N}$-stabilized, it implies $\Omega^{\gamma}_v \subseteq V_0$ *i.e.* does not contain Byzantine nodes. We can then apply Lemma 5.3.7 and thus u' is well-defined in γ'.

Well-definedness of u' in γ gives $x \in \Omega^*(u')^{\gamma}$. As every node of $\Omega^*(x)^{\gamma}$ is well-defined by Lemma 5.3.6, no such node may have executed **Reset** in the transition, thus their Ω-value can only grow in the transition. Thus we have $x \in \Omega^*(u')^{\gamma'}$ which implies that x is well-defined in γ' from Lemma 5.3.4, and $u \in \Omega^*(x)$ which then gives $u \in V_1''(\gamma')$. □

There is still to prove that our V_1'' will contain V_1 at some point in the execution, as we advertised that the algorithm would converge on "at least" V_1.

Lemma 5.3.9. *Let γ be a $\mathcal{N}$-stabilized configuration. Every node of V_1 is well-defined in the configuration γ' reached after one round from γ.*

Proof. Suppose there are some nodes of V_1 that are not well-defined in γ. **Reset** is activable on them until they become well-defined or execute **Reset**, in which case they are well-defined in the configuration following this execution. By definition of a round one of them must happen before the round ends. Then from Lemma 5.3.8, those may not stop being well-defined, thus they are well-defined in γ'. □

5.3.3 . Any merging process ends

On $\mathcal{N}$-stabilized configurations, the algorithm behaves on a large time scale as if the only existing nodes were the ones that are currently their own leaders with their neighborhood modified to be the intersection of the neighborhood of every node under their rule. Once such a node has performed a move to merge with another leader (using the **Merge lead** rule), their subjects simply follow. Here, we prove that when a merging has begun, a leader must wait for its followers before

doing anything more, and that followers end up synchronized with their leader at some point.

To do this, we first prove that in the clique of such a leader node, the only rules that can be enabled are those that are designed to synchronize a follower to its leader, either **Update** (for those that were already followers before), or **Merge follow** (for the leader that is to become a follower after the merging is completed).

Lemma 5.3.10. *Let γ be a $\mathcal{N}$-stabilized configuration. If $u \in V_1''$, $leader(u) = u$, and $\neg Stab(u)$, only rules **Update** and **Merge follow** may be enabled on nodes of $\Omega^*(u)$ in γ.*

Proof. From Lemma 5.3.4 every node of $\Omega^*(u)$ is also well-defined. Consider then $v \in \Omega^*_u$.

- If $\neg Stab(v)$, only **Update** can be enabled on v.
- Else, we have $Stab(v)$. If $leader(v) \neq v$, v is not activable, and there is nothing to prove. Else, from well-definedness of u, we get $\min(\Omega^*(u)) = u \in \{leader(leader(v)), \beta_{leader(v)}\}$. As $leader(leader(v)) = v$ by hypothesis, we must have $\beta_v = u$, and then the only rule that can be enabled on v is **Merge follow**.

□

When the said **Update** or **Merge follow** are executed, we can observe that it leads to synchronization with the leader.

Lemma 5.3.11. *Let γ be a $\mathcal{N}$-stabilized configuration. Consider $u \in V_1''$ such that $leader(u) = u$ and $\neg Stab(u)$ in γ, and a transition $\gamma \xrightarrow{t} \gamma'$. We have:*

- $\Omega^*(u)^\gamma = \Omega^*(u)^{\gamma'}$;
- *If $v \in \Omega^*(u)^\gamma$ appears in a move of t, $\beta_v^{\gamma'} = \perp$ and $\Omega_v^{\gamma'} = \Omega^*(u)^\gamma$.*

Proof. Let's prove separately the two points of the lemma:

- Lemma 5.3.10 only allows **Update** and **Merge follow** to be enabled on nodes of $\Omega^*(u)^\gamma$ in γ. By well-definedness $\Omega^*(u)^\gamma = \Omega_u^\gamma$, and $\beta_u^\gamma = \perp$ thus no rule is enabled on u. Then **Update** and **Merge follow** may only change the Ω-value of nodes in Ω_u^γ to values that are included in Ω_u^γ in the transition. Thus $\Omega^*(u)^{\gamma'} = \Omega_u^\gamma = \Omega^*(u)^\gamma$.
- If $v \in \Omega^*(u)^\gamma$ appears in a move t, it means that it executed either **Update** and **Merge follow** in the transition. In both cases, the well-definedness of u implies that the new Ω-value of v must be $\Omega^*(u)^\gamma$.

□

To bound the time it takes for every node to synchronize with the new leader, we begin by removing **Merge follow** from the equation, which takes at most one round.

Lemma 5.3.12. *Let γ be a $\mathcal{N}$-stabilized configuration. Consider $u \in V_1''$ such that $leader(u) = u$, and $\neg Stab(u)$ in γ. Then after at most one round a configuration γ' is reached such that $\Omega^*(u)^{\gamma'} = \Omega^*(u)^{\gamma}$ and $\forall x \in \Omega^*(u)^{\gamma'}, \beta_x^{\gamma'} = \bot$.*

Proof. Using well-definedness, we know that at most one node of $\Omega^*(u)$ with β-value non-$\bot$ in γ. Remember that from Lemma 5.3.8 u will be well-defined in every future configuration.

If there is no such node, then γ already verifies the condition, and there is nothing to prove.

Suppose then such a node v exists, well-definedness gives that $\beta_v^{\gamma} = u$ and guarantees that **Merge follow** is enabled on it until this β-value changes. Thus, after at most one round, **Merge follow** is executed by this node. Consider the first transition when it happens $\gamma'' \to \gamma'$.

By hypothesis v does not execute **Merge follow** in any transition in $\gamma \to^* \gamma''$. Thus $\beta_v = u$ in every configuration between γ and γ''.

By immediate induction using our Lemma 5.3.11 and the fact that $\beta_v = u$, every configuration in $\gamma \to^* \gamma''$ is such that $leader(u) = u$ and $\neg Stab(u)$. We then have $\Omega^*(u)^{\gamma''} = \Omega^*(u)^{\gamma}$, and v is the only node of $\Omega^*(\gamma'')$ with β-value non-$\bot$.

Then using Lemma 5.3.11 we get in γ' that $\Omega^*(u)^{\gamma'} = \Omega^*(u)^{\gamma''}$ and since v is activated in the transition we have $\beta_v^{\gamma'} = \bot$. As the other nodes of $\Omega^*(u)^{\gamma'}$ had already β-value $\bot$ and no rule may have changed that in the transition by Lemma 5.3.10, we have $\forall x \in \Omega^*(u)^{\gamma'}, \beta_x^{\gamma'} = \bot$. □

Lemma 5.3.13. *Let γ be a $\mathcal{N}$-stabilized configuration. Consider $u \in V_1''$ such that $leader(u) = u$ and $\neg Stab(u)$ and $\forall x \in \Omega^*(u), \beta_x = \bot$ in γ.*

*Then $\forall x \in \Omega^*(u)$ either $\Omega_x = \Omega^*(u)$ or **Update** is enabled on x in γ.*

Proof. From Lemma 5.3.10 only **Update** and **Merge follow**. As the guard of **Merge follow** requires a non-$\bot$ β-value it cannot be enabled.

Consider $v \in \Omega^*(u)^{\gamma}$. The well-definedness of u gives $\min(\Omega^*(u)) \in \{leader(leader(v)), \beta_{leader(v)}\}$. As $u = \min(\Omega^*(v))$ and with the constraints on β-values, we get that $leader(leader(v)) = u$, thus $\Omega^*(v) = \Omega^*(u)$. Well-definedness gives also $\Omega_{leader(v)} = \Omega^*(u)$.

Then, either $\Omega_v = \Omega_{leader(v)} = \Omega^*(u)$ (and **Update** is not enabled on v), or $\Omega_v \neq \Omega_{leader(v)}$ and **Update** is enabled on v. □

Then, we prove that if only **Update** is enabled on the followers of a leader, after at most one round, a configuration where every follower is synchronized with the leader is reached.

Lemma 5.3.14. *Let γ be a $\mathcal{N}$-stabilized configuration. Consider $u \in V_1''$ such that $leader(u) = u$, $\neg Stab(u)$, and $\forall x \in \Omega^*(u)^\gamma, \beta_x^\gamma = \bot$.Then after at most one round a configuration γ' is reached such that $\Omega^*(u)^{\gamma'} = \Omega^*(u)^\gamma$, $Stab(u)$ is true and $\beta_u = \bot$ in γ'.*

Proof. Consider such a configuration γ and such a node u. Let v be a node such that $\Omega_v \neq \Omega^*(u)$.

Let's prove by induction that in every configuration until v executes **Update**, $\Omega^*(u)$ does not change, and $\neg Stab(u)$ is true. It is enough for that purpose to see that $\Omega_v \neq \Omega^*(u)$ prevents $Stab(u)$ to be true, thus the induction works using Lemma 5.3.11.

Consider then $S = \{x \in \Omega^*(u)^\gamma | \Omega_v^\gamma \neq \Omega^*(u)^\gamma\}$. As every node in S will be activable until it has executed **Update**, after at most one round every one of them will have done this. Consider then the configuration just after the last of them is activated for the first time since γ, γ'.

By the above argument, every configuration in $\gamma \rightarrow^* \gamma'$ before γ' is such that $\neg Stab(u)$ and $\Omega^*(u)$ is the same as in γ, and then using Lemma 5.3.11 for the last transition we have $\Omega^*(u)^{\gamma'} = \Omega^*(u)^\gamma$. Then as Ω-value may only grow on well-defined nodes, in γ', we must have $Stab(u)$. Morever as in every configuration in $\gamma \rightarrow^* \gamma'$ before γ' we have $\neg Stab(u)$, no rule may have been executed to change a β-value in $\Omega^*(u)$, thus $\beta_u^{\gamma'} = \bot$. □

As a summary, it takes at most 2 rounds for every follower to synchronize with the leader.

Lemma 5.3.15. *Let γ be a $\mathcal{N}$-stabilized configuration. Consider $u \in V_1''$ such that $\neg Stab(u)$ in γ. Then after at most two rounds a configuration γ' is reached such that $\Omega^*(u)^{\gamma'} = \Omega^*(u)^\gamma$ and $Stab(u)$ in γ'.*

Proof. Using Lemma 5.3.12, from γ, after at most 1 round, a configuration γ'' is reached and is such that $\Omega^*(u)^{\gamma''} = \Omega^*(u)^\gamma$ and $\forall x \in \Omega^*(u), \beta_x^{\gamma''} = \bot$.

Using Lemma 5.3.8 we know that u is still well-defined in γ'', which gives us that $\Omega_u^{\gamma''} = \Omega^*(u)^{\gamma''} = \Omega^*(u)^\gamma = \Omega_u^\gamma$. Thus $leader(u)^{\gamma''} = leader(u)^\gamma = u$.

If $Stab(u)^{\gamma''}$, we can take $\gamma' = \gamma''$, and there is nothing left to prove. Suppose now that $\neg Stab(u)^{\gamma''}$. Then, using Lemma 5.3.14, from γ'', after at most 1 round, a configuration γ' is reached and is such that $\Omega^*(u)^{\gamma'} = \Omega^*(u)^{\gamma''}$ and $Stab(u)$ is true in γ'. Which concludes the proof, as $\Omega^*(u)^{\gamma''} = \Omega^*(u)^\gamma$. □

5.3.4 . Merging happens and makes the solution progress

Now that we know that once a merging has begun it ends in a small number of rounds, we want to be sure that some merging happens.

Definition 5.3.16. Consider a configuration γ, and $K = \{\Omega_x | x \in V_1''(\gamma)\})$ the set of all Ω-values in $V_1''(\gamma)$.

We define: $\mathcal{C}(\gamma) = \{\omega \in K | \omega \text{ is maximal for inclusion in } K\}$

To begin with, we prove that it is indeed a clique decomposition.

Lemma 5.3.17. *Let γ be a $\mathcal{N}$-stabilized configuration, then $\mathcal{C}(\gamma)$ is a clique decomposition of V_1''.*

Proof. Being a clique decomposition of V_1'' is to be a set of cliques, to cover the entire set, and to have pairwise disjoint members:

- Since every node of V_1'' is well-defined by Lemma 5.3.6, and since γ is $\mathcal{N}$-stabilized, every Ω-value of nodes in V_1'' is a clique of G.
- From well-definedness we have that every node is contained in its Ω-value thus $\mathcal{C}(\gamma)$ is a cover of V_1''.
- Suppose by contradiction that c and c' distinct elements of $\mathcal{C}(\gamma)$ are such that $c \cap c' \neq \emptyset$. Consider then $u \in c \cap c'$.

  Consider v and v' in V_1'' such that $\Omega_v = c$ and $\Omega_{v'} = c'$ which exist by definition of $\mathcal{C}(\gamma)$. They are distinct since $\Omega_v = c \neq c' = \Omega_{v'}$. If $v = u$ then by well-definedness we have $c \subseteq c'$, which is a contradiction to c being a member of $\mathcal{C}(\gamma)$. Symmetrically the same can be said if $v' = u$. Suppose then that neither v nor v' is equal to u. Well-definedness of v and v' implies that they are both in $\{leader(leader(u)), \beta_u\}$. If $\beta_u = \perp$, this is impossible as $\min(c)$ and $\min(c')$ are distinct. Else $\beta_u \neq \perp$, and well-definedness of u implies that $leader(leader(u)) = u$, which is impossible as u, v and v' are supposed distinct. Thus, by contradiction, we have that $c \cap c' = \emptyset$.

Thus $\mathcal{C}(\gamma)$ is a clique decomposition of V_1''. □

The next lemma is about some well-formed property of the cliques of $\mathcal{C}(\gamma)$: every such clique must have a leader, and every node in a clique must have an Ω-value included in the clique. It is what we expect from the way the algorithm forms new cliques by merging.

Lemma 5.3.18. *Let γ be a $\mathcal{N}$-stabilized configuration, and consider $c \in \mathcal{C}(\gamma)$. We have $\forall x \in c, \Omega_x \subseteq c$. Moreover, $\exists! u \in V_1'', leader(u) = u \wedge \Omega_u = c$.*

Proof. Consider $x \in c$. Consider also $v \in V_1''$ such that $\Omega_v = c$, which exists by definition of $\mathcal{C}(\gamma)$. By well-definedness of v we have that $\Omega_x \subseteq \Omega_v = c$.

Consider then $u = \min(c)$. We have by definition of $leader$ that $leader(v) = u$, and then by well-definedness $c \subseteq \Omega_u$. As $u \in c$, we also have $\Omega_u \subseteq c$, thus $\Omega_u = c$.

Unicity comes from that the leader of such a node must have $\min(c)$ as a leader. □

Definition 5.3.19 (Representative)**.** Since we know the existence and unicity in every clique of $\mathcal{C}(\gamma)$ of a node that is its own leader and has the clique as Ω-value, we call $min(c)$ the *representative* of the clique $c \in \mathcal{C}(\gamma)$ in configuration γ.

To progress toward our goal, we need to prove that $\mathcal{C}$ makes progress in some sense. To this aim, we prove a heredity property as well as a non-regression property across transitions for $\mathcal{C}$.

Lemma 5.3.20. *Let γ be a $\mathcal{N}$-stabilized configuration and $\gamma \to \gamma'$ a transition. We have:*

- *(Non-regression)* $\forall c \in \mathcal{C}(\gamma)$, $\exists c' \in \mathcal{C}(\gamma')$, $c \subseteq c'$.
- *(Heredity)* $\forall c' \in \mathcal{C}(\gamma')$, $\exists c \in \mathcal{C}(\gamma)$, $c \subseteq c'$.

Proof. Let's first prove the first point of the lemma. Consider $c \in \mathcal{C}(\gamma)$, and consider u the representative of c in γ (see Lemma 5.3.18). Consider also $c' \in \mathcal{C}(\gamma')$ such that $u \in c'$ (which exists by Lemma 5.3.17), with v the representative of c' in γ' (see Lemma 5.3.18).

As $u \in \Omega_v^{\gamma'}$, by well-definedness $\Omega_u^{\gamma'} \subseteq \Omega_v^{\gamma'}$. But since $u \in V_1''(\gamma)$, we also have $\Omega_u^{\gamma} \subseteq \Omega_u^{\gamma'}$ (only **Reset** may make the Ω-value shrink). Thus $\Omega_u^{\gamma} \subseteq \Omega_v^{\gamma'}$ *i.e.* $c \subseteq c'$. Hence the first point of the Lemma.

Let's then prove the second point of the lemma. Consider $c' \in \mathcal{C}(\gamma')$ with v the representative of c in γ (see Lemma 5.3.18). If $c' \in \mathcal{C}(\gamma)$, there is nothing to prove, suppose then w.l.o.g. that it's not the case.

As the Ω-value of v must have changed in the transition, it executed either **Merge lead**, **Merge follow**, or **Update**.

- In fact, **Update** is not a valid option as it is not enabled on u in γ. It would otherwise imply the existence of $u \in V_1''(\gamma)$ such that $\Omega_u = c'$. Then there would be $c \in \mathcal{C}(\gamma)$ such that $\Omega_u \subseteq c$ by definition of $\mathcal{C}$. Then with the first point of the Lemma plus the fact that $\mathcal{C}(\gamma')$ is a clique decomposition by Lemma 5.3.17 we get $c' \subseteq c \subseteq c'$ *i.e.* $c = c'$ which is false by hypothesis.

- If v performed **Merge lead** in the transition, let's write $u = \beta_v^\gamma$. We have also from the guard of **Merge lead** that $\beta_u^\gamma = v$, $\Omega_u^\gamma \cap \Omega_v^\gamma = \emptyset$, $Stab(v)^\gamma$, and u is not enabled in γ. By the definition of $V_1''(\gamma')$ and the fact that $Stab(v)$ is true, $\Omega_v^{\gamma'}$ must contain a node of V_1. Then, as $\Omega_v^{\gamma'} = \Omega_u^\gamma \cup \Omega_v^\gamma$, either Ω_u^γ or Ω_v^γ.
  - Suppose Ω_u^γ contains a node of V_1, we have $u \in V_1''$. Suppose by contradiction that $\Omega_u^\gamma \notin \mathcal{C}(\gamma)$. There must exist $u' \in V_1''(\gamma)$ such that $\Omega_u^\gamma \subseteq \Omega_{u'}^\gamma$. $u' \notin \Omega^*(v)^\gamma$ from the fact that $\Omega_u^\gamma \cap \Omega_v^\gamma = \emptyset$. But then $\beta_u = v$ is a contradiction to the well-definedness of u'. Thus, by contradiction we have $\Omega_u^\gamma \in \mathcal{C}(\gamma)$, we then write $c = \Omega_u^\gamma$. After the transition we have $\Omega_v^{\gamma'} = \Omega_u^\gamma \cup \Omega_v^\gamma$, thus $c \subseteq c'$.
  - Suppose now Ω_v^γ contains a node of V_1. By the same reasoning as the previous case, we have that $\Omega_v^\gamma \in \mathcal{C}(\gamma)$, and then by taking $c = \Omega_v^\gamma$ we have $c \subseteq c'$.
- Else, v must have performed **Merge follow** in the transition, and let's write $u = \beta_v^\gamma$. The guard of the rule also gives us $Stab(v)^\gamma$, $\Omega_u^\gamma = c'$. Observe that as we supposed that $c' \notin \mathcal{C}(\gamma)$ this implies that $u \notin V_1''$, and that $v \in V_1''(\gamma)$. By the same reasoning than for the **Merge lead** case we get $\Omega_v^\gamma \in \mathcal{C}(\gamma)$, and then by taking $c = \Omega_v^\gamma$ we have $c \subseteq c'$.

□

Remark. *Lemma 5.3.20 implies that in such a transition* $|\mathcal{C}(\gamma')| \leq |\mathcal{C}(\gamma)|$.

When $\mathcal{C}$ changes, it makes progress. But we have to ensure that it does change sometimes. The next four lemmas are a toolbox that will be used to prove that.

Before proving that $\mathcal{C}$ makes progress, we prove a lemma about what happens when a node of $V_0 \setminus V_1''$ is merged with one from V_1''.

Lemma 5.3.21. *Let γ be $\mathcal{N}$-stabilized configuration and v a node of $V_1''(\gamma)$. If v executes rule **Merge lead** or **Merge follow** in the transition $\gamma \to \gamma'$, then $\beta_v^\gamma \in V_1''(\gamma')$.*

Proof. Suppose v executes **Merge lead** in the transition $\gamma \to \gamma'$, it means that **Merge lead** was enabled on v in γ. Then, as $\beta_v \in merge\_candidate(v)$ in γ, we have $leader(\beta_v) = \beta_v$ in γ, and thus $\beta_v \in \Omega_{\beta_v}$ in γ. Thus, we have $\beta_v^\gamma \in \Omega_v^{\gamma'} = \Omega_v^\gamma \cup \Omega_{\beta_v}^\gamma$.

Suppose now that v executes **Merge follow** in the transition $\gamma \to \gamma'$, it means that **Merge follow** was enabled on v in γ. Then, as $coherent\_clique(\beta_v)$ in γ, we have $\beta_v \in \Omega_{\beta_v}$ in γ. Thus, we have $\beta_v^\gamma \in \Omega_v^{\gamma'} = \Omega_{\beta_v}^\gamma$.

In both cases, $\beta_v^\gamma \in \Omega_v^{\gamma'}$. But as $u \in V_1''(\gamma)$, from Lemma 5.3.8 we have $u \in V_1''(\gamma')$. Thus $\beta_v^\gamma \in V_1''(\gamma')$. □

Starting in a $\mathcal{N}$-stabilized configuration, either we directly get what we want (Case 4 of Lemma 5.3.22), or we reach one of two types of configurations (Cases 1,2 and 3 of Lemma 5.3.22) that will be dealt with in other lemmas.

Lemma 5.3.22. *Let γ be a $\mathcal{N}$-stabilized configuration. Suppose $\mathcal{C}(\gamma)$ is not a minimal clique decomposition of $V_1''(\gamma)$.*

Then, after at most two rounds, a configuration γ' is reached where one of those is true:

1. ***Seduction** or **Mariage** is enabled on at least one node of $V_1''(\gamma')$.*
2. *$\exists u \in V_1'', \exists v \in V_0$ such that $\beta_u = v$ and **Mariage** is enabled on v.*
3. *$\exists u \in V_1'', \exists v \in V_0$ such that **Merge Lead** is enabled on u or v.*
4. *$|\mathcal{C}(\gamma')| < |\mathcal{C}(\gamma)|$ or $V_1''(\gamma) \subsetneq V_1''(\gamma')$*

Proof. Suppose there exists $c, c' \in \mathcal{C}(\gamma)$ such that $c \cup c'$ is a clique (they exist by hypothesis, as $\mathcal{C}(\gamma)$ would be a minimal clique decomposition of $V_1''(\gamma)$ otherwise).

Consider then u (resp. u') the representative of c (resp. c') in γ. If **Seduction** or **Mariage** is enabled on u there is nothing to prove. The same can be said if $\beta_u \neq \perp$ and **Merge Lead** is enabled on either u or β_u. The same can be said for u'.

Let's then suppose it's not the case in γ. We are in either of those cases :

- $Stab(u)$, $\beta_u^\gamma \neq \perp$, and **Merge follow** is enabled on u. We have $\beta_u^\gamma \notin V_1''$ as if it was not the case, u would not be a representative. Then the guard of **Merge follow** guarantees that $Stab(\beta_u)$ in γ and that this will not change until **Merge follow** is executed by u. Thus after at most one round, u executes **Merge follow**, and in the resulting configuration β_u^γ is in V_1'' from Lemma 5.3.21 and we have $V_1''(\gamma) \subsetneq V_1''(\gamma')$ (Case 4 of the lemma).
- $Stab(u)$, $\beta_u^\gamma \neq \perp$, and **Merge follow** is not enabled on u. By hypothesis, **Merge lead** is not enabled on u either.
  - If **Abandonment** is not enabled on u we have $\beta_v \in \{\perp, u\}$ and $v \in merge\_candidate(u)$. If $\beta_v = \perp$, **Mariage** is then enabled on v (Case 2 of the lemma), else $\beta_v = u$ and **Merge lead** is enabled on u or v (Case 3 of the lemma)
  - If **Abandonment** is enabled on u, it remains so until either it executes **Abandonment**, or **Abandonment** is not activable anymore, and one of those must happen in at least one round. In the first

case in the configuration just after the transition where it executes **Abandonment** we have $\beta_u = \perp$, and $Stab(u)$. In the second case, as in the previous point, we are in either Case 2 or 3 of the lemma.

- $\neg Stab(u)$ in γ, in which case by Lemma 5.3.15 a configuration where $Stab(u)$ is true and $\beta_u = \perp$ is reached after at most two rounds.

Thus, after at most two rounds, either a configuration that satisfies one of the conditions of the lemma has been reached, or a configuration where $Stab(u)$ and $\beta_u = \perp$ has been reached. Note that when this is the case, the only rules that may be enabled on u are **Seduction** or **Mariage**, so we may assume w.l.o.g. that in the configuration when exactly two rounds have passed $Stab(u)$ and $\beta_u = \perp$ (otherwise we reached a configuration corresponding to Case 1 of the lemma before that).

By symmetry, the exact same argument applies to u'.

Then, after two rounds, we are in a configuration where $Stab(u)$, $\beta_u = \perp$, $Stab(u')$ and $\beta_{u'} = \perp$. Then, as $c \cup c'$ is supposed to be a clique, **Seduction** is enabled on u and u', which concludes the proof. □

Then, starting in a configuration corresponding to Case 1 of Lemma 5.3.22, we reach with probability at least $\frac{1}{\Delta}$ a configuration having the same properties as the one of Case 3 of Lemma 5.3.22, *i.e.* a configuration where a clique merging is about to begin.

Lemma 5.3.23. *Let γ be a $\mathcal{N}$-stabilized configuration and suppose **Seduction** or **Mariage** is enabled on at least one node of V_1''. Then, there is a probability at least $\frac{1}{\Delta}$ that after at most two rounds a configuration γ' is reached where: $\exists u \in V_1'', \exists v \in V_0$ such that **Merge Lead** is enabled on u or v.*

Proof. Consider a node $u \in V_1''(\gamma)$ such that **Mariage** is enabled on it. Any node $w \in V_0$ such that $\beta_w^\gamma = u$ and $w \in merge\_candidate(u)$ cannot execute any rule while **Mariage** is enabled on u. There is at least one such node since **Mariage** is enabled on u. Thus after at most one round, u executes **Mariage**, and in the resulting configuration we have $\beta_u = v \in V_0$, $\beta_u = v$, and $\beta_v = u$, with $v \in merge\_candidate(u)$. Thus, in this configuration, **Merge Lead** is enabled on $\min(u, v)$.

Suppose now that **Mariage** is not enabled on any node of V_1''. Consider $u \in V_1''\gamma$ such that **Seduction** is enabled on it. **Seduction** will remain enabled on u until either it is executed, or one member of $merge\_candidate(u)$ executes **Seduction**.

- Suppose that in the first transition where one of those events happens u executes **Seduction** (after at most one round), and γ' the resulting

configuration. If $v = \beta_u^{\gamma'}$ did not execute any rule in the transition, then we have $v \in merge\_candidate(u)^{\gamma'}$, $\beta_v^{\gamma'} = \perp$, and thus **Mariage** is enabled on it. It will remain enabled until v executes **Mariage** (after at most one round), and in the resulting configuration with probability at least $\frac{1}{\Delta}$ in the resulting configuration γ'' we have $\beta_v^{\gamma''} = u$, and **Merge lead** is enabled on $\min(u, v)$ in γ''.

- Suppose now that in the first transition where one of those events happens (after at most one round) u does not execute **Seduction** and at least one node $w \in merge\_candidate(u)$ executes **Seduction**. In the resulting configuration γ' we have then $Stab(u)$, $\beta_u^{\gamma'} = \perp$. Moreover, with probability at least $\frac{1}{\Delta}$, $\beta_w = u$. Thus in γ' **Mariage** is now enabled on u until it is executed on u as no rule will be enabled on v until then. When this happens (after at most one round), in the resulting configuration γ'', $v = \beta_u^{\gamma}$ is such that $\beta_u = v$, $\beta_v = u$, $v \in merge\_candidate(u)$ and $u \in merge\_candidate(v)$. Thus, **Merge lead** is enabled on $\min(u, v)$ in γ''.

□

From a configuration of the type of Case 2 of Lemma 5.3.22 we will reach a configuration of the type of Case 4 of Lemma 5.3.22 with probability at least $\frac{1}{\Delta}$.

Lemma 5.3.24. *Let γ be a $\mathcal{N}$-stabilized configuration such that $\exists u \in V_1'', \exists v \in V_0$ such that $\beta_u = v$ and **Mariage** is enabled on v.*

*Then, there is a probability at least $\frac{1}{\Delta}$ that after at most one round a configuration γ' is reached where: $\exists u \in V_1'', \exists v \in V_0$ such that **Merge Lead** is enabled on u or v.*

Proof. As **Mariage** is enabled on v in γ, we have that v is well-defined and $v \in merge\_candidate(u)$, thus Ω_v does not have Byzantine nodes. **Mariage** will then remain enabled on v until it is executed as no node of Ω_u and Ω_v except is activable while **Mariage** is enabled on $\beta_u = v$. When the first time v executes **Mariage** starting in γ (which happens after at most one round), in the resulting configuration γ', there is a probability at least $\frac{1}{\Delta}$ (Δ being the maximum size of $merge\_candidate(v)$) that $\beta_v = u$. If this is the case, as no other node than v in Ω_u and Ω_v

Observe that, as no other node than v in Ω_u and Ω_v was activable before v executed **Mariage**, we still have $v \in merge\_candidate(u)$ and $u \in merge\_candidate(v)$ in γ'. Then, if $\beta_v^{\gamma'} = u$, **Merge lead** is enabled on either u or v in γ'.

Thus, with probability at least $\frac{1}{\Delta}$, **Merge lead** is enabled on either u or v in γ'. □

And then from a configuration of the type of Case 3 of Lemma 5.3.22 we will reach a configuration of the type of Case 4 of Lemma 5.3.22.

Lemma 5.3.25. *Let γ be a $\mathcal{N}$-stabilized configuration such that $\exists u \in V_1'', \exists v \in \{x \in V_0 | well\_defined(x)\}$ such that **Merge Lead** is enabled on u or v. Then after 2 rounds a configuration γ' is reached where $\beta_u = \beta_v = \perp$, and either is true:*

- $|\mathcal{C}(\gamma')| < |\mathcal{C}(\gamma)|$,
- $|\mathcal{C}(\gamma')| \leq |\mathcal{C}(\gamma)|$ *and* $V_1''(\gamma) \subsetneq V_1''(\gamma')$.

Proof. As $u \in V_1''(\gamma)$, by definition of V_1'' there is $w \in V_1$ well-defined in γ such that $u \in \Omega^*(w)^\gamma$. As w is well-defined in γ, we have $\min \Omega^*(w)^\gamma \in \{leader(leader(u)), \beta_u\}$ *i.e.* $\min \Omega^*(w)^\gamma \in \{u, v\}$. If $\min \Omega^*(w)^\gamma$ was v, it would contradict w well-definedness since we would have $\beta_v^\gamma \in \Omega^*(w)^\gamma$, thus by contradiction $\min \Omega^*(w)^\gamma = u$.

Moreover well-definedness of w also implies that $w \in \min \Omega^*(w)^\gamma$, thus $w \in \Omega_u$. From the guard of **Merge lead**, either $u \in merge\_candidate(v)$ or $v \in merge\_candidate(u)$. In both cases it implies that every node of Ω_v is neighbor of $w \in V_1$, thus $\Omega_v \subseteq V_0$.

Let us write $x = \min(u, v)$ and $y = \max(u, v)$

Given the constraints on the configuration, we know that:

- **Merge lead** (and only this rule) is enabled on x in γ,
- No node in $\Omega_x^\gamma \setminus \{x\}$ or $\Omega_y^\gamma \setminus \{y\}$ is activable as we supposed $Stab(x)$, $Stab(y)$, and that x and y are well-defined,
- No rule is activable on y, as $\beta_x = y$.

Those facts will remain true after any transition where x is not activated (as it means that no node in Ω_x^γ or Ω_y^γ made a move, and thus the hypothesis on x and y stay true in the resulting configuration).

Thus, in the first transition where a node of $\Omega_x^\gamma \cup \Omega_y^\gamma$ is activated, **Merge lead** is executed by x and no other node of $\Omega_x^\gamma \cup \Omega_y^\gamma$ is activated. It happens after at most 1 round since **Merge lead** is continuously enabled on x until so. Consider the configuration γ' just after the transition where it happens.

Lemma 5.3.7 allow us to say that both x and y are still well-defined in γ', as neither Ω_x^γ nor Ω_y^γ contain Byzantine nodes.

We still have $Stab(y)$ and $leader(y) = y$ in γ' since no node of Ω_y was activated. Moreover we also still have $leader(x) = x$ in γ' as $x < y = min\Omega_y^\gamma$, and we have $\Omega_y^{\gamma'} = \Omega_y^\gamma \subseteq \Omega_x^\gamma \cup \Omega_y^\gamma = \Omega_x^{\gamma'}$. Moreover, as x is well-defined in γ', $coherent\_clique(x)$ is true in γ'.

Thus **Merge follow** is enabled on y. Observe that no other node of Ω_y is activable, and that it won't change until y executes a rule. Node x cannot make any move until $Stab(x)$ becomes true again, thus not before **Merge follow** is executed by y. It happens after at most 1 round since **Merge follow** is continuously enabled on y until so. Consider the configuration γ'' just after the transition where it happens.

Again using Lemma 5.3.7, x and y are still well-defined in γ''. From the command of **Merge follow** we have $\beta_y^{\gamma''} = \perp$. Moreover, from the guard of **Merge lead**, $\beta_y^{\gamma'} = \perp$, and since x did not get activated afterward $\beta_x^{\gamma''} = \perp$.

Then, as $v \in \Omega_u^{\gamma''}$ and $u \in V_1''(\gamma'')$, we have $v \in V_1''(\gamma'')$.

- If u and v where both in $V_1''(\gamma)$, we have $\Omega_u^\gamma \in \mathcal{C}(\gamma)$ and $\Omega_v^\gamma \in \mathcal{C}(\gamma)$ with $\Omega_u^\gamma \neq \Omega_v^\gamma$. Moreover, we have $\Omega_u^\gamma \cup \Omega_v^\gamma = \Omega_u^{\gamma''} \in \mathcal{C}(\gamma'')$. Using Lemma 5.3.20 we can then say $|\mathcal{C}(\gamma'')| < |\mathcal{C}(\gamma)|$.
- Else, v was not in $V_1''(\gamma)$. Thus, as we have $v \in V_1''(\gamma'')$, $V_1''(\gamma) \subsetneq V_1''(\gamma'')$. Lemma 5.3.20 also implies $|\mathcal{C}(\gamma'')| \leq |\mathcal{C}(\gamma)|$.

□

5.3.5 . Convergence and time complexity

Then, as we visually represent in Figure 5.1, we have a probabilistic pattern that makes cliques grow that will repeat as long as $\mathcal{C}$ is not a minimal clique decomposition of V_1''. Trivially, it implies that the algorithm ends with probability 1, but we will try to be more precise than that.

To do this, we use a concentration inequality (Azuma's inequality) to give a probabilistic bound on the number of rounds it takes to reach a configuration where $\mathcal{C}$ is a minimal clique decomposition of V_1''.

Lemma 5.3.26. *Let γ be a $\mathcal{N}$-stabilized configuration. With probability at least p, after $4\max(-\Delta^2 \ln p, \frac{\sqrt{2}}{\sqrt{2}-1}\Delta n) + 6n$ rounds a configuration γ' such that $\mathcal{C}(\gamma')$ is a minimal clique decomposition of V_1'' is reached.*

Proof. The size of $\mathcal{C}$ may only decrease from Lemma 5.3.20, and V_1'' may only increase from Lemma 5.3.8.

Suppose that $\mathcal{C}(\gamma)$ is not a minimal clique decomposition of V_1''.

Starting in configuration γ, using Lemmas 5.3.22, 5.3.23 and 5.3.24 there is a probability at least $\frac{1}{\Delta}$ to reach a configuration corresponding to the pre-conditions of Lemma 5.3.25 in at most 4 rounds.

If it is successful, from Lemma 5.3.25, starting in the resulting configuration γ', after at most two rounds, a configuration γ'' is reached such that $|\mathcal{C}(\gamma'')| < |\mathcal{C}(\gamma)|$, or $|\mathcal{C}(\gamma'')| \leq |\mathcal{C}(\gamma)|$ and $V_1''(\gamma) \subsetneq V_1''(\gamma'')$.

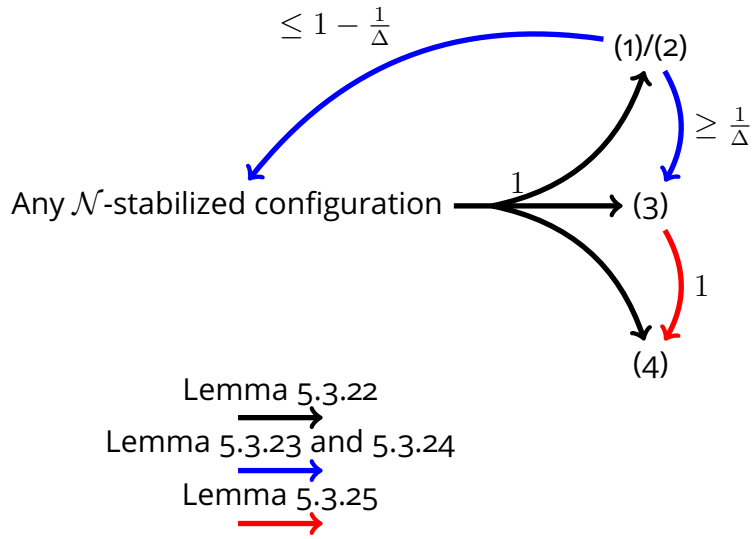


Figure 5.1: (1), (2), (3), and (4) refer to configurations having properties numbered in Lemma 5.3.22. Values on arrows are probabilities given by the lemmas.

In both cases (successful or not), either in the resulting configuration $\mathcal{C}$ is a clique decomposition of V_1'' (and there is nothing left to prove), or we may again apply the same set of lemmas.

Then, by Azuma's inequality, with probability at least p, in at most $\max(-\Delta^2 \ln p, \frac{\sqrt{2}}{\sqrt{2}-1}\Delta n)$ iterations, we get a configuration where $\mathcal{C}(\gamma')$ is a minimal clique decomposition of V_1''.

Successful iterations take at most 6 rounds, unsuccessful ones 4. Since there can be at most n successful iterations, we get to that configuration after at most $4\max(-\Delta^2 \ln p, \frac{\sqrt{2}}{\sqrt{2}-1}\Delta n) + 6n$ rounds. □

Follows the theorem as a direct corollary of Lemma 5.3.26 and 5.3.2

Theorem 5.3.27. *From any configuration γ, with probability at least p, after $4\max(-\Delta^2 \ln p, \frac{\sqrt{2}}{\sqrt{2}-1}\Delta n) + 6n + 1$ rounds a configuration γ' such that $\mathcal{C}(\gamma')$ is a minimal clique decomposition of V_1'' (and $V_1 \subseteq V_1''$) is reached .*

5.4 . Specification

As we introduced the notions $\mathcal{C}$ and V_1'' we can now express the specification of our algorithm: the legitimate configurations are configurations γ such that $V_1 \subseteq V_1''(\gamma)$ and $\mathcal{C}(\gamma)$ is a minimal clique decomposition of $V_1''(\gamma)$.

We cannot guarantee that when such a configuration is reached $\mathcal{C}$ and V_1'' will not change again. It's because nodes that neighbor Byzantine nodes, and were

previously entangled in dummy cliques staged by those Byzantine nodes, may at any time execute **Reset** after a move from a Byzantine node. If one of them merges with a clique of V_1'' after that, it makes V_1'' grow, and this can happen arbitrarily far in the execution. But what we can guarantee as a stability property is that when such a configuration is reached the property $\mathcal{C}$ is a minimal clique decomposition of V_1'' will be conserved, even if the values of $\mathcal{C}$ and V_1'' change. And recall that those may only change by "growing" in some sense (see Lemmas 5.3.20 and 5.3.8 respectively).

Lemma 5.4.1. *Let γ be a $\mathcal{N}$-stabilized configuration such that $\mathcal{C}(\gamma)$ is a minimal clique decomposition of $V_1''(\gamma)$. Suppose $\gamma \to \gamma'$.*

Then $\mathcal{C}(\gamma')$ is a minimal clique decomposition of $V_1''(\gamma')$.

Proof. Suppose by contradiction that $\mathcal{C}(\gamma')$ is not a minimal clique decomposition of $V_1''(\gamma')$.

As γ' is $\mathcal{N}$-stabilized by Lemma 5.3.3, Lemma 5.3.20 tells us that $\mathcal{C}(\gamma')$ is a clique decomposition of V_1''. Then $\mathcal{C}(\gamma')$ not being a minimal clique decomposition of $V_1''(\gamma')$ means that there exist c and c' in $\mathcal{C}(\gamma')$ such that $c \cup c'$ is a clique of $V_1''(\gamma')$.

But then, using Lemma 5.3.20, consider k, k' in $\mathcal{C}(\gamma)$ such that $k \subseteq c$ and $k' \subseteq c'$. Since $c \cup c'$ is a clique, it means that $k \cup k' \subseteq c \cup c'$ is also a clique. It is then a clique of $V_1''(\gamma)$, which contradicts the fact that $\mathcal{C}(\gamma)$ is a minimal clique decomposition of $V_1''(\gamma)$. □

5.5 . Correction

Moreover, we want a correction property for our algorithm. A natural thing would be to be in a legitimate configuration when no node is activable in V_1''. However, it's not the case, as a node of V_1'' might not be activable, but waiting for a node in $V_0 \setminus V_1''$ to execute **Merge lead** before being able to execute **Merge follow**. We will then restrict our correction property, expressed in Lemma 5.5.3 to configurations where β-values of every node in V_1'' is $\perp$. To prove this lemma, we will prove two preliminary lemmas.

First, a lemma that states some properties that are weaker than being a legitimate configuration but which are true whenever no node is activable in V_1''.

Lemma 5.5.1. *Let γ be a configuration with no node activable in V_1''. For $v \in V_1''$ we have in γ:*

1. $\mathcal{N}_v = N(v)$,
2. $\Omega_v = \Omega_{leader(v)}$,

3. $Stab(v)$,
4. Ω_v *is a clique of* G.

Proof. 1. If we had $\mathcal{N}_v = N(v)$, **Reset** would be activable on v.

2. Suppose $\Omega_v \neq \Omega_{leader(v)}$. Then either v is not well-defined and **Reset** is enabled on v, or **Update** is enabled on v.

3. Suppose $\neg Stab(v)$. There would be $w \in \Omega_v$ ($= \Omega_{leader(v)} \subseteq V_1''$) such that $\Omega_w \subsetneq \Omega_{leader(v)}$. Then either w is not well-defined, or **Update** is enabled on w, which contradicts the non-activable hypothesis.

4. Suppose that Ω_v is not a clique of G, *i.e.* $\exists s, t \in \Omega_v$ distinct such that s and t are not neighbors. By definition, s and t are in V_1'. From previous points, we have $Stab(v)$, thus $\Omega_v = \Omega_s = \Omega_t$. Then $t \notin N(s)$ implies either that $t \notin \mathcal{N}_s$ or $N(s) \neq \mathcal{N}_s$, and in both cases **Reset** is enabled on s which contradicts the hypothesis that no node is activable in V_1''.

□

Then we prove that in a configuration where no node is activable in V_1'', if two cliques of V_1'' are not already waiting to merge with some node in $V_0 \setminus V_1''$, then their union is not a clique.

Lemma 5.5.2. *Let γ be a configuration with no activable nodes in V_1'', and u, v two distinct nodes of V_1'' such that $\Omega_u \neq \Omega_v$ and $\beta_u = \beta_v = \perp$ in γ. The set $\Omega_u \cup \Omega_v$ does not form a clique in G.*

Proof. Suppose by contradiction that $\Omega_u \cup \Omega_v$ forms a clique in G with $\Omega_u \neq \Omega_v$.

By hypothesis, we have the properties from Lemma 5.5.1.

Thus $Stab(u)$ and $Stab(v)$, and we may suppose w.l.o.g. that $leader(u) = u$ and $leader(v) = v$.

The fact that $\Omega_u \cup \Omega_v$ is a clique of G means that $\Omega_u \cup \Omega_v \subseteq \bigcap_{x \in \Omega_u \cup \Omega_v} N(x) \cup \{x\}$, but by well-definedness we have $\forall x \in \Omega_u \cup \Omega_v, \mathcal{N}_x = N(x) \cup \{x\}$.

Moreover, $\Omega_u \cup \Omega_v = \emptyset$, since it would imply that $\Omega_u = \Omega_v$ using $Stab(v)$ and $Stab(u)$.

Then, since $leader(v) = v$ and $Stab(v)$, $v \in merge\_candidate(u)$. Then, since we have $\beta_u = \beta_v = \perp$, either there exists $w \in merge\_candidate(u)$ such that $\beta_w = u$ and **Mariage** is activable on u, or **Seduction** is activable on u. □

Our correction property follows immediately from Lemmas 5.5.1 and 5.5.2.

Lemma 5.5.3. *Let γ be a configuration with no node activable in V_1'' and where every node of γ has β-value $\perp$, $\mathcal{C}(\gamma)$ is a minimal clique decomposition of $V_1''(\gamma)$.*

5.6 . Conclusion

We have proved that Minimal Clique Decomposition can be solved in $O(\Delta n)$ rounds with high probability in the presence of Byzantine faults under the fair daemon.

The same algorithm could be used to do the same in a context without Byzantine nodes but under the adversarial daemon. However, as we used probabilistic rules to prevent the Byzantine nodes to be able to reliably trap us, we could probably remove randomness in this context, by saying $choose(A) = min(A)$ instead of drawing an element uniformly.

6 - Minimally Colored Maximum Matching

As we already said, graphs are a powerful modelization tool, whose uses are widespread. But when dealing with complex systems, we often want to use additional information along with the structure they offer. There are many works that deal with labeled graphs, such as edge-weighted graphs, that add a such new layer of information on the edges of the graph.

Another natural path, the one we will focus on, is to add information on the vertices. Here, we study graphs where the additional layer of information is given by a coloration on those. This formalism can be used, for example, to model the Web, where we complete the underlying graph with a coloration on each vertex to capture the type of content it holds [13]. By choosing constraints on colors, many new interesting objects and problems emerge.

Before going a bit more technical, we need some formalism about vertex-colored graphs.

6.1 . Notations and definitions

Throughout the chapter, $G = (V, E)$ denotes a simple undirected graph. The vertex and edge-sets vertex and edge-sets of G may also be denoted by $V(G)$ and $E(G)$ respectively. The edge between the vertices x and y (if any) is denoted by xy. The neighborhood $N(u)$ is the set containing all vertices adjacent to u in G. The closed neighborhood of $u \in V$, is defined by $N[u] = N(u) \cup \{u\}$. Given a set of colors $\mathcal{C}$, $G^c = (V, E, c)$ denotes a vertex-colored graph whose vertices are (not necessarily properly) colored by one of the colors in $\mathcal{C}$ by the function $c : V \to \mathcal{C}$. The color of a vertex $x \in V$ is then denoted by $c(x)$.

For any subgraph G' of G, we denote by $c(G') = \{c(x) \mid x \in V(G')\}$ the set of colors (in G^c) of the vertices of G'. Whenever H is a subset of E, then $V(H)$ denotes the vertex set of the subgraph of G induced by H. In that case, for simplicity, we write $c(H)$ instead of $c(V(H))$. A subgraph H of G is said to be *tropical* (with the coloration c) when $c(H) = c(G^c)$. A *matching* M is a subset of E without adjacent edges. A matching is *maximal* if no proper superset of M is also a matching whereas a *maximum* matching is a maximal matching with the highest cardinality among all possible maximal matchings. A *perfect* matching is a matching of size $\frac{|V|}{2}$.

Following the definitions above, a matching M of G^c is said to be *tropical* if and only if $c(M) = c(G^c)$. A tropical maximum matching (when it exists) is a tropical matching with size the size of a maximum matching.

6.2 . Introduction to the MCMM problem

The present work on colored matchings follows a previous study on another variation of that problem where the maximum matching was said to be tropical [16] (each color has at least one representative in the subgraph), a notion first introduced in [19].

The problem of finding a maximum matching is known to be polynomial [30], but what happens when we add some constraint on the colors to the problem? For example, one could think about the *tropical* version of the problem:

Tropical Maximum Matching

Input: *A vertex-colored graph* G^c
Output: *A tropical maximum matching* M *of* G^c*, if any*

Observe that a perfect matching is always tropical. As a consequence, the above question is only interesting for maximum (not perfect) matchings. In [16], the authors handle efficiently this case by giving a polynomial-time algorithm. Using their Theorem 2.2, an immediate corollary is that we still have a polynomial-time algorithm when we replace *tropical* with *maximum colored*:

Maximum (vertex-)colored Maximum Matching

Input: *A vertex-colored graph* G^c
Output: *A maximum matching* M *in* G^c *with maximum number of colors*

Another natural variation is to consider the minimization of the number of colors instead of maximizing it.

Minimum (vertex-)colored Maximum Matching (MCMM)

Input: *A vertex-colored graph* G^c
Output: *A maximum matching* M *in* G^c *with minimum number of colors*

To introduce the problem, we present the following complexity results on very simple graph families: on both complete graphs and complete bipartite graphs the problem can be solved in linear time.

Proposition 6.2.1. *MCMM is linear in complete graphs.*

Proof. If the number of vertices is even, every vertex will be matched and there is no choice to make (every matching will be equivalent).

If the number of vertices is odd, there is one choice to make: which vertex is going to be out of the matching. It is enough to choose a vertex (if any) whose color is unique in the graph. Otherwise, choose any arbitrary vertex.

Then it is enough to greedily construct a maximum matching on the complete graph -minus the chosen vertex if the number of vertices was odd- which takes linear time. □

Proposition 6.2.2. *MCMM is linear in complete bipartite graphs.*

Proof. Consider a bipartite graph (X, Y, E) with a coloration on vertices. Suppose $|X| \leq |Y|$. It is easy to see that any MCMM matches all the vertices of X. Then it is enough to do the following:

- Take vertices in Y that use colors already used in X until there is enough of them to create a maximum matching with the vertices of X or there are none left.
- If the latter, complete by taking vertices of the most represented color that has vertices left until enough vertices are taken.

Any choice of pairing between vertices of X and vertices of the constructed set gives a MCMM. □

However, as we prove in this chapter, the minimum colored version in its general case is not as easy to solve as the previous tropical and maximum colored variations. In fact, we prove (among other things) that the corresponding decision problem is NP-hard.

The chapter is organized as follows.

- In Section 6.3 we build a linear reduction of the Dominating Set problem parametrized by the size of the solution (a subset $S \subseteq V$ is a dominating set of a graph $G = (V, E)$ if every vertex either belongs to S or has a neighbor in S) to MCMM parametrized by the number of colors of the solution. As the Dominating Set problem is known to be W[2]-complete since the introduction of the W-hierarchy (in [28]), thus we get the $W[2]$-hardness of MCMM. The reduction used to do so also allows us to prove the NP-hardness of MCMM along the way.
- In Section 6.4, we deal with approximation issues. Since finding a classical (not colored) maximum matching is easy, one could spontaneously consider using the number of colors of such maximum matching as a measure to evaluate its quality. However, in Theorem 3.1 we show that *Minimum colored Maximum Matching*, using the number of colors of a solution as parameter, is as hard to approximate as the Minimum Set Cover problem. We do so by using a reduction that is quite similar to the one of Section 6.3.
- In Section 6.5, using another parameter -the size of a maximum matching- we show that the problem becomes *fixed-parameter tractable* (FPT).

6.3 . NP-hardness and W[2]-hardness of MCMM

In this section, we prove the following hardness result.

Theorem 6.3.1. *Minimum colored Maximum Matching (MCMM) is $W[2]$-hard on trees considering the total number of colors of the input as parameter.*

This section is devoted to prove this theorem.

Recall that a *dominating set* of a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ such that every vertex of the graph is either in S or has at least a neighbor in S. The classical optimization problem is then to minimize the size of such a subset.

Minimum Dominating Set

Input: *A graph G*
Output: *A dominating set S of minimum size*

The natural corresponding parameterized problem, where the parameter is the size of a solution, is known to be $W[2]$-complete [28]. The proof of the theorem is then based on a linear reduction from the Dominating Set problem and uses the construction and lemmas below. In particular, it will be an immediate consequence of Lemma 6.3.8.

We will now introduce a construction of an instance of MCMM from an instance of Dominating Set:

Given a connected simple non-colored graph $G = (V, E)$, let us define from G a vertex-colored tree T^c as follows :

- $V(T) = \{x_u \mid u \in V\} \cup \{x_{u,v} \mid u \in V, v \in N[u]\} \cup \{x'_0, x_0\}$,
- $E(T) = \{x_0x_u \mid u \in V\} \cup \{x_ux_{u,v} \mid u \in V, v \in N[u]\} \cup \{x'_0x_0\}$.

Then, we color the vertices of T using $V \uplus \{0\}$ as set of colors:

- $c(x'_0) = c(x_0) = 0$,
- For every $u \in V$, $c(x_u) = 0$,
- For each $(u, v) \in V \times N[u]$, $c(x_{u,v}) = v$.

Notice that $|V(T)| = 2|V|+2|E|+2$, and $|E(T)| = 2|V|+2|E|+1$, and that we can build T^c from G in polynomial time. Notice also that there are $|V| + 1$ internal vertices.

To make discussions easier, we let $\mathcal{R}$ denote the function that given G as input returns T^c. The following series of lemmas explores the properties of $\mathcal{R}$.

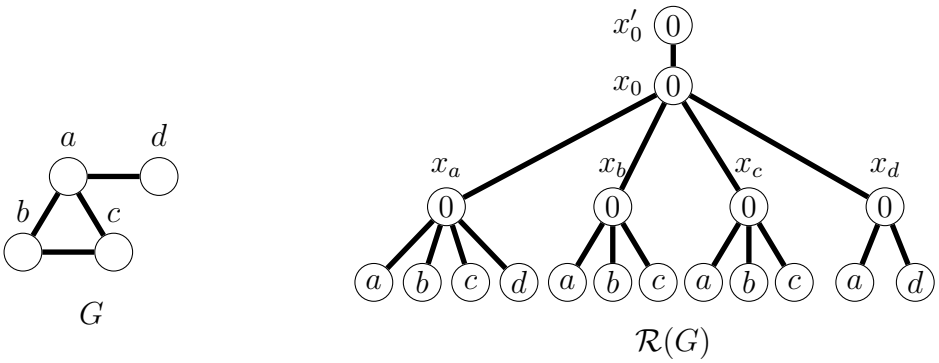


Figure 6.1: A graph G, its transformed version $\mathcal{R}(G)$ (colors depicted in the nodes).

Lemma 6.3.2. *$\{x'_0x_0\} \cup \{x_ux_{u,u} \mid u \in V\}$ is a maximum matching of $\mathcal{R}(G)$.*

Proof. One can easily see that $\{x'_0x_0\} \cup \{x_ux_{u,u} \mid u \in V\}$ is a matching and that there is no augmenting path since all paths between an unmatched vertex to another one are of length 4. Thus, this matching is a maximum one. □

An immediate consequence of the previous lemma is that the size of any maximum matching of $\mathcal{R}(G)$ is $|V| + 1$.

Lemma 6.3.3. *If M is a matching of $\mathcal{R}(G)$ and $M \cap \{x_0x_u \mid u \in V\} \neq \emptyset$, then M is not a maximum one.*

Proof. Let M be a matching of $\mathcal{R}(G)$.

Assume that $x_0x_u \in M$ for some $u \in V$. Then since M is a matching, x'_0x_0 and $x_ux_{u,u}$ are not in M, so $x'_0, x_0, x_u, x_{u,u}$ is an augmenting path and M is not maximum. □

Lemma 6.3.4. *Let u be a vertex in G. Any maximum matching of $\mathcal{R}(G)$ uses exactly one edge in $\{x_ux_{u,v} \mid v \in N[u]\}$ and contains the edge x'_0x_0.*

Proof. Assume that a matching M of $\mathcal{R}(G)$ has no edge in $\{x_ux_{u,v} \mid u \in N[u]\}$. Since $x_0x_u \notin M$ by Lemma 6.3.3, x_u is unmatched in M. Then $M \cup \{x_ux_{u,u}\}$ is a matching greater than M, and M is not maximum. Thus any maximum matching must contain at least one edge in $\{x_ux_{u,v} \mid u \in N[u]\}$, and thus contains exactly one as they all have x_u as an end.

By the same argument, x'_0x_0 must be in any maximum matching, which concludes the proof. □

Let M be a maximum matching of $\mathcal{R}(G)$. We then define a function g by $g(M) = \{v \mid \exists u \in V, x_ux_{u,v} \in M\}$.

Lemma 6.3.5. *If M is a maximum matching of $\mathcal{R}(G)$, then $g(M)$ is a dominating set of G.*

Proof. Let u be a vertex of G. As M is a maximum matching of $\mathcal{R}(G)$, by Lemma 6.3.4, M has one edge in $\{x_u x_{u,u}, x_u x_{u,v} : vu \in E\}$, say $x_u x_{u,v}$.

By the definition of $g(M)$, $v \in g(M)$, which ensures that u is dominated by v and also by $g(M)$. □

Lemma 6.3.6. *If M is maximum matching of $\mathcal{R}(G)$ with $k+1$ colors, then $g(M)$ is a dominating set of G of size k.*

Proof. Let M be a maximum matching of $\mathcal{R}(G)$ with $k+1$ colors.

By Lemma 6.3.4, x_0 of color 0 is covered by M. Thus M has k other colors in V. If $c(M)$ contains the color $v \in V$, then by construction of $\mathcal{R}(G)$, there is some u such that $x_u x_{u,v} \in M$. The definition of the function g implies that $v \in g(M)$. Thus $|g(M)| \geq k$.

Conversely, if M does not contain a color $v \in V$, by construction of $\mathcal{R}(G)$, there is no vertex u such that $x_u x_{u,v} \in M$. Moreover, by definition of g, $v \notin g(M)$. Thus, $|g(M)| \leq k$.

We conclude that $|g(M)| = k$, and since $g(M)$ is a dominating set of G by Lemma 6.3.5, $g(M)$ is then a dominating set of G of size k. □

Lemma 6.3.7. *Graph G admits a dominating set of size k if and only if $\mathcal{R}(G)$ admits a maximum matching with $k+1$ colors.*

Proof. By Lemma 6.3.6, if $\mathcal{R}(G)$ admits a maximum matching with $k+1$ colors, G admits a dominating set of size k.

Conversely, assume that S is a dominating set of size k in G.

Let α be an arbitrary injective valuation on V. For each $u \in V$ we define a function φ by

$$\varphi(u) = \begin{cases} u, \text{ if } u \in N_G[u] \cap S \\ min_\alpha(N_G[u] \cap S), \text{ otherwise} \end{cases}$$

Since S is a dominating set of G, for each $u \in V, N_G[u] \cap S$ is not empty, and φ is then well-defined.

Then we define $M = \{x'_0 x_0\} \cup \{x_u x_{u,v} \mid v = \varphi(u)\}$. M is a matching by construction, and is maximum since it is of size $|V| + 1$. Furthermore, any vertex covered by M is of color either 0 or $u \in S$, and each of those $k+1$ colors appears at least once (if $u \in S$ then by construction $x_u x_{u,u} \in M$, and $x_{u,u}$ has u as a color). Consequently, M is $k+1$-colored, which concludes the proof. □

To prove $W[2]$-hardness with our reduction $\mathcal{R}$, we need to show that it is in fact a FPT reduction, that is:

1. $\mathcal{R}$ is a reduction from Dominating Set to MCMM.
2. $\mathcal{R}$ is computable with a FPT algorithm.
3. A computable function g must exist such that the parameter of MCMM (the number of colors of the optimal maximum matching) in $\mathcal{R}(G)$ is less than g applied to the parameter (the size of an optimal solution) of Dominating Set in G.

Lemma 6.3.8. *$\mathcal{R}$ is a FPT-reduction from the Dominating Set problem with parameter size of the optimal solution to the MCMM problem on trees with parameter number of colors of the optimal solution.*

Proof. Point 1 and Point 3 are proven in Lemma 6.3.7.

The computation of $\mathcal{R}$ is polynomial in the size of G (see the construction). It is as such also FPT, and we have Point 2.

Thus, $\mathcal{R}$ is a FPT-reduction. □

Proof of Theorem 6.3.1. It is an immediate consequence of Lemma 6.3.8 using the fact that the Dominating Set problem is known to be $W[2]$-complete [28]. □

Theorem 6.3.9. *Minimum colored maximum matching is NP-complete on trees.*

Proof. It is enough to see that $\mathcal{R}$ is also a polynomial reduction from the Dominating Set problem to the MCMM problem on trees. □

6.4 . Hardness of approximating MCMM

We consider as candidate for approximating MCMM any maximum matching, with the weight function being the number of colors used. For that definition, we prove the following inapproximability result.

Theorem 6.4.1. *MCMM cannot be approximated on trees with an approximation ratio better than $\log(N-1)(1-\varepsilon)$ (with $0<\varepsilon<1$), where N is the number of internal vertices (vertices with degree at least 2) of G, unless P=NP.*

The proof of those theorems is based on a reduction from the Set Cover problem, which is known not to be approximable beyond a certain logarithmic ratio [25].

Minimum Set Cover

Input: *A finite set U, and $\mathcal{F} \subset \mathcal{P}(U)$ such that $U = \bigcup_{F \in \mathcal{F}} F$*

Output: *$\Xi \subset \mathcal{F}$ such that $U = \bigcup_{F \in \Xi} F$ with minimum cardinality*

As it is more convenient for us, we will use the equivalent following form of the problem :

Minimum Set Cover (bipartite graph)

Input: *A bipartite graph $G = (U, V, E)$ such that no $u \in U$ is isolated and no two v, v' distinct vertices of V have the same neighborhood*

Output: *$\Xi \subset V$ such that $U = \bigcup_{v \in \Xi} N(v)$ with minimum cardinality*

The proof of Theorem 6.4.1 uses the following construction and lemmas below. Note that the construction, and hence the following lemmas and proofs, are very close to what was done in the previous part. It should not be very surprising, given the proximity between the Dominating Set and the Set Cover problems.

Construction

Given an instance of Set Cover $G = (U, V, E)$, we define a vertex-colored tree T^c defined as follows:

- $V(T) = \{x'_0, x_0\} \cup \{x_u \mid u \in U\} \cup \{x_{u,v} \mid u \in U, uv \in E\}$,
- $E(T) = \{x_u x_{u,v} \mid u \in U, uv \in E\} \cup \{x_u x_0 \mid u \in U\} \cup \{x'_0 x_0\}$.

Then we color the vertices of T with $n + 1$ colors so that :

- $c(x'_0) = 0$, $c(x_0) = 0$, and for each $u \in U$, $c(x_u) = 0$,
- For each $uv \in E$, $c(x_{u,v}) = v$.

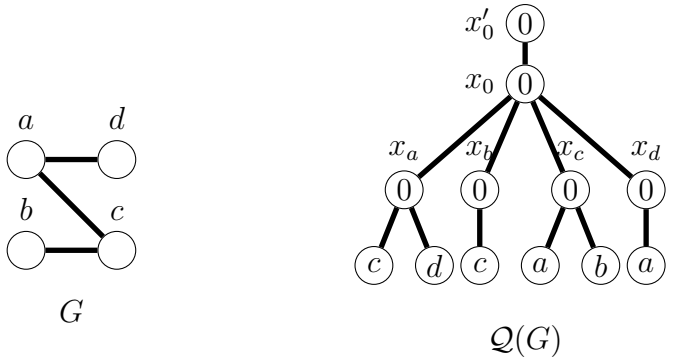


Figure 6.2: A bipartite graph G, and its transformed version $\mathcal{Q}(G)$ (colors depicted in the nodes).

Note that $|V(T)| = |U| + |E| + 2$, $|E(T)| = |U| + |E| + 1$, and that we can obtain T^c in polynomial time from G. Note also that there are $|U| + 1$ internal vertices.

We use $\mathcal{Q}$ to denote the function that given G as input returns T^c. The following lemmas explore the properties of $\mathcal{Q}$ to prove that it is indeed a reduction of Set Cover to MCMM on trees.

Lemma 6.4.2. *$\{x'_0x_0\} \cup \{x_ux_{u,v} \mid u \in U, uv \in E\}$ is a maximum matching of G.*

Proof. One can easily see that there is no augmenting path since all paths that go from an unmatched vertex to another are of length 4. □

Moreover, no maximum matching can use an edge that does not cover a leaf since this would create an augmenting path.

Lemma 6.4.3. *If M is a matching of $\mathcal{Q}(G)$ and $M \cap \{x_0x_u \mid u \in U\} \neq \emptyset$, then M is not a maximum matching.*

Proof. Let M be a matching of $\mathcal{Q}(G)$.

Let's suppose that $x_0x_u \in M$ for some $u \in U$. Since there is no isolated vertex in G, there exists $v \in V$ such that $uv \in E$. Then since M is a matching, x'_0x_0 and $x_ux_{u,v}$ are not in M, so $x'_0x_0x_ux_{u,v}$ is an augmenting path and M is not maximal therefore not maximum. □

Lemma 6.4.4. *Consider $u \in U$. Any maximum matching of $\mathcal{Q}(G)$ uses exactly one edge in $\{x_ux_{u,v} \mid v \in N(u)\}$, and contains the edge x'_0x_0.*

Proof. Let M be a matching of $\mathcal{Q}(G)$. Suppose that there exists $u \in U$ such that $M \cap \{x_ux_{u,v} \mid v \in N(u)\} = \emptyset$. Since $x_0x_u \notin M$ by Lemma 6.4.3, x_u is unmatched in M. Then $M \cup \{x_ux_{u,v}\}$ would be a matching of greater size, thus M cannot be maximum. Thus, any maximum matching must contain at least one edge in $\{x_ux_{u,v} \mid v \in N(u)\}$, and thus contains exactly one as they all have x_u as an end.

By the same argument, any maximum matching must contain x'_0x_0, which concludes the proof. □

Given a maximum matching M of $\mathcal{Q}(G)$, we then define g by $g(M) = \{v \in V \mid \exists u, x_ux_{u,v} \in M\}$.

Lemma 6.4.5. *If M is a maximum matching of $\mathcal{Q}(G)$, then $g(M) \cup \{0\} = c(M)$ and $g(M) = c(M) \setminus \{0\}$.*

Proof. Let M be a maximum matching of $\mathcal{Q}(G)$.

For $v \in g(M)$, by definition of $g(M)$, there is $u \in U$ such that $x_u x_{u,v} \in M$, thus $v \in c(M)$. Since we have by Lemma 6.4.4, $x'_0 x_0 \in M$, we have also $0 \in c(M)$, thus $g(M) \cup \{0\} \subset c(M)$.

Conversely, for $v \in c(M) \setminus \{0\}$, there must be $u \in U$ such that $x_u x_{u,v} \in M$ as only vertices $x_{u,v}$ have color v. By definition of $g(M)$, $v \in g(M)$. Thus, $c(M) \subset g(M) \cup \{0\}$.

Therefore, we have $g(M) \cup \{0\} = c(M)$, and the second equality follows immediately, as $0 \notin g(M)$ by definition. □

Lemma 6.4.6. *If M is a maximum matching of $\mathcal{Q}(G)$, then $g(M)$ is a set cover of G (i.e. a subset of U whose union of neighborhoods gives V).*

Proof. Let M be a maximum matching of $\mathcal{Q}(G)$ and u be a vertex from U. As M is a maximum matching of $\mathcal{Q}(G)$, by Lemma 6.4.4 there exists v such that $x_u x_{u,v}$ is in M, which ensures that u is covered by $g(M)$. □

Lemma 6.4.7. *If M is a $k+1$-colored maximum matching of $\mathcal{Q}(G)$, then $g(M)$ is a set cover of G of size k.*

Proof. Let M be a maximum matching of $\mathcal{Q}(G)$.

By Lemma 6.4.6, $g(M) = c(M) \setminus \{0\}$, so we have $|g(M)| = |c(M)| - 1 = k$ (since $0 \in c(M)$ by direct corollary of Lemma 6.4.4). By Lemma 6.4.6, $g(M)$ is also a set cover, which concludes the proof. □

Lemma 6.4.8. *A bipartite graph G admits a minimal set cover of size k if and only if $\mathcal{Q}(G)$ admits a minimally colored maximum matching (i.e., a matching whose set of colors is minimal but could not be minimum) with $k+1$ colors.*

Proof. Let α be a choice function on V (*i.e.* a function which, for any non-empty subset of V, gives an element of the said subset).

By Lemma 6.4.7, if $\mathcal{Q}(G)$ admits a minimally-colored maximum matching M with $k+1$ colors, then G admits a set cover $g(M)$ of size k. Assume that $g(M)$ was not minimal, *i.e.* that there exists $v_0 \in g(M)$ such that $g(M) \setminus \{v_0\}$ is a set cover of size $k-1$. For $u \in U$, let us write:

$$\varphi(u) = \alpha(\{v \mid uv \in E, v \in g(M) \setminus \{v_0\}\})$$

which is well-defined since $g(M) \setminus \{v_0\}$ is a set cover of G. We can then define $M' = \left\{x_u x_{u,\varphi(u)} \mid u \in U\right\} \cup \{x_0 x'_0\}$. Notice that M' is a maximum matching since it is a matching of size $|U|+1$. By construction, its color set is included in $(g(M) \cup \{0\}) \setminus \{v_0\}$, which contradicts the minimality of the color set of M.

Conversely, let S be a minimal set cover of G of size k. For $u \in U$, let us denote $\psi(u) = \alpha(\{v \mid uv \in E, v \in S\})$ (which is well-defined since S is a set cover of G). Then we define $M = \{x'_0 x_0\} \cup \{x_u x_{u,\psi(u)} \mid u \in U\}$. This matching M is of the same size as the one presented in Lemma 6.4.2. Thus it is a maximum matching with at most $k+1$ colors since all colors used are in $S \cup \{0\}$. It remains to prove that M has $k+1$ colors and is minimally colored. If it is false, that would mean either that it is not minimally colored, or that M has not $k+1$ colors.

- If M was not minimally-colored, there would be a maximum matching M' of $\mathcal{Q}(G)$ such that $c(M') \subsetneq c(M) \subset S \cup \{0\}$.

- If M had not $k+1$ colors, then we would have $c(M) \subsetneq S \cup \{0\}$.

In both case, there exists a matching M° such that $c(M^\circ) \subsetneq S \cup \{0\}$ which is equivalent to $g(M^\circ) \subsetneq S$. But $g(M^\circ)$ is a set cover of G of size at most $k-1$ (by Lemma 6.4.8), which contradicts the minimality of S.

□

Proof of Theorem 6.4.1

From every not minimal set cover, one can extract in polynomial time a minimal set cover that is smaller than the previous one. Then, without loss of generality, we only consider minimal set covers as approximation candidates for the Minimum Set Cover problem.

Let's suppose that MCMM is approximable with a ratio $f(N)$ where N is the number of internal vertices (vertices of degree at least 2) of the MCMM instance.

Given an instance G of the Set Cover problem (U, V, E) with universe U of size k, we use $\mathcal{Q}$ to compute in polynomial time an instance of MCMM (of polynomial-size in $|U|$ and $|V|$), with $k+1$ internal vertices. By the above hypothesis, we can compute a $f(k+1)$-approximation of that instance of MCMM. Then we can use g to build in polynomial time a set cover which is, by Lemma 6.4.8, of the same size as the approximate solution to MCMM, that is, at most a $f(k+1)$-approximation of the solution of the Minimum Set Cover on G.

Then, if $f(N)$ was asymptotically smaller than $\log(N-1)(1-\varepsilon)$, the corresponding approximation ratio for *Set Cover* would be better than $\log(k)(1-\varepsilon)$, contradiction unless P=NP [25].

Thus, Theorem 6.4.1 holds. □

6.5 . MCMM is FTP when parameterized by the maximum size

of a matching in the input graph

This section is devoted to prove the following result:

Theorem 6.5.1. *MCMM is FPT with the size of a maximum matching in the input as parameter.*

To show this, we construct an exploration tree in a similar way as in [31].

Let $G^c = (V, E, c)$ be a vertex-colored graph with maximum matching size k.

We consider an arbitrary maximum matching M_0 of G (which can be built in polynomial time). It will be used as reference to decompose other matchings.

Notation. *$I_0 = V(G) \setminus V(M_0)$, and $G[M_0]$ be the subgraph induced by $V(M_0)$ in G.*

If we consider a maximum matching M^*, each edge of M^* has at least one shared vertex extremity with M_0 (otherwise M_0 would not be a maximum matching). Thus we can split the edges of M^* into two parts, the one included in $G[M_0]$ and the remaining ones. We use that property to decompose the search for an optimal solution.

In a similar way, we use other "natural" splits to decompose the configuration space we want to explore (*i.e* the set of every possible maximum matching). For the first splits, we remain exhaustive (as, for those, it does not cost much). Then we make choices that break exhaustivity, we will have to prove afterward that if we miss some optimal solutions with those, we cannot miss them all.

Formally, we do so by building a rooted exploration tree, where each node represents the subset of matchings that are compatible with the choices made along the path from the root to the node. As such, every descendant of a node will represent a subset of matchings of those of its ancestors. The construction is performed as follows:

We create the root ω_0 which represents all possible maximum matchings, since no choice has been made so far. Every other vertex of the exploration tree will be given a label that contains the choices made at that level.

Then from the root we branch, for every possible selection (M, S) where $M \subset E(G[M_0])$ is a matching, $S \subset V(G[M_0]) \setminus V(M)$, and $|M| + |S| = k$, by adding a child $\omega_{M,S}$ labeled (M, S).

The exploration tree vertex $\omega_{M,S}$ represents the set of all maximum matchings that are compatible with the choice of the sets M and S: each of those matchings contains M, and every other edge of those matchings has one end in S (and the other in I_0) (see Figure 6.3).

Note that the condition $|M| + |S| = k$ comes from the fact that we search for a matching of size k, with all edges of M, and with an edge for each vertex in S. In

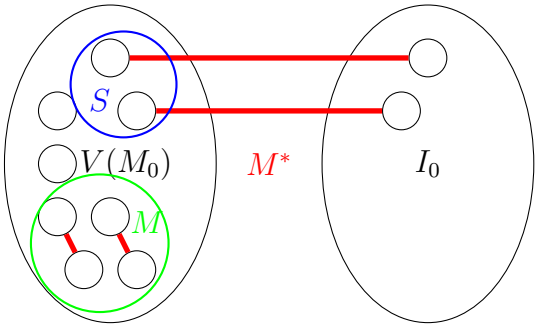


Figure 6.3: Decomposition of a matching M^* according to the structure of M_0.

this branch, and for every future branching under it, we will write $C = c(M) \cup c(S)$ for the sake of readability.

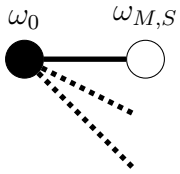


Figure 6.4: The exploration tree after the addition of the first layer of vertices.

Do note that since we branched for every possible choice, the sets of matchings represented by the children of the root form a partition of the set of all possible matchings.

Observation: At that point, we have created at most $T_k\binom{2k}{k}$ new leaves where T_i is the i-th telephone number (the number of possible matchings in a clique of size i).
This enumeration can be done in time $O(k \times T_k 2^{2k})$ ($O(k)$ by distinct choice).

Then, we want to consider the partition of S according to the color of the matching vertex in I_0 (see Figure 6.5).

To capture every potential such partition, for every leaf $\omega_{M,S}$ labeled (M, S) we branch for every partition Σ of S by adding a child ω_Σ labeled (Σ).

The set of matchings represented by ω_Σ is a subset of the one of its father $\omega_{M,S}$. It only keeps from its father the matchings that have Σ as a partition of S when you partition it with respect to the color elements of S are associated with on the I_0 side by the matching. Here again, as we branched for every possible choice of Σ, the sets represented by the children of $\omega_{M,S}$ form a partition of the set represented by their father.

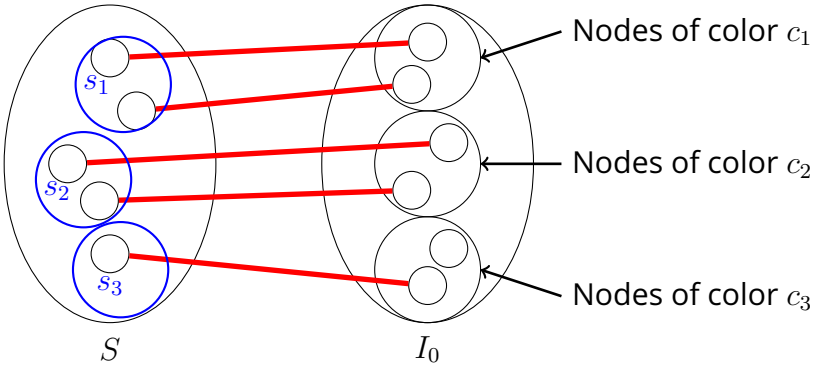


Figure 6.5: The partition $\Sigma = \{s_1, s_2, s_3\}$ of S by matching color on the "exterior side" (I_0). Red edges are the edges of a matching compatible with the partition Σ.

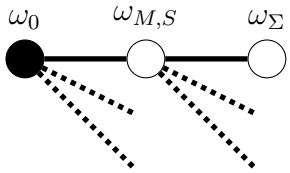


Figure 6.6: The exploration tree after the addition of the second layer of vertices.

Observation : For each leaf of the exploration tree at the previous step (leaves in Figure 6.4), we have created the $B_{|S|} \leq B_k$ possible partitions of S, where B_i is the i-th Bell number (the number of possible partitions of a set with i elements).
They can be enumerated in time $O(kB_k)$ ($O(k)$ by distinct partition).

Now, we have to assign a different color to every part of Σ. It will either be a color already in C, or a new color.

To cover the possible combination of those two options, for every leaf ω_Σ son of $\omega_{M,S}$, we branch for every possible choice of partial injective coloration of nonempty parts of Σ by colors of C, Ξ, by adding a child ω_Ξ labeled Ξ. Parts of Σ that are attributed the value 0 will be attributed a new color (*i.e.* not in C) later on in the construction of the exploration tree.

We formally define Ξ as a function $\Xi : \Sigma \to C \uplus \{0\}$ injective on $\Sigma \setminus \Xi^{-1}(0)$.

The set of matchings represented by ω_Ξ is a subset of the one of its father $\omega_{M,S}$. It only keeps the matchings that have, for every $s \in \Sigma$, the vertices of s matched with vertices of I_0 of color $\Xi(s)$ if $\Xi(s) \neq 0$, and matched with vertices of the same color not in C otherwise.

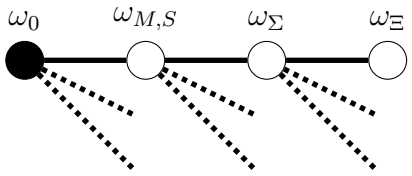


Figure 6.7: The exploration tree after the addition of the third layer of vertices.

Observation : For each leaf of the exploration tree at the previous step (leaves in Figure 6.6), we have created at most

$$\sum_{i=0}^{min(|C|,|\Sigma|)} i! \binom{|C|}{i} \leq min(|C|,|\Sigma|)! \times 2^{|C|} \leq k! \times 2^k$$

possible partial injective colorations of Σ (and that many new leaves), which can be enumerated in time $O(k \times k! \times 2^k)$ ($O(k)$ by distinct coloration).

Now we want to build partial matchings for every s of Σ, between s and I_0, where every vertex on the I_0 side has the same color. $\Xi(s)$ if $\Xi(s) \neq 0$, any color not in C otherwise. The goal being to be able to choose one partial matching for every part of the partition, with distinct colors, to compute a maximum matching. Note that if no partial matching exists for some $s \in \Sigma$, it means that the choices already made above in the tree do not lead to the construction of a valid maximum matching.

Formally, for every leaf ω_Ξ produced at the previous step, we compute matchings for every part of the partition $s \in \Sigma$ (the values of M, S and Σ are those that appear in the branch from the root to the said leaf):

- If $\Xi(s) \neq 0$, we compute, if any, μ a matching between s and vertices of I_0 of color $\Xi(s)$, and we write $\Gamma(s) = \{\mu\}$. If no such matching exists, $\Gamma(s) = \emptyset$.
- If $\Xi(s) = 0$ then for every color $c_0 \in c(V)$, we compute, if any, μ a matching between s and vertices of I_0 of color c_0, and denote by $\Gamma(s)$ the set of those matchings truncated at $k+1$ (we stop the computation when we already have $k+1$ such matchings).

Then we add exactly one child ω_Γ labeled Γ to ω_Ξ.

The set of matchings represented by ω_Γ is a subset of the one of its father. It only keeps the matchings whose restriction to the edges that have an end in $s \in \Sigma$ is in $\Gamma(s)$ for every $s \in \Sigma$. Do note that here we do not keep the exhaustivity, as we may have lost some matchings in the process.

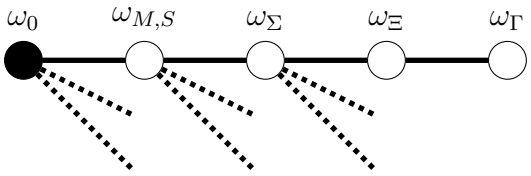


Figure 6.8: The exploration tree after the addition of the fourth layer of vertices.

Observation : For each leaf of the exploration tree at the previous step (leaves in Figure 6.7), for every color, we compute at most a maximum matching, each one being computed in $O(k^{5/2})$ [15].

Now that we have those partial matchings, we can build the bipartite graph with the elements of Σ on the left side, and colors on the right side. With an edge between $s \in \Sigma$ and a color c if there is a partial matching with color s on the I_0 side in $\Gamma(s)$. To compute a maximum matching that observes the constraints already chosen, it is enough to find a maximum matching of the graph we have just built, then take the union of the partial matchings corresponding to the edges of the maximum matching (see Figure 6.9).

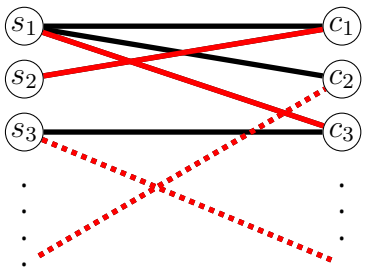


Figure 6.9: A matching in the bipartite graph with parts Σ to the left and colors of available partial matching to the right.

Formally, for each leaf ω_Γ, we compute a maximum matching γ on the bipartite graph

$$(\Sigma, c\left(\bigcup_{(s,\mu)\in\Sigma\times\Gamma(s)} V(\mu)\cap I_0\right), \{s\mathcal{C}(\mu) \mid s\in\Sigma, \mu\in\Gamma(s)\}),$$

where $\mathcal{C}(\mu)$ denotes the only color in $c(V(\mu)\cap I_0)$. (The values of M, S, Σ, and Ξ are those that appear in the branch from the root to the said leaf.)

Then we add a child ω_∞ to the said leaf. If $|\gamma| = |\Sigma|$, we define $M_\infty = \bigcup_{s\mathcal{C}(\mu)\in\gamma}\mu$, and we label the child with M_∞. Else it means that we failed to construct a maximum matching with the set of constraints we have, and we label it $\perp$.

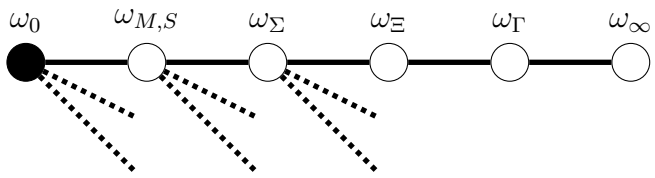


Figure 6.10: The completed exploration tree.

Here, the new leaf represents either exactly 1 maximum matching, or the empty set.

Observation : For each leaf of the exploration tree at the previous step (leaves in Figure 6.8), the computation of the auxiliary matching can be done in $O(k^{5/2})$ [15], the following computation of a matching of G takes $O(k^2)$, and finally the computation of its number of colors $|c(M_\infty)|$ in $O(k)$. It is then a $O(k^{5/2})$.

Lemma 6.5.2. *The exploration tree described above can be computed in time $O(k^4 T_k B_k k! 2^{3k}|V|)$ from a given maximum matching on G.*

Proof. From the analysis boxed between steps of the tree construction, we have that the tree can be computed in

$$O(k^{1/2}2^{2k}T_k \times (k + B_k(k + k! \times 2^k(k + k \times |c(G)| \times k^{5/2} + k^{5/2}))$$

Which is then $O(k^4 T_k B_k k! 2^k |V|)$, (by taking $|c(G)| = O(|V|)$). □

Remark. *To better visualize that complexity, one can note that for any $\varepsilon > 0$, the above is a $O((\frac{k}{e})^{(3/2+\varepsilon)k}|V|)$.*

Lemma 6.5.3. *There exists a leaf in the exploration tree which is labeled by a maximum matching whose number of colors is minimal.*

Proof. Let M_{opt} be a minimum colored maximum matching. Let us decompose it relatively to M_0 into $M_{opt} = M_{in} \uplus M_{out}$ where :

- $M_{in} = M_{opt} \cap E(G[V(M_0)])$
- $M_{out} = M_{opt} \setminus M_{in}$

Note that every edge in M_{out} must have an end in $V(M_0)$, since it would otherwise contradict the maximality of M_0.

We define $S_{out} = V(M_{out}) \cap V(M_0)$. Note that $S_{out} \cup V(M_{in}) = V(M_0)$. We then go in the exploration tree to the vertex $\omega_{M_{in},S_{out}}$ labeled (M_{in}, S_{out}) which exists since we branched exhaustively on the possible values of M and S.

We compute the following partition of S_{out} :

$$\Sigma = \{\{u \in S_{out} \mid uv \in M_{out}, c(v) = c_0\} \mid c_0 \in c(G)\} \setminus \{\emptyset\}$$

and search among the children of $\omega_{M_{in},S_{out}}$ for the child labeled Σ, ω_Σ, which exists since we branched on all the possible partitions of S_{out}.

Then we define $\Xi(s)$ as the only color in $c(\{v \mid u \in s, uv \in M_{out}\}) \cap (c(M_{in} \cup c(S_{out}))$ if it is nonempty (there cannot be more than one color in that set from the construction of Σ), and as 0 otherwise. From the construction of Σ, Ξ is injective on $\Sigma \setminus \Xi^{-1}(0)$. We search for ω_Ξ the child of ω_Σ labeled Ξ, which exists since we branched on all possible partial permutations of already chosen colors on the parts of the partition. By construction ω_Σ have one child ω_Γ. Let's consider the maximum matching computed in the fifth step of the creation of the exploration tree between the parts of the partition Σ and the colors. Recall that formally it is a matching in the following bipartite graph:

$$\Omega = (\Sigma, c(\bigcup_{(s,\mu)\in\Sigma\times\Gamma(s)} V(\mu) \cap I_0), \{s\mathcal{C}(\mu) \mid s \in \Sigma, \mu \in \Gamma(s)\}),$$

where $\mathcal{C}(\mu)$ denotes the only color in $c(V(\mu) \cap I_0)$

We know that the computed maximum matching is of size $|\Sigma|$, since we can construct the following matching :

- For all $s \in \Sigma$ that have k or less edges in Ω, we take the edge corresponding to the color attributed to s in M_{OPT} (that is the only color in $c(\{u \mid uv \in M_{OPT}, v \in s\})$). That color appears in Ω since we exhaustively enumerated possible colors for $s \in \Sigma$ that had less than k possible color to match with. Let us denote by $n_{matched}$ the number of $s \in \Sigma$ in this situation.
- Then, there are at most $k - n_{matched}$ parts of Σ that still needs to be matched. For every one of those s, $\Gamma(s)$ contains $k + 1$ matchings of different colors on the I_0 side, that is for every one of those s, it has edges to $k + 1$ colors in Ω, at least $k + 1 - |n_{matched}| > k - |n_{matched}|$ of them not being already matched. We can then choose greedily a different color to match every remaining $s \in \Sigma$.

The described matching is of size $|\Sigma|$, so the maximum matching computed when constructing the exploration tree must have size $|\Sigma|$. Then, by construction of the exploration tree, the only child of ω_Γ cannot be labeled $\perp$, and is labeled with a maximum matching M_∞. In that matching, the parts of Σ that are matched to colors already in M_{in} or in S_{out} are the same as in M_{OPT} (since there is only one edge from those in Ω). Every other part of Σ is matched in M_∞ to a different new color (a color not appearing in M_{in} or

S_{out}) as it is the case in M_{OPT} by construction of Σ. Thus M_∞ has the same number of colors as M_{OPT}, which concludes the proof. □

Proof of Theorem 6.5.1

Since we supposed that G admits a maximum matching of size k, we search for a such matching M_0 in time $O(|E|\sqrt{|V|})$ [53]. We construct the exploration tree described above and then search for a leaf not labeled $\perp$ with minimum number of colors in the matching of its label in time $O(k^4T_kB_kk!2^k|V|)$ (from Lemma 6.5.2). From Lemma 6.5.3, such a matching is a minimum-colored maximum one of G^c. Thus the algorithm above runs in time $O(|E|\sqrt{|V|}) + O(k^4T_kB_kk!2^k|V|)$. Thus Theorem 6.5.1 holds. □

6.6 . APX-completeness on collections of P_2 and P_3

In this section, we prove that MCMM restricted to collections of P_2 and P_3 (paths of length 2 and 3 respectively) is APX-complete.

First, as APX-completeness is defined under *approximation-preserving reductions* (AP-reductions). In the case of minimization optimization problems, *linear reductions* (L-reductions) happens to be also AP-reduction. As they are easier to handle, we will work here with linear reductions. Let us then define linear reductions.

For an optimization problem A, we denote by c_A its cost function, and when x is an instance of problem A, $OPT_A(x)$ is the minimum cost of a solution of problem A on x.

Then if A and B are optimization problems, and we have $\alpha > 0$ and $\beta > 0$ two constants, a linear reduction of ratios (α, β) from A to B is a pair of functions (f, g) such that:

1. f and g are computable in polynomial time,
2. If is an instance of A, then $f(x)$ is an instance of B,
3. If y is a solution to problem B on $f(x)$, $g(y)$ is a solution to problem A on x,
4. $OPT_B(f(x)) \leq \alpha OPT_A(x)$,
5. $|OPT_A(x) - c_A(g(y))| \leq \beta |OPT_B(f(x)) - c_B(y)|$.

Here, we will in fact only consider linear reductions of ratios $(1, 1)$.

As there is a natural reduction from this problem to MCMM restricted to collections of P_2 and P_3 to this problem, we introduce a variation of Minimum Vertex Cover: *Minimum Vertex Cover with mandatory vertices*.

Minimum Vertex Cover with mandatory vertices

Input: *A graph $G = (V, E)$, and $V_0 \subset V$*

Output: *A minimal vertex cover C of G such that $V_0 \subset C$ with minimum cardinality*

This problem is in fact equivalent to the regular Minimum Vertex Cover problem under linear reductions, as we will show.

One direction of this equivalence is trivial as one is a restricted version of the other, the following lemma proves the other direction.

Lemma 6.6.1. *There is a linear reduction from Minimum Vertex Cover with mandatory vertices to Minimum Vertex Cover.*

Proof. Let $G = (V, E)$, $V_0 \subset V$, be an instance of Minimum Vertex Cover with mandatory vertices. We construct an instance of Minimum Vertex Cover $G' = (V', E')$ by taking $V' = V \uplus V_0$ where we will denote $\bar{v}$ the new copy of $v \in V_0$ added in V', and $E' = E \cup \{v\bar{v} \mid v \in V_0\}$. It is trivial to observe that any minimal vertex cover S of G' either uses all vertices of V_0 or can be modified into a vertex cover of same size or less that does not use the new vertices by replacing all the $\bar{v} \in S$ by their corresponding v. Moreover, for any such minimal vertex cover, it is of exact same size, and we have then a linear reduction (with ratios 1 / 1). □

Then we can use the existence of those reductions to prove that Minimum Vertex Cover with mandatory vertices is APX-complete.

Lemma 6.6.2. *Minimum Vertex Cover with mandatory vertices is APX-complete.*

Proof. By Lemma-6.6.1 we have a linear reduction in one sense, and there is a trivial linear reduction from Minimum Vertex Cover to Minimum Vertex Cover with mandatory vertices: we take the same graph, and take $V_0 = \emptyset$. Minimum Vertex Cover being APX-complete, so is Minimum Vertex Cover with mandatory vertices. □

Proposition 6.6.3. *MCMM on collections of P_1 and P_2 is APX-complete.*

Proof. On one hand, there is a linear reduction from Minimum Vertex Cover to MCMM on collections of P_1 and P_2. For an instance of Minimum Vertex Cover $G = (V, E)$, we construct $G' = (V', E')$ colored with c by taking:

- $V' = \{x_{u,uv}, x_{uv}, x_{v,uv} \mid uv \in V\}$,
- $E' = \{x_{u,uv}x_{uv}, x_{uv}x_{v,uv} \mid uv \in V\}$,

- $c(x_{u,uv}) = u$, $c(x_{uv}) = 0$, for every u and v in V such that those nodes exist ($c(G) = V \uplus \{0\}$).

If M is a maximum matching of G'^c, $S = \{u \in V \mid \exists v \in V, x_{u,uv}x_{uv} \in M\}$ is a vertex cover of G. If it was not, there would be $uv \in E$ not covered by S, which would mean that neither $x_{u,uv}x_{uv}$ nor $x_{v,uv}x_{uv}$ are in M, and thus M would not be a maximum matching. Thus any maximum matching M of G'^c correspond to a vertex cover S of G such that $|c(M)| = |S|$. We have then a linear reduction (with ratios 1 / 1).

We construct an instance of Vertex Cover with mandatory Vertices $G' = (V', E')$ with

- $V' = c(V)$,
- $E' = \{c(x_{i,1})c(x_{i,3}) \mid i \in [\![1, k]\!]\}$,
- The mandatory set $V_0 = \{c(x_{i,2}) \mid i \in [\![1, k]\!]\} \cup \{c(y_{i,1}), c(y_{i,2}) \mid i \in [\![1, \ell]\!]\}$.

Then, if S is a minimal (not necessarily minimum) solution of Vertex Cover with mandatory vertices on G' with mandatory vertex set V_0, let's define $\phi(i)$ as 1 if $c(x_{i,1}) \in S$, and 3 otherwise. Observe that when $\phi(i) = 3$, as S is a vertex cover and $c(x_{i,1})c(x_{i,3}) \in E'$, we must have $c(x_{i,3}) \in S$. We can then define $M = \left\{x_{i,\phi(i)}x_{i,2}\right\} \cup \{y_{i,1}y_{i,2} \mid i \in [\![1, \ell]\!]\}$, and it is a maximum matching since it is of cardinality $k + \ell$. We have $c(M) = S$, as if it was not the case it would imply that S is not minimal. Thus we have a linear reduction (with ratio 1 / 1). □

6.7 . Conclusion

In this chapter, we have shown that MCMM is NP-hard, W[2]-hard with the number of colors of the optimal solution as parameter, FPT with the size of a maximum matching as parameter, and that it is hard to approximate.

Several questions are directly raised by those results. The size of a maximum matching is indeed a "big" parameter, and thus it is not very surprising that MCMM is FPT with respect to it, but the classical "small" parameter treewidth is not of any help here, as MCMM is hard even on trees. Is there a sensible parameter, smaller in general than the size of a maximum matching, for which MCMM is FPT? Additionally, we have given an inapproximability result that gives a lower bound for achievable approximation ratios, but no approximation algorithm that would set an upper bound on the best approximation achievable.

Both questions may lead to further study around the MCMM problem.

7 - Conclusion

The work presented in this thesis can be divided in two, the first part focusing on self-stabilization in distributed systems, and the second one on classical graph algorithms. In the self-stabilization part, we deal with Byzantine faults for problems that had no prior algorithm handling those. We also use one of them to propose a way to produce self-stabilizing algorithms for mendable problems in anonymous networks. In the classical graph algorithm part, we study a new problem that extends some previous work on colored matchings and give a hardness result as well as an FPT algorithm in a specific case.

Chapter 3 introduces an algorithm that handles Byzantine faults and solves the MIS problem in anonymous systems in $O(n^2)$ rounds with high probability under the fair distributed daemon. We then give a slightly modified version of this algorithm, that solves the same problem under the adversarial distributed daemon (without handling Byzantine faults) in $O(n^2)$ moves. Chapter 5 introduces an algorithm that handles Byzantine faults and solves the Minimal Clique Decomposition problem in $O(\Delta n)$ rounds under the fair distributed daemon in systems with unique identifiers. As in the chapter about the MIS problem, it should be possible to adapt this algorithm to the non-Byzantine case with the adversarial distributed daemon. Moreover, getting rid of the randomness should be doable in such an adaptation, as randomness was only necessary to confine Byzantine influence.

As it is necessary to handle Byzantine faults to work with some fairness property in the daemon to guarantee the convergence of algorithms, complexities are classically expressed in rounds. However, we may be interested in quantifying the amount of work done by "non-compromised" nodes by expressing the complexities in terms of moves of those nodes. Excluding Byzantine nodes is obviously not enough: Byzantine influence may, depending on the algorithm, make non-Byzantine nodes do an unbounded number of moves before anything else happens. However, we often rely on choosing a containment radius and then relaxing the constraint by considering "well-defined" nodes in a sense that we can be sure that such a node cannot fall back under Byzantine influence. Thus, we could express a move complexity that would count every move made by nodes outside the initial containment radius, plus the move of such additional nodes.

In Chapter 4 we introduce an algorithm that solves the $(k, k-1)$-ruling set problem in anonymous networks under the Gouda daemon. The parallel construction of multiple such ruling sets allows to find a distance-K coloring in an anonymous network. Then we explain how to use this distance-K coloring as identifiers to solve any mending problems on anonymous networks. We do not give any complexity in our work as focused on proving it was possible. Now that we know it

is, it would be interesting to question the complexities of mending problems in anonymous systems.

Finally, in Chapter 6, we introduce a new problem, the Minimum Colored Maximum Matching problem, that extends what had already been done on colored matchings. We show the problem to be NP-hard and hard to approximate within a logarithmic ratio of the size of the graph. However, we do not give an approximation algorithm that would have given an upper bound for the approximation. Searching for such an algorithm would be a natural way to extend this work. We also prove this problem to be $W[2]$-hard with the parameter "size of the solution", but fixed-parameter tractable with the parameter "size of a maximum matching". As this second parameter would be considered a "big" parameter, another natural extension would be to search for a smaller parameter for which MCMM would be FPT.

Bibliography

[1] Noga Alon, László Babai, and Alon Itai. "A fast and simple randomized parallel algorithm for the maximal independent set problem". In: *Journal of Algorithms* 7.4 (1986), pp. 567–583.

[2] Baruch Awerbuch. "Complexity of Network Synchronization". In: *J. ACM* 32.4 (1985), pp. 804–823.

[3] Baruch Awerbuch et al. "Network decomposition and locality in distributed computation". In: *FOCS*. Vol. 30. Citeseer. 1989, pp. 364–369.

[4] Alkida Balliu, Sebastian Brandt, and Dennis Olivetti. "Distributed lower bounds for ruling sets". In: *SIAM Journal on Computing* 51.1 (2022), pp. 70–115.

[5] Alkida Balliu et al. "Local mending". In: *arXiv preprint arXiv:2102.08703* (2021).

[6] Alkida Balliu et al. "Lower Bounds for Maximal Matchings and Maximal Independent Sets". In: *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*. 2019, pp. 481–497.

[7] Alkida Balliu et al. "Lower bounds for maximal matchings and maximal independent sets". In: *Journal of the ACM (JACM)* 68.5 (2021), pp. 1–30.

[8] Leonid Barenboim, Michael Elkin, and Uri Goldenberg. In: 2018, pp. 437–446.

[9] Leonid Barenboim et al. "The locality of distributed symmetry breaking". In: *Journal of the ACM (JACM)* 63.3 (2016), pp. 1–45.

[10] Badreddine Benreguia et al. "Self-stabilizing Algorithm for Maximal Distance-2 Independent Set". In: *CoRR* abs/2101.11126 (2021).

[11] Shimon Bitton et al. "Fully Adaptive Self-Stabilizing Transformer for LCL Problems". In: *arXiv preprint arXiv:2105.09756* (2021).

[12] Sebastian Brandt et al. "LCL problems on grids". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing*. 2017, pp. 101–110.

[13] Sharon Bruckner et al. "Evaluation of ILP-Based Approaches for Partitioning into Colorful Components". In: *Experimental Algorithms*. Ed. by Vincenzo Bonifaci, Camil Demetrescu, and Alberto Marchetti-Spaccamela. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 176–187.

[14] Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. “Derandomizing Local Distributed Algorithms under Bandwidth Restrictions”. In: *Distrib. Comput.* 33.3–4 (2020), pp. 349–366.

[15] Bala G. Chandran and Dorit S. Hochbaum. “Practical and theoretical improvements for bipartite matching using the pseudoflow algorithm”. In: *CoRR* abs/1105.1569 (2011).

[16] J. Cohen et al. “Tropical matchings in vertex-colored graphs”. In: *Electronic Notes in Discrete Mathematics* 62 (2017), pp. 219–224.

[17] Richard Cole and Uzi Vishkin. “Deterministic coin tossing with applications to optimal parallel list ranking”. In: *Information and Control* 70.1 (1986), pp. 32–53.

[18] Alain Cournier, Stephane Devismes, and Vincent Villain. “Snap-Stabilizing PIF and Useless Computations”. In: *Proceedings of the 12th International Conference on Parallel and Distributed Systems - Volume 1*. ICPADS ’06. USA: IEEE Computer Society, 2006, pp. 39–48.

[19] J.-A. Anglès d’Auriac et al. “Tropical dominating sets in vertex-coloured graphs”. In: *Journal of Discrete Algorithms* 48 (2018), pp. 27–41.

[20] Elias Dahlhaus and Marek Karpinski. “A fast parallel algorithm for computing all maximal cliques in a graph and the related problems”. In: *SWAT 88*. Ed. by Rolf Karlsson and Andrzej Lingas. Springer Berlin Heidelberg, 1988, pp. 139–144.

[21] François Delbot, Christian Laforest, and Raksmey Phan. “New approximation algorithms for the vertex cover problem”. In: *International Workshop on Combinatorial Algorithms*. Springer. 2013, pp. 438–442.

[22] François Delbot, Christian Laforest, and Stephane Rovedakis. “Self-stabilizing Algorithms for Connected Vertex Cover and Clique Decomposition Problems”. In: *Principles of Distributed Systems*. Ed. by Marcos K. Aguilera, Leonardo Querzoni, and Marc Shapiro. Springer International Publishing, 2014, pp. 307–322.

[23] Edsger W. Dijkstra. “Self-Stabilizing Systems in Spite of Distributed Control”. In: *Commun. ACM* 17.11 (1974), pp. 643–644.

[24] Edsger W. Dijkstra. “Self-stabilizing systems in spite of distributed control”. In: *Communications of the ACM* 17.11 (1974), pp. 643–644. issn: 0001-0782.

[25] Irit Dinur and David Steurer. "Analytical Approach to Parallel Repetition". In: *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*. STOC '14. New York, New York: ACM, 2014, pp. 624–633.

[26] Shlomi Dolev. *Self-stabilization*. MIT press, 2000.

[27] Shlomi Dolev, Amos Israeli, and Shlomo Moran. "Uniform dynamic self-stabilizing leader election". In: *Distributed Algorithms*. Ed. by Sam Toueg, Paul G. Spirakis, and Lefteris Kirousis. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 167–180.

[28] Rod G. Downey and Michael R. Fellows. "Fixed-Parameter Tractability and Completeness I: Basic Results". In: *SIAM Journal on Computing* 24.4 (1995), pp. 873–921.

[29] Swan Dubois and Sébastien Tixeuil. "A taxonomy of daemons in self-stabilization". In: *arXiv preprint arXiv:1110.0334* (2011).

[30] Jack Edmonds. "Paths, Trees, and Flowers". In: *Canadian Journal of Mathematics* 17 (1965), pp. 449–467.

[31] Michael R. Fellows, Jiong Guo, and Iyad Kanj. "The parameterized complexity of some minimum label problems". In: *Journal of Computer and System Sciences* 76.8 (2010), pp. 727–740.

[32] Xiaofeng Gao et al. "A Novel Approximation for Multi-Hop Connected Clustering Problem in Wireless Networks". In: *IEEE/ACM Trans. Netw.* 25.4 (2017), pp. 2223–2234.

[33] Mohsen Ghaffari. "An Improved Distributed Algorithm for Maximal Independent Set". In: *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '16. Arlington, Virginia: Society for Industrial and Applied Mathematics, 2016, pp. 270–277.

[34] Mohsen Ghaffari, Christoph Grunau, and Václav Rozhoň. "Improved Deterministic Network Decomposition". In: *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '21. Virtual Event, Virginia: Society for Industrial and Applied Mathematics, 2021, pp. 2904–2923.

[35] Mohsen Ghaffari, Fabian Kuhn, and Yannic Maus. "On the complexity of local distributed graph problems". In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. 2017, pp. 784–797.

[36] Wayne Goddard et al. "Self-Stabilizing Protocols for Maximal Matching and Maximal Independent Sets for Ad Hoc Networks". In: 2003.

[37] Mohamed G Gouda. “The theory of weak stabilization”. In: *International Workshop on Self-Stabilizing Systems*. Springer. 2001, pp. 114–123.

[38] Maria Gradinariu and Sebastien Tixeuil. “Conflict Managers for Self-stabilization without Fairness Assumption”. In: *27th International Conference on Distributed Computing Systems (ICDCS ’07)*. 2007, pp. 46–46.

[39] Nabil Guellati and Hamamache Kheddouci. “A Survey on Self-Stabilizing Algorithms for Independence, Domination, Coloring, and Matching in Graphs”. In: *J. Parallel Distrib. Comput.* 70.4 (2010), pp. 406–415.

[40] Stephen Hedetniemi. “Self-Stabilizing Domination Algorithms”. In: 2021, pp. 485–520.

[41] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. “A deterministic almost-tight distributed algorithm for approximating single-source shortest paths”. In: *SIAM Journal on Computing* 50.3 (2019), STOC16–98.

[42] Michiyo Ikeda, Sayaka Kamei, and Hirotsugu Kakugawa. “A Space-Optimal Self-Stabilizing Algorithm for the Maximal Independent Set Problem”. In: (2002).

[43] H. Ishii and H. Kakugawa. “A self-stabilizing algorithm for finding cliques in distributed systems”. In: *21st IEEE Symposium on Reliable Distributed Systems, 2002. Proceedings.* 2002, pp. 390–395.

[44] Esther Jennings and Lenka Motyčková. “A distributed algorithm for finding all maximal cliques in a network graph”. In: *LATIN ’92*. Ed. by Imre Simon. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 281–293.

[45] Colette Johnen and Mohammed Haddad. *Efficient self-stabilizing construction of disjoint MDSs in distance-2 model*. Research Report. Inria Paris, Sorbonne Université ; LaBRI, CNRS UMR 5800 ; LIRIS UMR CNRS 5205, 2021.

[46] Richard M. Karp. “Reducibility among Combinatorial Problems”. In: *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*. Ed. by Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger. Springer US, 1972, pp. 85–103.

[47] Stephan Kunne, Johanne Cohen, and Laurence Pilard. “Self-stabilization and Byzantine Tolerance for Maximal Matching”. In: *Stabilization, Safety, and Security of Distributed Systems*. Ed. by Taisuke Izumi and Petr Kuznetsov. Cham: Springer International Publishing, 2018, pp. 80–95.

[48] Leslie Lamport, Robert Shostak, and Marshall Pease. “The Byzantine Generals Problem”. In: *ACM Trans. Program. Lang. Syst.* 4.3 (1982), pp. 382–401.

[49] Nathan Linial. “Distributive graph algorithms Global solutions from local data”. In: *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*. 1987, pp. 331–335.

[50] Tao Liu, Xiaodong Wang, and Le Zheng. “A Cooperative SWIPT Scheme for Wirelessly Powered Sensor Networks”. In: *IEEE Transactions on Communications* PP (2017), pp. 1–1.

[51] Li Lu, Yunhong Gu, and Robert Grossman. “dMaximalCliques: A Distributed Algorithm for Enumerating All Maximal Cliques and Maximal Clique Distribution”. In: *2010 IEEE International Conference on Data Mining Workshops*. 2010, pp. 1320–1327.

[52] M Luby. “A Simple Parallel Algorithm for the Maximal Independent Set Problem”. In: *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*. STOC ’85. Providence, Rhode Island, USA: Association for Computing Machinery, 1985, pp. 1–10.

[53] S. Micali and V. V. Vazirani. “An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs”. In: *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*. 1980, pp. 17–27.

[54] Moni Naor and Larry Stockmeyer. “What can be computed locally?” In: *SIAM Journal on Computing* 24.6 (1995), pp. 1259–1277.

[55] Alessandro Panconesi and Aravind Srinivasan. “The local nature of Δ-coloring and its algorithmic applications”. In: *Combinatorica* 15.2 (1995), pp. 255–280.

[56] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. USA: Society for Industrial and Applied Mathematics, 2000.

[57] Václav Rozhoň and Mohsen Ghaffari. “Polylogarithmic-Time Deterministic Network Decomposition and Distributed Derandomization”. In: *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 350–363.

[58] Zhengnan Shi, Wayne Goddard, and Stephen T. Hedetniemi. “An anonymous self-stabilizing algorithm for 1-maximal independent set in trees”. In: *Information Processing Letters* 91.2 (2004), pp. 77–83.

[59] Sandeep Shukla, Daniel Rosenkrantz, and S. Ravi. “Observations on self-stabilizing graph algorithms for anonymous networks”. In: *Proceedings of the Second Workshop on Self-Stabilizing Systems* (1995).

[60] Jukka Suomela. “Survey of local algorithms”. In: *ACM Computing Surveys (CSUR)* 45.2 (2013), pp. 1–40.

[61] Hideyuki Tanaka et al. “A Self-stabilizing 1-maximal Independent Set Algorithm”. In: *Journal of Information Processing* 29 (2021), pp. 247–255.

[62] Volker Turau. “Linear Self-Stabilizing Algorithms for the Independent and Dominating Set Problems Using an Unfair Distributed Scheduler”. In: *Inf. Process. Lett.* 103.3 (2007), pp. 88–93.

[63] Volker Turau. “Making Randomized Algorithms Self-stabilizing”. In: 2019, pp. 309–324.

[64] Volker Turau and Christoph Weyer. “Randomized Self-Stabilizing Algorithms for Wireless Sensor Networks”. In: *Proceedings of the First International Conference, and Proceedings of the Third International Conference on New Trends in Network Architectures and Services Conference on Self-Organising Systems*. IWSOS’06/EuroNGI’06. Passau, Germany: Springer-Verlag, 2006, pp. 74–89.

[65] Yanyan Xu et al. “Distributed Maximal Clique Computation”. In: *2014 IEEE International Congress on Big Data*. 2014, pp. 160–167.