Computing models for networks of tiny objects

Nesrine Ouled Abdallah

THÈSE EN COTUTELLE PRÉSENTÉE

POUR OBTENIR LE GRADE DE

DOCTEUR DE L'UNIVERSITÉ DE BORDEAUX ET L'UNIVERSITÉ DE SFAX

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

ÉCOLE DOCTORALE DES SCIENCES ÉCONOMIQUES, GESTION ET INFORMATIQUE

SPÉCIALITÉ : INFORMATIQUE

Par **Nesrine OULED ABDALLAH**

Computing Models for Networks of Tiny Objects

Sous la direction de : Akka ZEMMARI

et de Mohamed JMAIEL

Date de soutenance : 22 Mai 2017

Devant la commission d'examen composée de :

Olivier Flauzac	Professeur, Université de Reims Champagne-Ardenne	Rapporteur (Président)
Vlady Ravelomanana	Professeur, Université Paris-Diderot	Rapporteur
Mohamed Jemni	Professeur, Université de Tunis	Examinateur
Mohamed Jmaiel	Professeur, Université de Sfax	Directeur
Akka Zemmari	Maître de conférences HDR, Université de Bordeaux	Directeur
Mohamed Mosbah	Professeur, Bordeaux INP	Co-directeur

– 2017 –

Titre **Modèles de calcul pour les réseaux d'objets à capacité restreinte**

Résumé Dans cette thèse, nous nous intéressons aux modèles de calcul dans les réseaux d'objets à capacité restreinte, tels que les réseaux de capteurs sans fil. Nous nous focalisons sur les protocoles de population proposés par Angluin et al. Dans ce modèle, les objets sont représentés par des agents à états finis, passivement mobiles, communiquant entre paires et formant un réseau asynchrone et anonyme. Nous présentons deux études comparatives qui nous permettent par la suite de proposer une approche établissant le lien des protocoles de population avec deux autres modèles : le modèle des tâches avec les systèmes de réécritures de graphes, et le modèle asynchrone et anonyme d'échange de messages. Nous passons ensuite au problème d'ordonnancement dans les protocoles de population. Nous proposons un nouvel ordonnanceur probabiliste, $1-$central, basé sur les rendez-vous randomisés et appelé HS Scheduler. Contrairement aux autres ordonnanceurs, il est permet à plus d'une paire de communiquer à la fois. Nous prouvons qu'il est équitable avec probabilité 1. Nous analysons par la suite les temps de stabilisation de certains protocoles s'exécutant sous le Random Scheduler ou le HS Scheduler et sur différentes topologies du graphe d'interaction. Nous prouvons que le HS Scheduler est équivalent en temps au Random Scheduler quand le graphe d'interaction est complet mais qu'il permet une stabilisation plus rapide quand le graphe est aléatoire. Par la suite, nous proposons un autre ordonnanceur qui prend en considération les états des agents et permet d'introduire la terminaison à certains protocoles : le Prorotol Aware HS Scheduler. Nous prouvons qu'il est équitable avec probabilité 1. Nous faisons l'analyse des temps de stabilisation de certains protocoles s'exécutant sous cet ordonnanceur en considérant différentes topologies du graphe d'interaction. Finalement, nous implémentons et simulons sur ViSiDiA l'ensemble des scénarios étudiés et validons nos résultats théoriques.

Mots-clés Objet à capacité restreinte, Protocoles de populations, Modèles de calcul, Ordonnanceur probabiliste, Rendez-vous randomisés, Équité.

Title **Computing Models for Networks of Tiny Objects**

Abstract In this work, we consider computing models for networks of tiny objects such as wireless sensor networks. We focus on the population protocols, a pairwise computational model introduced by Angluin et al. where the tiny objects are represented by anonymous, passively mobile, finite state agents forming asynchronous networks. We establish two comparative studies between the population protocol model (and its extensions) and the two following ones: tasks with graph relabeling systems, and anonymous asynchronous message passing. These studies aim to establish possible mappings between the population protocols and these two models. We then focus on the scheduling of the pairwise interactions in population protocols. We propose the HS Scheduler, a new probabilistic $1-$central scheduler based on randomized handshakes. Compared to the existing schedulers, this scheduler allows to more than one pair of agents to communicate simultaneously. We prove that this scheduler is fair with probability 1. We thereafter present analyses of the complexity of the stabilization time of some protocols running under the scheduling of the Random Scheduler and the HS Scheduler, and over different topologies of the interaction graph. We prove that these two schedulers are time equivalent with

respect to these protocols when the interaction graph is complete, however computations under the HS Scheduler stabilize faster when the interaction graph is random. We then introduce the Protocol Aware HS Scheduler, a slightly modified version of the HS Scheduler that takes into account the states of the agents and allows termination in some protocols. We also prove that this scheduler is fair with probability 1. We present analyses of the time complexity of some protocols running under the scheduling of the Protocol Aware HS Scheduler and over different structures of the interaction graph. We implement the different scenarios in ViSiDiA, and validate through simulations our theoretical results.

Keywords Tiny objects, Population protocols, Computing models, Probabilistic scheduler, Randomized handshakes, Fairness.

Laboratoire d'accueil Laboratoire Bordelais de Recherche en Informatique (LaBRI)-UMR 5800. Domaine universitaire, 351, cours de la Libération, 33405 Talence

Acknowledgments

I would like to thank my advisers Professor Akka Zemmari and Professor Mohamed Jmaiel, and my co-adviser Professor Mohamed Mosbah for introducing me to this research field, for welcoming me in their teams and for their directives, support and guidance.

I would also like to thank my committee members especially Professor Olivier Flauzac and Professor Vlady Ravelomanana for having accepted to review my thesis, for their presence at the defense and for insightful comments and encouragement.

Many thanks go to all the administrative staff especially: Maïté, Céline and Sylvie.

Also, many thanks go to all my friends and colleagues. A special thank you to all the girls from the lunch group: Lorijn, Claire, Thao, Thanh, Samah, Amal and Pauline for all of the nice moments we shared.

I am especially grateful to my friends Omessaad, David, and Mortharia for supporting me during tough times and helping me carrying on.

Finally, I think that words cannot express how grateful I am to my parents for their endless love and sacrifices. Thank you for believing in me and encouraging me to go further. I am also deeply grateful to my husband, my sister, my brothers, and all our family for all their love and comfort and for bringing joy to my soul even though we are always far away. I am thankful to all persons who are close to my heart.

Thanks go also to all other people that contributed in any way to the success of this thesis.

Contents

List of Figures

List of Tables

Résumé étendu

Les réseaux d'objets à capacités restreintes, tels que les réseaux de capteurs sans fil, sont des systèmes distribués asynchrones formés de petites entités (aussi appelés objets, agents, ou équipement) dont les ressources : énergie, mémoire et puissance de calcul sont limitées. Ces entités sont capables de communiquer via des liens sans fil. Elles peuvent aussi être passivement mobiles n'ayant aucun contrôle de leurs mouvements (attachées à des objets mobiles par exemple). A cause de la contrainte sur l'espace mémoire, ces entités sont souvent supposées comme étant anonymes. Une fois le réseau est déployé, ces objets vont devoir communiquer et coopérer pour atteindre un objectif global. Cependant, comment arriver à faire les calculs nécessaires pour atteindre cet objectif malgré les ressources limitées du réseau?

Dans le cadre de ce travail, nous nous sommes intéressés aux protocoles de populations [5, 4]. C'est un modèle de calcul entre paires proposé par Angluin et al. pour de tels réseaux. Dans ce modèle, les agents formant le réseau sont considérés comme des automates finis sans identifiants uniques. Ils sont identiquement programmés et forment ainsi une population de taille finie mais non bornée. Un protocole de population calcule un prédicat ou une fonction des entrées distribuées que ces agents ont reçu de leur environnement. Par exemple, un protocole de population s'exécutant dans une population d'oiseaux pourrait être conçu de telle manière qu'il soit capable de décider si le nombre d'oiseaux qui sont malades atteint un certain seuil. Les calculs dans ce modèle sont basés sur des interactions entre paires. Deux agents seront capables de communiquer chaque fois qu'ils seront à proximité l'un de l'autre et partageront ainsi un lien de communication. Lors de cette communication, un agent jouera le rôle d'initiateur tandis que le deuxième jouera celui du receveur. Ces deux agents échangeront leurs états actuels. Par la suite, chacun mettra à jour son état selon son rôle, son état actuel, l'état reçu, et les règles de transition du protocole. Toutes les interactions possibles entre paires d'agent sont représentées par les arêtes dirigées du graphe d'interactions dont les nœuds représentent les agents de la population.

Les protocoles de population ne s'arrêtent pas mais plutôt se stabilisent. Cette stabilisation a lieu lorsque les sorties des agents convergent, après un temps fini de calcul, vers le résultat correct. Cette stabilisation est une propriété globale du système que les agents ne détectent souvent pas. En effet, la connaissance d'un agent est limitée à son état local. Par conséquent, même si le protocole se stabilise, les agents continuent à interagir.

La puissance de calcul du modèle de protocole de populations a été caractérisée comme étant exactement l'arithmétique de Presburger [6, 10]. La plupart des travaux qui se sont intéressés à ce modèle ont essayé d'améliorer sa puissance de calcul. Par exemple, Chatzi-

giannakis et al. ont proposé le modèle des mediated protocoles de population qui permet d'attribuer des états non seulement aux agents de la population mais aussi aux liens de communications qui les relient [29]. Un autre exemple d'extension des protocoles de population est le modèle PALOMA qui permet aux agents d'avoir, au lieu d'une mémoire constante, un espace mémoire plus important [27]. Contrairement à tous ces travaux, nous nous intéressons à un aspect autre que la puissance de calcul, qui est le modèle d'interaction dans les protocoles de population.

En effet, une interaction entre paire d'agents consiste en un échange abstrait, simultané et bidirectionnel des états de cette paire communicantes. Un concept similaire de communication abstraite est utilisé dans le modèle de tâche avec les systèmes de réécritures de graphes [48, 47]. Dans ce modèle, un système distribué est représenté par un graphe étiqueté, et un algorithme distribué s'exécutant sur ce système est codé par une séquence de réétiquetages du graphe correspondant. Ce modèle est un modèle de calculs locaux, tout comme les protocoles de population. Nous proposons donc d'explorer dans cette thèse les correspondances qui existent entre ces deux modèles ainsi que la possibilité de plonger chaque modèle dans l'autre.

Les protocoles de population ont été conçus pour des réseaux tels que les réseaux de capteurs sans fil, donc adopter l'hypothèse de communication abstraite est une hypothèse assez forte. Ainsi, décrire cet échange bidirectionnel entre paire à travers un modèle de communication moins théorique, basé sur un minimum d'hypothèses supplémentaires, serait assez intéressant. Par conséquent, représenter une population par un système distribué anonyme et asynchrone basé sur échange de messages avec numérotation de ports s'avère une bonne alternative. Un protocole de population pourrait ainsi être décrit par un algorithme distribué basé sur échange de messages et tournant sur ce système.

Dans un autre volet de ce travail, nous nous intéressons à l'ordonnancement des interactions dans les protocoles de population. L'ordonnanceur est l'entité responsable du choix des paires d'agents communicantes : il choisit des paires d'agents adjacents dans le graphe d'interaction. Pour garantir une exécution équitable des protocoles, un ordonnanceur doit satisfaire une propriété d'équité. Un ordonnanceur conçu pour les protocoles de population doit être aussi probabiliste pour décrire l'imprédictibilité des interactions causée par la mobilité et l'asynchronisme des agents. Tous les ordonnanceurs déjà proposés pour les protocoles de population sont des ordonnanceurs centraux (aussi appelés séquentiels) : à chaque étape de calcul du protocole, l'ordonnanceur n'autorise qu'à une seule paire d'agents de communiquer. Les populations d'agents sont des systèmes distribués où il n'y a aucun contrôle centralisé. Ainsi, un ordonnanceur distribué serait plus adéquat dans ce contexte vu qu'il arrive à capturer l'aspect concurrent des interactions : il peut permettre à plus d'une paire d'agents de communiquer simultanément. Nous proposons donc un nouvel ordonnanceur pour les protocoles de population, qui est probabiliste, équitable et distribué.

L'équité des ordonnanceurs ne garantit pas leurs équivalences [25]. Nous proposons donc d'explorer l'équivalence en temps de l'ordonnanceur que nous proposons avec le Random Scheduler, lordonnanceur proposé par Angluin et al. [5]. Nous pensons aussi que permettre à cet ordonnanceur distribué proposé de prendre en compte les états des agents

permettrait d'introduire la notion de terminaison aux protocoles de population. Nous pensons même que ceci pourrait permettre une stabilisation plus rapide des protocoles : les agents ayant atteints un certain état ne seront plus sélectionnés par l'ordonnanceur.

Contributions

Nous présentons dans ce qui suit un résumé des principales contributions de ce travail.

1. **Plongement du modèle des protocoles de populations dans le modèle de tâches avec les systèmes de réécritures de graphes et vice versa.** Nous présentons une étude comparative des (mediated) protocoles de populations et les tâches avec les systèmes de réécritures de graphes. Étant donné que ces deux modèles utilisent des communications abstraites entre paires, nous étudions les similarités, ainsi que les différences qui existent entre eux. Basé sur cette étude comparative, nous proposons une approche qui permet le plongement d'un modèle dans l'autre. Nous prouvons que le calcul d'un (mediated) protocole de population peut toujours être exprimé comme la réalisation d'une tâche avec un système de réécriture de graphe. Par contre, l'autre sens n'est pas toujours vrai. En effet, seule la réalisation d'une tâche dont les règles de réécritures sont limitées à des règles $LC0$ sans contextes interdits ou priorités et dont les étiquettes sont de taille constante peut être décrite comme le calcul d'un (mediated) protocole de population.

2. **Plongement du modèle des protocoles de population dans le modèle des algorithmes distribués anonymes et asynchrones basés sur échange de messages.** Nous établissons une étude comparative du modèle des protocoles de population (et qui est valable pour tout modèle étendant les protocoles de population tout en préservant le principe de communication entre paires) avec le modèle anonyme et asynchrone d'échange de messages. En effet, nous considérons qu'utiliser un modèle de communication abstraite dans des réseaux d'objets connectés est une hypothèse assez forte et théorique. Nous proposons ainsi une approche qui permet de traduire le calcul d'un protocole de population en un algorithme distribué basé sur échange de messages dans un système asynchrone et anonyme avec numérotation de ports. La numérotation de ports est la connaissance minimale et nécessaire pour pouvoir passer du modèle de communication abstraite à un modèle de communication explicite. L'échange simultané et bidirectionnel entre une paire d'agents dans un protocole de population est ainsi traduite en deux phases dans l'algorithme distribué correspondant : une phase de synchronisation, puis une phase d'envoi et de réception simultanés.

   En combinant ces deux approches avec des travaux déjà existants, nous établissons une passerelle entre ces trois différents modèles qui offre la possibilité de décrire chaque modèle à travers un autre.

3. **L'algorithme du Random Scheduler.** Basé sur la contribution précédente, nous proposons un algorithme distribué et anonyme basé sur échange de messages qui encode l'ordonnanceur Random Scheduler. Cet algorithme est conçu pour le cas où le graphe d'interaction est un graphe complet. Cette contribution est une amélioration d'un résultat que nous avons publié dans [60].

4. **Le Handshake Scheduler.** Nous présentons un nouvel ordonnanceur probabiliste basé sur les rendez-vous randomisés. Nous l'appelons le Handshake Scheduler et le notons le *HS Scheduler*. Contrairement aux ordonnanceurs déjà proposés pour les protocoles de population, ce dernier n'est pas séquentiel mais plutôt $1-$central. Par conséquent, il peut permettre à plus d'une paire d'agents d'interagir en même temps tant qu'elles sont disjointes. Inspirés des algorithmes de rendez-vous randomisés qui existent dans la littérature, nous proposons un algorithme pour le *HS Scheduler*. Nous faisons l'analyse de cet algorithme. Cette analyse nous sert par la suite à prouver que le *HS Scheduler* est un ordonnanceur probabiliste, consistent et équitable avec probabilité 1. Cette contribuant a fait l'objet d'un papier [61].

5. **Equivalence en Temps: le Random Scheduler et le *HS Scheduler*.** Nous étudions l'équivalence en temps du Random Scheduler et du *HS Scheduler*, par rapport aux protocoles suivants : le *OU*, l'Élection d'un Leader, et le Couplage Maximal. Nous présentons des analyses théoriques des temps de stabilisation de ces protocoles en considérant deux scénarios d'ordonnancement. Dans le premier scénario, c'est le Random Scheduler qui est utilisé. Dans le deuxième, c'est le *HS Scheduler* qui est utilisé. Nous considérons aussi deux différents types de graphe d'interaction : le graphe complet, et le graphe aléatoire. Nous prouvons que le Randam Scheduler et le *HS Scheduler* sont équivalents en temps par rapport aux protocoles étudiés lorsqu'ils sont exécutés sur des graphes d'interaction complets. Par contre, lorsque le graphe d'interaction est un graphe aléatoire, le *HS Scheduler* permet une stabilisation plus rapide de ces protocoles.

6. **L'ordonnaceur Protocol Aware HS Scheduler.** Nous proposons un nouveau modèle de calcul appelé protocoles de population avec une certaine détection locale de la terminaison. Ce modèle est une extension d'une sous classe des protocoles de population (et des modèles qui les étendent) avec une fonction de détection de terminaison locale qui permet à un agent de détecter si son état est final. Par la suite, nous présentons le Protocol Aware HS Scheduler, un nouvel ordonnanceur distribué et probabiliste qui prend en compte le protocole exécuté. Il s'agit d'une version ajustée du *HS Scheduler* de telle manière qu'il permet aux agents dont les états sont finaux de ne plus participer au calcul du protocole, et ainsi terminer. Ainsi, ces petites entités pourront éviter de dissiper leurs ressources limitées pour des interactions inutiles. Nous prouvons que cet ordonnanceur est équitable avec probabilité 1. Nous présentons aussi des calculs des bornes supérieures des temps de stabilisation des protocoles suivants : Élection d'un Leader et Couplage Maximal, s'exécutants sous cet ordonnanceur. Nous prouvons que cet ordannanceur permet une stabilisation plus rapide de ces protocoles comparé au Random Scheduler et au *HS Scheduler*. Une partie de cette contribution est apparue dans [61].

   Une partie de ces résultats ont fait l'objet d'un article que nous avons soumis au journal "Mathematical Structures in Computer Science", et dont nous attendons encore la réponse.

Organization du manuscrit

Nous commençons ce travail par un chapitre préliminaire (Chapitre 1) pour introduire quelques concepts et outils de base reliés aux systèmes et algorithmes distribués, à la théorie des graphes, à la théorie de la probabilité, et aux mathématiques. Ce chapitre facilitera ainsi la compréhension de ce qui suivra dans ce travail.

Par la suite, dans le Chapitre 2, nous introduisons le modèle des protocoles de population proposé par Angluin et al. Nous présentons une définition formelle de ce modèle ainsi que les résultats obtenus par Angluin et al. concernant sa puissance de calcul. Nous donnons ensuite un aperçu des variantes de ce modèle tel que le One Way protocoles de population qui restreint l'échange bidirectionnel entre les paires d'agents en un échange unidirectionnel. Nous présentons aussi les modèles qui ont étendu les protocoles de population en attribuant des états aux arêtes, ou en allouant plus de mémoire aux agents, *etc.*

Le Chapitre 3 est constitué de deux parties. Dans la première, nous présentons le modèle des tâches avec les systèmes de réécritures de graphes. Ce modèle, tout comme les protocoles de population, est un modèle de calculs locaux basé sur des communications abstraites. Nous établissons par la suite une étude comparative qui nous permet de définir une approche décrivant la possibilité du plongement du modèle des protocoles de population dans le modèle des tâches avec les systèmes de réécritures de graphes et vice versa.

Dans la deuxième partie de ce chapitre, nous présentons le modèle anonyme et asynchrone d'échange de messages. Nous détaillons les méthodes de numérotation de ports qui existent dans la littérature. Par la suite, nous établissons une étude comparative grâce à laquelle nous proposons une approche permettant d'exprimer un protocole de population comme étant un algorithme distribué s'exécutant dans un système anonyme et asynchrone basé sur échange de messages avec numérotation de ports.

Nous concluons ce chapitre en établissant une passerelle entre les trois modèles suivants : protocoles de population, tâches avec systèmes de réécritures de graphes, et échange de messages anonyme et asynchrone.

Dans le Chapitre 4, nous nous intéressons à l'ordonnancement des interactions dans les systèmes distribués, et plus précisément dans les protocoles de population. Nous présentons un aperçu des ordonnanceurs qui ont déjà été proposés ainsi que les différentes définitions de la notion d'équité. Nous proposons un algorithme pour l'ordonnanceur Random Scheduler. Par la suite, nous introduisons le *HS Scheduler*. Étant donné qu'il est basé sur les rendez-vous randomisés, nous étudions les algorithmes de rendez-vous randomisés de la littérature. Inspirés de ces algorithmes, nous présentons un algorithme qui encode le *HS Scheduler* dont nous faisons l'analyse pour prouver l'équité de cet ordonnanceur.

Nous comparons l'ordonnanceur que nous proposons avec le Random Scheduler. Tous deux sont équitables, cependant sont-ils équivalents en temps? Nous explorons cette équivalence dans le Chapitre 5 par rapport à trois protocoles différents s'exécutant sous ces ordonnanceurs en considérant différentes topologies du graphe d'interaction. Les résultats

établis sont comme décrit dans la contribution 5. Nous clôturons ce chapitre avec un tableau représentant les résultats de l'analyse du temps de stabilisation de chaque protocole pour chacun des scénarios d'ordonnancement.

Dans le Chapitre 6, nous présentons le modèle des halting protocoles de population proposé par Michail et al [55]. Nous nous focalisons sur le protocole Élection d'un Leader avec détecteur d'absence. Nous étudions le temps de sa stabilisation quand son calcul est effectué sous l'ordonnancement du Random Scheduler, et puis sous l'ordonnancement du *HS Scheduler*.

Nous proposons ensuite une extension d'une sous classe des protocoles de population (et les modèles qui les étendent) avec une détection de la terminaison locale qui permet à chaque agent de détecter s'il a atteint un état final. Un état final est un état irréversible qui n'apparaît pas dans les règles de transition effectives. Nous présentons par la suite notre contribution 6 relié au Protocol Aware HS Scheduler. Nous clôturons ce chapitre avec un tableau représentant les résultats obtenus concernant les temps de stabilisation des protocoles étudiés pour les différents scénarios d'ordonnancement.

Dans le Chapitre 7, nous présentons la plate-forme ViSiDiA implémentée en Java et conçue pour implémenter, simuler et visualiser des algorithmes distribués. Nous avons enrichi cette plate-forme en ajoutant un module qui permet la génération automatique de différents types de graphes. Grâce à ViSiDiA, nous illustrons les différentes approches de mapping que nous avons établies dans le Chapitre 3. Nous implémentons sous cette plate-forme tous les ordonnanceurs proposés ainsi que tous les protocoles étudiés. Puis, nous procédons la simulations de tous les scénarios étudiés. Ceci nous permet de valider tous les résultats théoriques obtenus tout au long de ce travail.

Finalement, nous clôturons ce travail avec une conclusion et quelques perspectives.

Introduction

Networks of tiny objects, such as sensor networks, are asynchronous distributed systems formed by small agents (also called objects, entities or devices) with limited resources, memory and computational power and that are able to establish wireless communications. Due to the constrained memory space, agents are usually supposed to be anonymous. They can also be passively mobile having no control of their movement. Once the network is deployed, these objects should communicate and cooperate to reach a global goal. Yet, how to compute in such networks with restricted resources and capacities ?

In this work, we focus on the population protocols, a pairwise computational model introduced by Angluin et al. and designed for such networks [5, 4]. In this model, the agents of the network are considered as finite automata with no unique identifiers. They are identically programmed and form a population of a finite but unbounded size. A population protocol computes a given function or predicate of the distributed inputs of the agents, received from their environment. For example, a population protocol running in a population of birds can decide if a given threshold related to the number of sick birds is reached. Computations in this model are based on pairwise interactions. Two agents are able to communicate whenever they come sufficiently close to each other and share a communication link. A communicating pair of agents consists of an initiator and a responder which breaks the symmetry. The two agents exchange their respective current states. Then, each agent updates its state accordingly and with respect to its role and to the transition rules of the protocol. All possible interactions between pair of agents are represented by the directed edges of the interaction graph whose vertices are the agents forming the population.

Population protocols do not halt but only stabilize. Stabilization is reached when the outputs of the agents converge after some finite time to a correct value. Stabilization is a global property of the population that agents usually do not detect. In fact, the only knowledge that an agent may have is a local one consisting of its own state. Consequently, even when the protocol stabilizes, the pairwise interactions do not stop.

The computational power of the population protocol model was characterized to be exactly the Presburger arithmetic [6, 10]. Most of the studies that were interested in this model dealt with the enhancement of its computational power. They proposed to extend this model either by adding states to the communication links which is the case of the mediated population protocols [29], or by allowing more memory to the agents such as the PALOMA model [27], *etc*, in order to obtain more powerful models. However, in this work, we focus on another aspect of this model: the interaction model of the population protocols.

A pairwise interaction in a population protocol is represented as an abstract simultane-

ous bidirectional exchange between the communicating pair of agents. A similar concept of abstract pairwise communication is used in the model of tasks with graph relabeling systems [48, 47]. This model represents a distributed system as a labeled graph, and a distributed algorithm executed in this system as a sequence of relabellings of the corresponding graph. This model belongs, as the population protocols, to local computations systems. In this thesis, we propose to investigate the correspondences between these models and the possibility of establishing a bridge between both of them.

Assuming abstract communication in a computational model designed for networks such as sensor network is a theoretical hypothesis. Thus, it becomes interesting to describe the pairwise bidirectional exchange through a less theoretical communication model using only minimalistic assumptions. Consequently, representing a population as an anonymous asynchronous system based on message passing and with port numbering is a suitable choice. A population protocol corresponds consequently to a distributed algorithm based on message passing running in this system.

We also focus on another aspect of the execution of a population protocol which is the scheduling. A scheduler is the entity responsible for the choice of the interacting pairs: it picks pair of agents that are adjacent in the interaction graph. A scheduler should satisfy the fairness property to guarantee a fair execution of the protocols. A scheduler designed for population protocols should also be probabilistic to describe the unpredictable interaction pattern in the population caused by the mobility and the asynchrony of the agents. All the schedulers proposed for the population protocols are probabilistic central (also called sequential) schedulers: at each computation step of a protocol, they allow to only one pair of agents to communicate. As populations are distributed systems, where there is no centralized control, a distributed scheduler is more fitted to capture the concurrent interactions as it is able to pick more than one interacting pair at each computation step. We thus propose to introduce a new probabilistic fair scheduler for population protocols which is distributed.

The fairness of the schedulers does not guarantee their equivalence [25]. Hence, we propose to investigate how the proposed distributed scheduler affects the stabilization time of some protocols compared to the Random Scheduler of Angluin et al. [5]. We also consider that enabling the proposed scheduler to take into account the states of the agents may allow introducing termination in population protocols. We even think that this may lead to faster computations of some protocols as agents that reach some specific states are not picked any more for interactions.

Contributions

The main contributions of this thesis are summarized as follows.

1. **Mapping Population Protocols to Tasks with Graph Relabeling System and Vice versa.** We present a comparative study involving the (mediated) population protocols and tasks with graph relabeling systems. As both use abstract and pairwise communication, we investigate the existing similarities, and also the

existing differences, between these two models. Based on this comparative study, we propose a mapping approach. We prove that the computation of a (mediated) population protocols can always be mapped to a realization of a task with a graph relabeling system. However, the reverse is not true. In fact, only a realization of a task with a graph relabeling system whose relabeling rules are limited to $LC0$ computations with no forbidden contexts or priorities and whose labellings are of constant size can be mapped to a computation of (mediated) population protocols.

2. **Mapping Population Protocols to Distributed Algorithms in Anonymous Asynchronous Systems based on Message Passing.** We establish a comparative study of the population protocol model (that holds for any model extending as long as it preserves the pairwise interaction concept) and the anonymous asynchronous message passing model. In fact, we consider that using an abstract communication model for such networks of tiny objects is a theoretical assumption. Accordingly, we propose a mapping approach that describes the computation of a population protocol as a distributed algorithm based on message passing in an anonymous asynchronous system with port numbering. Port numbering is the minimalistic knowledge that we should afford to go from abstract communication to explicit one. The bidirectional and simultaneous pairwise interaction in the protocol is described by two phases in the corresponding algorithm: a synchronization phase, then a simultaneous send and receive phase.

   By combining these two mapping approaches with some existing works, we establish a bridge between these three models that offers the possibility of describing each model with another.

3. **The Random Scheduler Algorithm.** Based on the previous contribution, we propose an anonymous distributed algorithm with message passing that encodes the Random Scheduler. This algorithm is designed to run in complete interaction graphs. This contribution is an enhancement of a result that we published in [60].

4. **The Handshake Scheduler.** We introduce a new probabilistic scheduler based on randomized handshakes. We call it the Handshake Scheduler and also denote it the *HS Scheduler*. Unlike the proposed schedulers for population protocols that are sequential, this scheduler is $1-$central. Consequently, it allows to more than only one interaction to take place at a computation step as long as the interacting pairs of agents are disjoint. Inspired from the existing randomized rendezvous algorithms, we propose an algorithm for this *HS Scheduler*. We present the analysis of this algorithm based on which we prove that the *HS Scheduler* is a probabilistic consistent fair scheduler with probability 1. This contribution appeared in [61].

5. **Time Equivalence: The Random Scheduler and the *HS Scheduler*.** We investigate the time equivalence of the Random Scheduler and the *HS Scheduler* with respect to three protocols: the OR population protocol, the Leader Election and the Maximal Matching. We present theoretical analyses of the stabilization time of these protocols when assuming two scenarios of scheduling: the first under the Random Scheduler, and the second under the *HS Scheduler*. We also consider different topologies of the interaction graph: either a complete graph, or a random graph. We prove that the Random Scheduler and the *HS Scheduler* are time

equivalent with respect to these three protocols when running over a complete interaction graph. However, in a random interaction graph, we prove that each of these protocols stabilizes faster when running under the *HS Scheduler*.

6. **The Protocol Aware HS Scheduler.** We propose the model of population protocols with some local termination detection. This model extends a subclass of the populations protocols (and of the models extending them) with a local termination detection function enabling an agent to detect that its state is final. We then propose a new probabilistic distributed scheduler which is protocol aware: the Protocol Aware HS Scheduler. This is a slightly modified version of the *HS Scheduler* that allows agents with final states to stop participating in the protocol's computation and terminate. Consequently, the tiny agents avoid consuming their limited resources in ineffective interactions. We prove that this scheduler is fair with probability 1. We also present some upper bounds of the stabilization time of the Leader Election and the Maximal Matching protocols running under this scheduler. We prove that the computations of these protocols under this scheduler stabilize faster compared to a scheduling under the Random Scheduler or the *HS Scheduler*. A part of this contribution appeared in [61].

   Some of these results were submitted to the journal of Mathematical Structures in Computer Science and are still under review.

Organization of the Document

We start this dissertation by a preliminary section (Chapter 1) presenting some basic tools and concepts related to: distributed systems and algorithms, graph theory, mathematics and probability theory, that the reader may need to better understand what will follow in this work.

Then, in Chapter 2, we introduce the population protocol model as proposed by Angluin et al. We give the formal definition of this model and present the result of Angluin et al. related to its computational power. We then give an overview of the existing variants of this model , such as the One Way population protocols that restrict the bidirectional pairwise interactions to unidirectional ones. We also present the models that extend the population protocols, by assigning states to the edge, or by allowing more memory space for the agents, *etc*.

Chapter 3 consists of two parts. In the first part, we present the model of tasks with graph relabeling systems which, as the population protocols, belongs to local computation systems and uses abstract communications. We then establish the comparative study and the mapping approach of population protocols to tasks with graph relabeling systems and vice versa. It describes the possibility, or not, of representing each of these two models with one another.

Then, in the second part of this chapter, we present the anonymous asynchronous message passing model. We describe the existing methods of port numbering in literature. Then we introduce the comparative study and the mapping approach of a population protocol to a distributed algorithm running in an anonymous asynchronous system based

on message passing with port numbering.

We conclude this chapter by establishing a bridge between these three models: population protocols, tasks with graph relabeling systems and anonymous asynchronous message passing.

In Chapter 4, we focus on the scheduling of the interactions in distributed systems, and more precisely in population protocols. We give an overview of the existing schedulers as well as the fairness conditions presented in this context of population protocols. We propose an algorithm that encodes the Random Scheduler in a complete graph. We then introduce the *HS Scheduler*. As it is based on randomized handshakes, we give an overview of the existing randomized handshake algorithms. We then propose the *HS Scheduler* algorithm that we analyze to prove the fairness of this scheduler.

We compare our proposed scheduler with the Random Scheduler. Both are protocol oblivious fair schedulers. However, are they time equivalent? We investigate this equivalence in Chapter 5 with respect to three different protocols running under these two schedulers over different interaction graph structures. The established results are as described in contribution 5. We conclude this chapter with a table representing the result of the analysis of the stabilization time of each protocol for each scheduling scenario.

In Chapter 6, we present the model of halting population protocols introduced by Michail et al [55]. We focus on the halting Leader Election protocol with absence detector. We study its stabilization time when its computation takes place under the Random Scheduler, and then under the *HS Scheduler*.

We then propose to extend a subclass of population protocols (and the models extending it), with a local termination detection that enables each agent to detect if it reached a final state. A final state is an irreversible state that does not appear in effective transition rules. We then introduce contribution 6 related to the Protocol Aware HS Scheduler. We conclude this chapter with a table representing the established results concerning the stabilization time of the studied protocols in different scheduling scenarios.

We present in Chapter 7 the ViSiDiA platform which is a Java framework designed to implement, simulate and visualize distributed algorithms. We enhanced this platform by adding the possibility of automatically generating graphs. Thanks to this platform, we illustrate the mapping approaches we established in Chapter 3. We implement all the proposed schedulers and the studied protocols in this platform. We proceed on simulating all the studied scenarios. This enables us to validate all the theoretical results obtained during this work through simulations.

Finally, we conclude this dissertation and present some directions for possible future works.

Chapter 1

Preliminaries

This chapter is an introduction to some basic concepts useful for a better understanding of what is going to be introduced later in this document. In fact, our focus will basically be on population protocols which can be considered as distributed algorithms that run over distributed systems. We hence give an overview of such distributed systems and algorithms. As distributed systems are usually represented by graphs, we introduce some basic concepts of graph theory. We will also proceed all along this work, on analyzing some protocols under different probabilistic scheduling. We do thus need some mathematical and probability theory tools that we also define in this chapter.

1.1 Distributed Systems and Algorithms

1.1.1 Distributed Systems

According to Leslie Lamport quotes, "a distributed system is one in which the failure of a computer you did not even know existed can render your own computer unusable". More formally, according to Tel in [67], a distributed system can be described as follows:

Definition 1.1. (**Distributed System**) A distributed system is an interconnected collection of autonomous computers, processes, or processors.

These computers, processes, or processors are also called the nodes of the distributed system. Tel considers as autonomous nodes those that have their own private control [67]. In other words, each process should have its own local memory, local program, and local time (no shared global time). To be interconnected, nodes must be able to communicate and exchange information.

There are three basic communication models for distributed systems: the shared memory, the message passing and the local computations. Further details concerning these two latter models will be provided later in this work.

Each process in the distributed system has a set of variables that forms its state, also called local state. Whatever the chosen communication model is, an action is the result of an interaction between processes leading to a modification of at least one of their states and consequently influencing the global state of the system which can be defined as follows:

Definition 1.2. (**Configuration (System's Global State)**) The set of local states of all the processes of the system forms its global state. A global state of a system is also called a configuration.

A distributed system can be either synchronous or asynchronous.

Definition 1.3. (**Synchronous vs Asynchronous Systems**) In the synchronous model, the processors of the distributed system execute in lockstep: the execution is partitioned into rounds.

However, in case of asynchronous systems, there in no upper bound of the time separating two consecutive computation steps of a processor. Processors run with different speeds.

A distributed system can be anonymous, which is:

Definition 1.4. (**Anonymous Distributed System**) An anonymous system is a system where no unique identities are available to distinguish the different processes.

Anonymity in distributed systems implies symmetry. In fact, processors may play the same role and become identical: there is no way to distinguish them [44].

1.1.2 Algorithms

Distributed Algorithms

Definition 1.5. (**Local Algorithm**) A local algorithm of a process is an algorithm that runs independently of the number of nodes in the network and whose output is a function of the input of the process as well as the input available within a constant-radius of its neighborhood [66, 65].

Definition 1.6. (**Distributed Algorithm**) A distributed algorithm of a collection $\mathbb{P} = \{p_1, \ldots, p_n\}$ of processes is a collection of local algorithms, one for each process in $\mathbb{P}$ [67].

Definition 1.7. (**Deterministic vs Non Deterministic Algorithms**) An algorithm is deterministic, if starting from a given input, it always produces the same output.

This is not the case for a non deterministic algorithm which, for a given input, may have more than one possible output.

Randomized Algorithms

Lynch [49] as well as Fisher and Jiang [35] provided some impossibility results in distributed computing that can be caused by the determinism of the solutions, the symmetry of the processes, etc. However, introducing randomization to the solutions and breaking the symmetry can make these distributed problems solvable [50].

According to Motwani et al [57], a randomized algorithm can be defined as follows:

Definition 1.8. (**Randomized Algorithm**) A randomized algorithm is an algorithm that uses random numbers to influence the choices it makes during its computation. Thus, its behavior varies from one execution to another even with a fixed input.

Tel also defines a randomized algorithm as a probabilistic (randomized) process that can be modeled by a process that flips a coin at each step it executes [67]. Thus the possible next steps depend on the state of the process as well as on the outcome of the coin flip.

Definition 1.9. (**Randomized Distributed Algorithm**) A randomized distributed algorithm is a collection of probabilistic processes.

Randomized algorithms are non deterministic while non probabilistic algorithms are usually referred to as deterministic ones. However, using non probabilistic algorithms does not guarantee a deterministic behavior of a system. We can consider the case of a distributed non probabilistic algorithm running over an asynchronous system. The order of the events (as the order of the receipt of messages) can vary from one execution to another due to the asynchronism of the system. This can lead to an outcome of the system that differs from one execution to another.

1.1.3 Complexity

Analyzing an algorithm consists on defining theoretical estimates for the resources needed by this algorithm to resolve a given computational problem [44]

Definition 1.10. (**Time Complexity**) The time complexity is the maximal time required by this algorithm for any execution.

If the algorithm is synchronous, the time complexity is the number of rounds needed for the execution of this algorithm.

If the algorithm is asynchronous, this complexity corresponds to the maximal number of unit of times needed in the worst case in a synchronous execution of this algorithm.

Definition 1.11. (**Message Complexity**) In a system based on message passing, the message complexity is measured by the total number of messages exchanged during the execution of an algorithm.

Definition 1.12. (**Bit Complexity**) The total number of bits in the messages of an algorithm is counted by the bit complexity.

In the analysis of a randomized algorithm, which is rather established is bounds on the expected value of a given performance measure [58]. In [57, 42, 43], some techniques for the design and for the analysis of randomized distributed are presented. One concept that may be encountered in the context of the analysis of such algorithms is: with high probability (w.h.p).

Definition 1.13. (**With High Probability**) An event occurs with high probability if, for any $\alpha \geq 1$, the event occurs with probability at least $1 - \frac{c_\alpha}{n^\alpha}$, where c_α depends only on α.

Since we can choose α , we can make the probability arbitrarily low, at a cost of time and/or space.

In the sequel, with high probability will mean with probability $(1 - o(\frac{1}{n}))$, where n represents the size of the network where the distributed algorithm runs.

1.1.4 Termination and Termination Detection

A distributed algorithm designed to compute a given function can implicitly or explicitly terminate. With an implicit termination, an algorithm has a finite execution and reaches a last configuration where each node of the system has the correct output. However, the nodes do not know that they reached the last states of the execution of this algorithm. This not the case when the distributed algorithm explicitly terminates. In fact, the nodes become aware of the termination that can be local or global [40]. Accordingly, a local detection of a local termination consists on the fact that a node knows that it has locally terminated. While, a local detection of a global termination allows a node with its local knowledge to be aware that all the system terminated.

Designing distributed algorithms with termination detection is still one of the basic issues especially in case of anonymous distributed systems.

1.2 Graph Theory

A graph is a set V of vertices and a set E of edges (V and E are finite sets unless declared otherwise) such that all the endpoints of the edges in E are contained in V [63]. This graph is often denoted $G = (V, E)$, or (V_G, E_G), or $(V(G), E(G))$.

The topology of a distributed system is usually represented by a graph [67, 12] where: the set of vertices represents the set of the processes of the system, and the set of edges represent the communication links between the processes. We therefore introduce some notions and terminologies of graph theory that are useful for the study of distributed systems and algorithms. The definitions are basically taken from [63, 19, 67, 22].

We would mention that in the sequel we can use node as a synonym for a vertex and which refers to whatever is modeled by a vertex in a graph model.

1.2.1 Undirected Graphs

Definition 1.14. (**Undirected Graphs**) An undirected graph G consists of a set of vertices V and a set of edges E such that E is a collection of unordered pairs of nodes from V. Then, for any u and v from V, $(u, v) \in E$ is equivalent to $(v, u) \in E$.

The size of a graph is the cardinality of its set of vertices. In the sequel, it will be denoted n. And the cardinality of the set of its edges will be denoted m.

Definition 1.15. (**Graphs with no self-loops**) A loop or a self-loop is an edge whose both endpoints are the same vertex.

A graph G is with no self-loop if $\forall e \in E$, the extremities of e are two distinct elements of V.

Definition 1.16. (**Simple Graphs**) A simple graph is a graph with no self-loops and where each two vertices are connected by at most one edge.

Definition 1.17. (**Subgraph**) A subgraph of a graph $G = (V_G, E_G)$ is a graph $H = (V_H, E_H)$ whose vertex set and edge set are subsets of V_G and E_G, respectively, such that for each edge $e \in E_H$, the endpoints of e (as they occur in G) are in V_H.

H is called a spanning subgraph of G if $V_H = V_G$.

Definition 1.18. (**Adjacency**)

- An edge e is adjacent with a vertex v if v is an endpoint of e.
- Vertices u and v are adjacent if there is an edge whose endpoints are u and v.
- Two edges are adjacent if they have a common endpoint.

Definition 1.19. (**Neighborhood**) A neighbor of a vertex is any vertex to which it is adjacent. The set of the neighbors of a vertex u, in a graph G, is denoted $N_G(u)$.

Definition 1.20. (**Degree**) The degree of a vertex v, denoted in the sequel $d(v)$, is the size of the set of neighbors $N_G(v)$.

Definition 1.21. (**Path**) A path of length k between two vertices v_0 and v_k is a sequence $P = (v_0, \ldots, v_k)$ of nodes such that: $\forall\, i < k$, $(v_i, v_{i+1}) \in E$. v_0 is the begin node in this path, and v_k the end node.

A cycle is a path whose end node is the begin node.

Definition 1.22. (**Distance in a graph**) $\forall u, v \in V$, the distance between u and v, denoted $dist(u, v)$, is the length of the shortest path between this two vertices.

Definition 1.23. (**Diameter of a graph**) The diameter of a graph is the largest distance between any two vertices in this graph.

Definition 1.24. (**Connectivity**) An undirected graph G is connected if $\forall u, v \in V$, a path between u and v exists in G.

Definition 1.25. (**Matching**) Let $G = (V, E)$ be an undirected graph. M is a matching of G if $M = \{e \in E \mid \forall e_1, e_2 \in M,\ if\ e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ then $u_1 \neq u_2$, $u_1 \neq v_2$, $v_1 \neq u_2$ and $v_1 \neq v_2\}$.

In words, a matching M of G is a set of edges from E such that each two edges from M do not share any vertex.

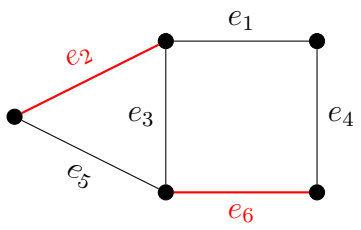

Figure 1.1: The set $M = \{e_2, e_6\}$ is a matching of this graph.

Definition 1.26. (**Maximal Matching**) Let $G = (V, E)$ be an undirected graph. Let M be a matching of G. The matching M is maximal, if by adding any edge to it, it will not be a matching any more. More formally, M is maximal if, for any matching M' of G, $M \not\subset M'$.

The matching represented in Figure 1.1 is maximal.

1.2.2 Directed Graphs

Definition 1.27. (**Directed Graph**) A directed graph G (or digraph) is defined by a set V of nodes and a set E (sometimes also denoted A) of arcs, or directed edges, and by two functions $s; t : E \to V$ that specify the source and the target of each arc.

We rewrite some of the definitions presented above adapted to the context of directed graphs.

Definition 1.28. (**Neighborhood**) In a digraph $G = (V, E)$, an arc (u, v) from E is called an outgoing edge of u and an incoming edge of v. u is called an out-neighbor of v and v is an in-neighbor of u.

Definition 1.29. (**Degree**)

- The in-degree of a node v, denoted $d^-(v)$, is the number of its incoming edges.
- The out-degree of a node v, denoted $d^+(v)$, is the number of its outgoing edges.
- The degree d of a node equals the sum of its in-degree and out-degree.

Definition 1.30. (**Connectivity**)

- A directed graph G is strongly connected if $\forall u, v \in V$, there is a path between u and v in G.
- A directed graph is weakly connected if, by replacing all of its directed edges with undirected edges, we obtain a connected undirected graph.

Definition 1.31. (**Symmetric Graphs**) A symmetric directed graph (V, A, s, t) is a digraph endowed with a symmetry, that is, an involution $Sym : A \to A$ such that $\forall a \in A$, we have: $s(a) = t(Sym(a))$.

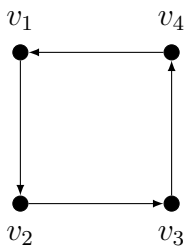

Figure 1.2: A symmetric directed graph

Figure 1.2 represents a symmetric directed graph where: $Sym((v_1, v_2)) = (v_4, v_1)$, $Sym((v_2, v_3)) = (v_1, v_2)$, $Sym((v_3, v_4)) = (v_2, v_3)$ and $Sym((v_4, v_1)) = (v_3, v_4)$.

1.2.3 Graphs Topologies

Definition 1.32. (**Tree**) A tree is an undirected, connected, acyclic graph.

Definition 1.33. (**Spanning Tree**) A spanning tree of a graph G is a spanning subgraph of G, whose structure is a tree.

Definition 1.34. (**Star**) A graph is a star if it is a rooted tree of a maximum diameter 2. The root is called the center of the star, and the remaining vertices are called the leaves.

Definition 1.35. (**Regular Graph**) A graph is regular if all its vertices have the same degree.

Definition 1.36. (**Bounded Degree Graph**) A graph $G = (V, E)$ has a degree bounded by Δ, if $\forall v \in V$, $d(v) \leq \Delta$.

Definition 1.37. (**Complete Graph**) A graph is complete if for any pair of its vertices, there is an edge linking them. A complete graph of n vertices is denoted K_n and it has $\frac{n(n-1)}{2}$ edges.

Definition 1.38. (**Random Graph**) A random graph is a graph obtained by starting with a set of n vertices and adding edges between them at random.

Different random graph models produce different probability distributions on graphs. We focus on a commonly studied one: the Erdős-Rényi $G_{n,p}$ model.

Definition 1.39. ($G_{n,p}$ **Graph**) A $G_{n,p}$ graph is a random graph of n vertices that includes each possible edge independently with probability p (and so, the edge is not included with probability $q = 1 - p$).

1.3 Probability Theory

We recall in this section some concepts related to the probability theory and that are going to be useful later for algorithms analysis. These definitions are basically taken from [63] and [57].

1.3.1 Basic Definitions

Definition 1.40. (**Experiment**) An experiment is any physically or mentally conceivable undertaking that results in a measurable outcome.

Definition 1.41. (**Sample Space**) The sample space is the set Ω of all possible outcomes of an experiment.

Definition 1.42. (**Event**) An event in the sample space Ω is a subset of Ω.

Let $\{E_j | \, j \in J\}$ be a family of events, then:

Definition 1.43. (**Union of Events**) The union $\bigcup_{j \in J} E_j$ is the set of outcomes belonging to at least one E_j.

Definition 1.44. (**Intersection of Events**) The intersection $\bigcap_{j \in J} E_j$ is the set of all outcomes belonging to every E_j.

Definition 1.45. (**Complement of an Event**) The complement $\overline{E}$ of an event E is the set of outcomes in the sample space not belonging to E.

Definition 1.46. (**Disjoint Events**) Any two events E and F are disjoint if $E \cap F = \emptyset$.
The events E_1, E_2, E_3, . . . are pairwise disjoint if every pair E_i, E_j of distinct events are disjoint.

Definition 1.47. ($\sigma-$**Field**) Let Ω be a sample space. A $\sigma-$field (or $\sigma-$algebra) $\mathbb{F}$ of Ω consists of a collection of subsets of Ω satisfying the following conditions:

1. $\emptyset \in \mathbb{F}$.
2. If $E \in \mathbb{F}$, then $\overline{E} \in \mathbb{F}$.
3. If $E_1, E_2, \ldots \in \mathbb{F}$, then $E_1 \cup E_2, \cup \ldots \in \mathbb{F}$.

$(\Omega, \mathbb{F})$ is called a measurable space.

Definition 1.48. (**Probability Measure**) Given a measurable space $(\Omega, \mathbb{F})$, a probability measure $\mathbb{P}r$: $\mathbb{F} \to [0,1]$ is a function that satisfies the following conditions:

1. $\mathbb{P}r(\Omega) = 1$.
2. $\mathbb{P}r(\bigcup_{i=1}^{\infty} E_i) = \sum_{i=1}^{\infty} \mathbb{P}r(E_i)$, if the events $\{E_i\}$ are pairwise disjoint.

Definition 1.49. (**Probability Space**) A probability space $(\Omega, \mathbb{F}, \mathbb{P}r)$ consists of a measurable space $(\Omega, \mathbb{F})$ with a probability measure $\mathbb{P}r$ defined on it.

Proposition 1.1. *Let E be an event, then*

$$\mathbb{P}r(\overline{E}) = 1 - \mathbb{P}r(E).$$

Proposition 1.2. *(**Principle of inclusion-exclusion: the simple form**) For any events E and F,*

$$\mathbb{P}r(E \cup F) = \mathbb{P}r(E) + \mathbb{P}r(F) - \mathbb{P}r(E \cap F).$$

Proposition 1.3. *(**Booles inequality**) If E_1, E_2, ..., E_n are events, then*

$$\mathbb{P}r\left(\bigcup_{i=1}^{n} E_i\right) \leq \sum_{i=1}^{n} \mathbb{P}r(E_i).$$

Proposition 1.4. *(**Monotonicity**) If $F \subseteq E$, then*

$$\mathbb{P}r(F) \leq \mathbb{P}r(E).$$

1.3.2 Dependence and Independence

Definition 1.50. (**Independence**) Two events E and F are independent if:

$$\mathbb{P}r(E \cap F) = \mathbb{P}r(E)\,\mathbb{P}r(F).$$

The n events E_1, E_2, ..., E_n are independent if: $\forall\, k$ such that $2 \leq k \leq n$, and for j_1, j_2, ..., j_k with $1 \leq j_1 < j_2 < \ldots < j_k \leq n$, we have:

$$\mathbb{P}r(E_{j_1} \cap E_{j_2} \cap \ldots \cap E_{j_k}) = \mathbb{P}r(E_{j_1})\,\mathbb{P}r(E_{j_2}) \ldots \mathbb{P}r(E_{j_k}).$$

Definition 1.51. (**Conditional Probability**) Given two events E and F with $\mathbb{P}r(F) > 0$, the conditional probability of E given F is defined as,

$$\mathbb{P}r(E|F) = \frac{\mathbb{P}r(E \cap F)}{\mathbb{P}r(F)}.$$

Proposition 1.5. *(**Law of total probabilities**) For any event E and any partition of Ω into events F_1, F_2, . . . , F_n,*

$$\mathbb{P}r(E) = \sum_{i=1}^{n} \mathbb{P}r(E \cap F_i) = \sum_{i=1}^{n} \mathbb{P}r(E|F_i)\,\mathbb{P}r(F_i).$$

Proposition 1.6. *(**Chain Rule**) For any events E_1, E_2, . . . , E_n satisfying:*

$$\mathbb{P}r(\bigcap_{i=1}^{n-1} E_i) > 0,$$

we have:

$$\mathbb{P}r(E_1 \cap E_2 \cap E_3 \ldots \cap E_n) = \mathbb{P}r(E_1)\,\mathbb{P}r(E_2|E_1)\,\mathbb{P}r(E_3|E_2 \cap E_1)\ldots\,\mathbb{P}r(E_n|\bigcap_{i=1}^{n-1} E_i).$$

1.3.3 Random Variables

Definition 1.52. (**Random Variable**) A random variable X is a real-valued function on a probability space Ω.

Definition 1.53. (**Distribution Function**) The distribution function of a random variable X is the function given by

$$F_X(x) = \mathbb{P}r(X \leq x).$$

Definition 1.54. (**Density Function**) The density function of a random variable X is

$$p_X(x) = \mathbb{P}r(X = x).$$

Definition 1.55. (**Expected Value**) The expected value (mean) $\mathbb{E}$ of a discrete random variable X is given by $\mathbb{E}(X) = \sum_x x\,\mathbb{P}r(X = k)$, where the summation is over the range of X.

Proposition 1.7. *(**Linearity of the Expectation**) Let X_1, X_2, . . . , X_n be random variables, and a_1, a_2, . . . , a_n reals. Then,*

$$\mathbb{E}(a_1X_1 + a_2X_2 + \ldots + a_nX_n) = a_1\,\mathbb{E}(X_1) + a_2\,\mathbb{E}(X_2) + \ldots + a_n\,\mathbb{E}(X_n).$$

Proposition 1.8. *(**Iterated Expectation**) Let X and Y be two random variables, then:*

$$\mathbb{E}(X) = \mathbb{E}(\mathbb{E}(X|Y)).$$

1.3.4 Probability Distributions

We now introduce some distributions that we are going to use later and describe their corresponding properties [57].

Bernoulli Distribution

We suppose that we flip a coin for which p is the probability of obtaining heads. Let X be the random variable whose value is 1 if the result is heads, and 0 otherwise. Then, X has the Bernoulli distribution with the parameter p.

Definition 1.56. (**Bernoulli Distribution**) A Bernoulli trial with parameter p, with $0 \leq p \leq 1$, is a random experiment with exactly two possible outcomes: either success whose probability is p, or failure whose probability is $q = 1 - p$.

Proposition 1.9. *If the random variable X has the Bernoulli distribution with the parameter p, then*

$$\mathbb{E}(X) = p.$$

Binomial Distribution

We consider n independently and identically distributed random variables X_1, X_2, ..., X_n whose common distribution is the Bernoulli distribution with parameter p. Let $X = X_1 + X_2 + \ldots + X_n$ be the random variable denoting the number of heads in a sequence of n coin flips. Then, X has a binomial distribution with parameter n and p, abbreviated $B(n, p)$.

Definition 1.57. (**Binomial Distribution**) Let X denotes the number of successes of a Bernoulli trial, with parameter p, repeated n times. The Binomial distribution $B(n, p)$ corresponds to the distribution of X.

Proposition 1.10. *Let k be an an integer with $0 \leq k \leq n$, then the random variable X with the binomial distribution $B(n, p)$ has the following density function:*

$$\mathbb{P}r(X = k) = \binom{n}{k} p^k q^{n-k}.$$

Also, the expected value of the number of successes is:

$$\mathbb{E}(X) = np.$$

Geometric Distribution

We repeat now the experiment of flipping a coin until head appears for the first time. We assume that each coin toss has the Bernoulli distribution with parameter p. Let X be the random variable that denotes the total number of coin flips. Then, X has the geometric distribution with parameter p.

Definition 1.58. (**Geometric Distribution**) Let consider a Bernoulli trial repeated until the first success is obtained. Let X denotes thus the number of trials. Then, X has the geometric distribution.

Proposition 1.11. *The probability of having k failures before the first success is*

$$\mathbb{P}r(X = k) = (1 - p)^k p.$$

And, the expected value of X is

$$\mathbb{E}(X) = \frac{1}{p}.$$

1.4 Mathematical Tools

We provide in this section some elementary mathematical materials such as the asymptotic notation, approximations for binomial coefficients,... whose definitions are basically from [57]. These tools are required later especially for the analyses of randomized protocols.

1.4.1 Notation for Asymptotics

Definition 1.59. For any two functions f and g: $\mathbb{R} \to \mathbb{R}^+$, we say that:

- $f(n) = O(g(n))$, and say that f is asymptotically at most g, if there exist positive numbers c and N such that, $\forall n \geq N$, $f(n) \leq c\, g(n)$.
- $f(n) = \Omega(g(n))$, and say that f is asymptotically at least g, if there exist positive numbers c and N such that, $\forall n \geq N$, $f(n) \geq c\, g(n)$. Thus, $f(n) = \Omega(g(n))$ if $g(n) = O(f(n))$.
- $f(n) = \Theta(g(n))$, and say that f is asymptotically the same as g, if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
- $f(n) = o(g(n))$ if $\lim_{n\to\infty} f(n)/g(n) = 0$. We say that f is asymptotically strictly smaller than g.
- $f(n) \sim g(n)$ if $\lim_{n\to\infty} f(n)/g(n) = 1$.

1.4.2 Combinatorial Inequalities

We start by defining the binomial coefficients as follows:

Definition 1.60. (**Binomial Coefficients**) Let $n \geq k \geq 0$, then

$$\binom{n}{k} = \binom{n}{n-k} = \frac{n!}{k!(n-k)!}.$$

If $k > n > 0$, then $\binom{n}{k} = 0$.

The binomial coefficients owe their name to their appearance in the binomial expansion:

$$(p+q)^n = \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k}.$$

We now introduce the following power series expansions.

Proposition 1.12. $\forall x$,

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Proposition 1.13. $\forall x$,

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} \dots$$

$\forall\, n \in \mathbb{N}$, the nth Harmonic number H_n is defined as follows :

Definition 1.61. (**Harmonic Number**) $\forall\, n \in \mathbb{N}$, the Harmonic number is:

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \dots + + \frac{1}{n}.$$

Proposition 1.14. *For any $n \in \mathbb{N}$, the nth Harmonic number is*

$$H_n = \ln(n) + \Theta(1).$$

We also consider the inequality involving the harmonic, geometric and arithmetic means:

Proposition 1.15. *Let $n \geq 1$, and let $(x_1, x_2, \ldots, x_n)$ positive real numbers. Then we have:*

$$\frac{n}{\sum_{k=1}^{n} \frac{1}{x_k}} \leq \left(\prod_{k=1}^{n} x_k \right)^{\frac{1}{n}} \leq \frac{1}{n} \sum_{k=1}^{n} x_k \, .$$

1.4.3 Markov Chains

We introduce concepts related to the Markov chains, based on studies in [57, 43, 69].

Definition 1.62. (**Stochastic Process**) A stochastic process is simply a collection of random variables indexed by time.

Definition 1.63. (**Filtration**) Let $\mathbb{F}$ be a $\sigma-$field. A filtration $(\mathcal{F}_n)_{n\geq 0}$ is an increasing family of sub-$\sigma-$algebras of $\mathbb{F}$ such that:

$$\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \mathcal{F}_2 \ldots \subseteq \mathbb{F}.$$

Definition 1.64. (**Adapted Process**) A process $X = (X_n)_{n\geq 0}$ is called adapted to the filtration $(\mathcal{F}_n)$ if, for each n, X_n is $\mathcal{F}_n$-measurable.

Definition 1.65. (**Markov Property**) Let $(\Omega, \mathbb{F}, \mathbb{P}r)$ be a probability space with a filtration $(\mathcal{F}_s,\ s \in I)$ for some (totally ordered) index set I, and let $(S, \mathcal{S})$ be a measurable space. A $(S, \mathcal{S})$-valued stochastic process $X = (X_t, t \in I)$ adapted to the filtration is said to possess the Markov property if, for each $A \in \mathcal{S}$ and each $s, t \in I$ with $s < t$

$$\mathbb{P}r(X_t \in A | \mathcal{F}_s) = \mathbb{P}r(X_t \in A | X_s).$$

In the case where S is a discrete set with the discrete $\sigma-$algebra and $I = \mathbb{N}$, this can be reformulated as follows:

$$\mathbb{P}r(X_n = x_n | X_{n-1} = x_{n-1}, \ldots, X_0 = x_0) = \mathbb{P}r(X_n = x_n | X_{n-1} = x_{n-1}),\ \forall n \in \mathbb{N}.$$

Definition 1.66. (**Discrete-time Markov Chain**) A discrete-time Markov chain is a sequence of random variables X_1, X_2, X_3, ... with the Markov property, that is:

$$\mathbb{P}r(X_n = x_n | X_{n-1} = x_{n-1}, \ldots, X_0 = x_0) = \mathbb{P}r(X_n = x_n | X_{n-1} = x_{n-1}),\ \forall n \in \mathbb{N}.$$

A Markov chain $\mathcal{M}$ is thus a discrete-time stochastic process defined over a set of states S, which is the possible values of X_i with of a matrix P of transition probabilities. The transition probability matrix P has one row and one column for each state in S. The

Markov chain is in one state at any time, making state-transitions at discrete time-steps $t = 1, 2, \ldots$.

The probability that the next reached state is j, knowing that the current one is i, is P_{ij} in the transition probability matrix. Accordingly, $\forall\, i,\, j \in S$, we have $0 < P_{ij} < 1$, and $\sum_j P_{ij} = 1$.

Satisfying the Markov property implies the memorylessness of a Markov chain. In fact, only the current state of a Markov chain, and not how it reached it, influences its future behavior.

1.4.4 Martingales

We here introduce the martingales, super-martingales and sub-martingales. The following definitions are basically taken from [69].

Definition 1.67. (**Martingale, Super-martingale and Sub-martingale**) A process X is called a martingale (relative to $(\mathcal{F}_n, \mathbb{P}r)$) if:

1. X is adapted.
2. $\forall n,\, \mathbb{E}(|X_n|) < \infty$.
3. $\mathbb{E}(X_n|\mathcal{F}_{n-1}) = X_{n-1}$, with $n \geq 1$.

A super-martingale satisfies the same conditions except the third one, which replaced by the following:

$$\mathbb{E}(X_n|\mathcal{F}_{n-1}) \leq X_{n-1}, \text{ with } n \geq 1.$$

Also, a sub-martingale is defined similarly by maintaining the first and second conditions and replacing the last one by:

$$\mathbb{E}(X_n|\mathcal{F}_{n-1}) \geq X_{n-1}, \text{ with } n \geq 1.$$

1.4.5 Chernoff Bound

In the following definition, we present one of the possible forms of the Chernoff Bound usually useful in establishing upper bounds of randomized algorithms with high probability [43].

Definition 1.68. (**Chernoff Bound**) Let X be the sum of n independent random variables each of which is 1 with probability p and 0 with probability $1 - p$. Hence, X has a binomial distribution $B(n, p)$ and $\mathbb{E}(X) = np$. Then:

$$\forall\, 0 < a < np,\, \mathbb{P}r(|X - \mathbb{E}(X)| > a) \leq 2e^{-a^2/3np}.$$

1.5 Conclusion

We introduced in this chapter some basic concepts related to: distributed systems and algorithms, graph theory, mathematical and probability theory. These concepts are going to be used in the study and the analyses of the population protocols that can be considered as distributed algorithms running under probabilistic schedulers. This model of population protocols is going to be introduced in the chapter that follows.

Chapter 2

The Population Protocols

We are interested in the population protocol model, a computational one introduced by Angluin et al. in [5, 4]. This model was designed for networks of tiny artifacts, such as sensor networks, that are passively mobile finite state machines. With no control of their movement, and with limited memory resources, these artifacts cooperate by establishing pairwise interactions to compute some function of their distributed entries.

We present in this chapter the formal definition of this model and the characterization of its computational power. We then give an overview of its variants: the one way population protocols that restrict the interactions to one way [9], the network graphs protocols that are designed to study graph properties of the network formed by these artifacts [3], and the fast population protocol that assume the existence of an already elected leader in the population where the protocol runs [7].

We also present models that extend the population protocols. The extension can consist on assigning states to the communication links which corresponds to the mediated population protocols [29]. Or, it can consist on attributing more memory space to the agents as it is the case for the passively mobile machines protocols [27]. The community protocols also extend the population protocols by attributing a distinguished identity to each artifact [41].

2.1 The Population Protocols

Angluin et al. introduced the population protocol model in 2004 [5, 4]. It is a pairwise computational model designed for anonymous, passively mobile, finite state agents forming populations of finite but unbounded sizes. A population protocol executed in a population can compute a function or a predicate of distributed entries gathered from the environment of this population. Each agent of this population considers its entry as an input which is then mapped to a state according to an input function. Thereafter, interactions between pairs of agents can take place. Any two interacting agents communicate to each other their respective states and update them according to a defined transition function of the protocol. These pairwise interactions never stop, but the result of the computation of the population protocol can be retrieved from the outputs of the agents once this protocol stabilizes.

A more formal description of this model as well as a characterization of its computational are presented in this section.

2.1.1 The Population Protocol Model

Angluin et al. define a **population** $\mathcal{P}$ as a finite set of n agents (with $n \geq 2$) that can establish pairwise communications [5, 4]. Two agents are able to interact if they share a communication link. An **interaction graph** $G = (A, E)$, also called **communication graph**, is associated to the population $\mathcal{P}$, with $E \subseteq A \times A$ the set of directed edges representing all permissible interactions between pairs of agents. The interaction graph G can not contain self-loops or multi-edges.

Formally, a population protocol that can run in a population $\mathcal{P}$ is described as follows [5, 4]:

Definition 2.1. (**Population Protocol**) A population protocol $\mathcal{A}$ consists of a 6−tuple (X, Y, Q, I, O, δ), where:

- X: a finite input alphabet,
- Y: a finite output alphabet,
- Q: a finite set of states,
- I: $X \to Q$: an input function mapping inputs to states,
- O: $Q \to Y$: an output function mapping states to outputs,
- δ: $Q \times Q \to Q \times Q$: a transition function defined on pairs of states as a set of transition rules. If $\delta(q_u, q_v) = (q'_u, q'_v)$, then: $(q_u, q_v) \mapsto (q'_u, q'_v)$ is called a transition, and $\delta_1(q_u, q_v) = q'_u$, and $\delta_2(q_u, q_v) = q'_v$ are defined.

The transition function δ is not symmetric. Consequently, $\delta(q_u, q_v)$ is not necessarily equal to $\delta(q_v, q_u)$. This is due to the fact that, for an interacting ordered pair of agents (u, v) with states (q_u, q_v), each agent plays a distinguished role: u is the **initiator**, while v is the **responder** (or also called **receiver**). This is a fundamental assumption for asymmetry for the population protocol model. Such possible interaction is represented by the directed edge (u, v) in the set of edges E of the interaction graph.

Now, let $\mathcal{A}$=(X, Y, Q, I, O, δ) be a population protocol. Let $\mathcal{P}$ be a population consisting of the finite set A of agents and whose interaction graph is $G = (A, E)$. Running the protocol $\mathcal{A}$ in the population $\mathcal{P}$ can be described as follows according to Angluin et al. [5, 4]:

Initially, all agents receive their inputs from their environment. In case of sensor networks, these inputs represent the sensed values of a fixed parameter such as: temperature, humidity, *etc*. The inputs are described by the alphabet X. This initialization corresponds to the **input assignment** x which is a function such that: $x : A \to X$.

Thereafter, all inputs are mapped to states from Q according to the input function I. The set of resulting states forms the initial configuration C_0 of the population $\mathcal{P}$. As introduced in [5, 4], a **population configuration** is a snapshot of the states of the agents forming the population. A more formal definition can be as follows:

Definition 2.2. (**Population Configuration**) A configuration of the population $\mathcal{P}$ where the protocol $\mathcal{A}$ runs, is a mapping C: $A \to Q$ that specifies the corresponding state of each agent of this population.

Once states are attributed, interactions between pairs of agents can take place. If two agents are able to interact, they establish a two-way communication. They hence exchange their respective states and update them according to the transition function δ. If this update leads to a modification of at least one of the two states, then the population transitions from the current configuration to a new one.

Definition 2.3. (**Transition**) Let C and C' be population configurations, and let u and v be distinct agents. We say that C goes to C' via an encounter $e = (u, v) \in E$, denoted $C \xrightarrow{e} C'$, if:

- $C'(u) = \delta_1(C(u), C(v))$,
- $C'(v) = \delta_2(C(u), C(v))$ and,
- $C'(w) = C(w)$ for all $w \in A \setminus \{u, v\}$.

C' is the configuration resulting from the interaction between the pair of agents u and v on the configuration C.

We say that C goes to C' in one step, denoted $C \to C'$, if $C \xrightarrow{e} C'$ for some encounter $e \in E$. $C \to C'$ is called a transition.

Starting from a configuration C, the configurations that can be reached by the population $\mathcal{P}$ after some (one or more) transitions are called **reachable** configurations.

Definition 2.4. (**Reachable Configuration**) Let C and C' be two population configurations. We say that C' is reachable from C if there is a sequence of configurations $C_0, C_1, \ldots, C_k$ where $C = C_0$ and $C_k = C'$ and $\forall i \in \{0, 1, \ldots, k-1\}$, $C_i \to C_{i+1}$. We denote this by $C \xrightarrow{*} C'$.

Accordingly, a formal definition describing the execution of a population protocol over a population is as follows [4]:

Definition 2.5. (**Execution**) An execution is a finite or infinite sequence of population configurations: C_0, C_1, ..., C_i, ..., such that: $\forall i$, $C_i \to C_{i+1}$.

An infinite execution is **fair** if for every possible transition $C \to C'$, if C occurs infinitely often in the execution, then C' also occurs infinitely often.

A **computation** is an infinite fair execution.

We can deduce from the definition of a computation that a population protocol does not halt and that the pairwise interactions never stop. Nevertheless, a population protocol can stabilize (converge). Stabilization is reached once the outputs stop changing: this is called **output stabilization**.

Definition 2.6. (**Output Stabilization**) A population protocol $\mathcal{A}$ reaches output-stabilization when its computation leads to a configuration C such that, for any C' reachable from C, $O(C) = O(C')$ and consequently, $\forall\, u \in A$, $O(C(u)) = O(C'(u))$.

The **output assignment** of a configuration C is a function $y_C : A \to Y$ defined as $y_C(u) = O(C(u))$, $\forall\, u \in A$. In other words, a population protocol reaches output-stabilization if, it reaches a configuration C such that: for any configuration C' reachable from C, C and C' have the same output assignment. The configuration C is called an **output-stable configuration**.

Stabilization is a global property of the population that agents, based on their limited knowledge (which consists of their own states), can not detect. Hence, they continue interacting. However, whatever the interactions that take place are, the outputs do not change.

If the computation of the protocol does not converge, the output of this computation is said undefined.

There is a particular case of output stabilization, which is **state stabilization** [54].

Definition 2.7. (**State Stabilization**) A population protocol $\mathcal{A}$ state-stabilizes, or has stabilizing states, if any computation of $\mathcal{A}$ eventually reaches a configuration C such that: for every configuration C' reachable from C, $C' = C$. The configuration C is a state-stable configuration.

State stabilization is a stronger requirement compared to output stabilization. Any protocol that state-stabilizes also output-stabilizes, however the inverse is not always true. We would also mention that "eventually"means "in a finite number of steps", that depend on the choice of the encounters during transitions. We will explore this in details in Chapter 5.

Angluin et al. focused on **always-convergent** population protocols only. A protocol $\mathcal{A}$ is always-convergent if every computation on every input assignment x converges.

Let $\mathcal{X} = X^A$ denote the set of all input assignments of the population protocol $\mathcal{A}$. And, let $\mathcal{Y} = Y^A$ denote the set of all its output assignments.

Let $R_\mathcal{A}$ be a relation. $\forall x \in \mathcal{X}$, $\forall y \in \mathcal{Y}$, $R_\mathcal{A}(x, y)$ holds iff there is a computation of $\mathcal{A}$ beginning in configuration C_x, where $C_x(w) = I(x(w))$ for all agent $w \in A$, that stabilizes to output y. We say that the population protocol $\mathcal{A}$ running in the population $\mathcal{P}$ **stably computes the input-output relation** $R_\mathcal{A}$.

Now, if the relation $R_\mathcal{A}$ is a single-valued relation, and more specifically a predicate, then a definition of its stable computation can be as follows:

Definition 2.8. (**Stably Computable Predicate**) Let $p : \mathcal{X} \to \{0, 1\}$ be a predicate over $\mathcal{X}$. p is said to be stably computable by the population protocol model $\mathcal{A}$ if for any input assignment $x \in \mathcal{X}$, any computation of $\mathcal{A}$ starting from $I(x)$ eventually reaches an output stable configuration C where, for all agents w, $O(C(w)) = p(x)$.

That is, when a population protocol $\mathcal{A}$ stably computes a predicate p, it stabilizes to an output stable configuration in which all agents agree about a correct answer which is the output of $p(x)$, consequently 0 or 1.

According to the description of the population protocol model, we can notice that these protocols have two main characteristics that are:

- Anonymity: The design of a population protocol does not depend on the identities of the agents. In fact, these agents are supposed to be equipped with a constant memory space. Consequently, they do not have enough memory space to store identities: they are hence anonymous.

- Uniformity: The design of a population protocol does not depend on the population size.

To better illustrate the computation of a population protocol, we consider now as an example the scenario of the flock of birds. This scenario was introduced by Angluin et al. in [5].

Example 2.1. The Threshold Protocol. We consider a population of birds where each one is equipped with a sensor to measure its temperature. A bird is supposed to have an elevated temperature if this last is higher than a defined constant c. The sensors are also equipped with wireless media that enable them to form an ad-hoc network. They

communicate and exchange data to provide the monitor of this flock of birds with some information about the global state of this flock. Let us consider the following question: are there at least five birds with elevated temperatures? If yes, an alert of a possible epidemic should be launched. Thus, let N_{SB} denote the size of the set of sick birds (those having elevated temperatures). The question can then be reformulated to the following predicate: $N_{SB} \geq 5$.

The authors proposed the population protocol $Threshold$, that once run in the monitored population, provides the answer to this question. This protocol is described by the $6-$tuple (X, Y, Q, I, O, δ) with:

- $X = Y = \{0, 1\}$,
- $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$,
- $I(i) = q_i$,
- $O(q_i) = \begin{cases} 0, & \text{if } 0 \leq i \leq 4 \\ 1, & \text{otherwise.} \end{cases}$
- $\delta(q_i, q_j) = \begin{cases} (q_{i+j}, q_0), & \text{if } 0 \leq i + j < 5 \\ (q_5, q_5), & \text{otherwise.} \end{cases}$

The computation of this protocol begins at a global start signal. Each sensor measures the temperature of the bird carrying it. The result of this measurement is described with the input alphabet and should consequently be as follows: 1 if the temperature is elevated, and 0 otherwise. Each agent (sensor) maps its input according to the input function I such that: $I(1) = q_1$ and $I(0) = q_0$. Pairwise interaction can then take place. Each two communicating sensors exchange their states and update them according to δ. If the sum of both states is less than 5, then the state of the initiator is updated to this sum, while the state of the responder is reinitialized to q_0. Otherwise, they both become with state q_5.

This protocol stabilizes if we wait a sufficient interval after the global signal start. If there are at least five birds with elevated temperatures, the state q_5 will appear and will be spread among the whole population. Hence, all the agents, being with state q_5, will output 1 which corresponds to a "yes" answer to the asked question (and to "true" as a value of the threshold predicate). However, if the number of birds with elevated temperatures is less than 5, the state q_5 will never appear and all the agents will output 0 which corresponds to a "no"answer (or "false" as a value of the predicate). The correct answer can be retrieved from any sensor of the population.

2.1.2 Computational Power

The computational power of the population protocols was characterized in [6, 10] to be exactly the Presburger Arithmetic. The Presburger arithmetic, also defined as semi-linear predicates, is the first-order theory of natural numbers with addition and no multiplication [62]. It is a restriction of Peano's arithmetic where all the axioms for multiplication are removed.

According to [4], a Presburger formula can be expressed as a Boolean formula over predicates that can be described by one of the following three forms:

1. $\sum a_i\, x_i + c_1 < \sum b_i\, x_i + c_2$
2. $\sum a_i\, x_i + c_1 = \sum b_i\, x_i + c_2$
3. $\sum a_i\, x_i + c_1 \equiv_m \sum b_i\, x_i + c_2$. The symbol $\equiv_m$ with $m \geq 2$ denotes equivalence modulo m. $\forall x$, y integers, $x \equiv_m y$ holds iff $x \equiv y \pmod{m}$, that is: there exist two integers z and q such as: $(z = m \times q)$ and $(x + z = y)$.

where a_i, b_i, c_1, c_2 and m are integer constants and x_i are non-negative integers.

2.1.3 Restricted Communication Graph

Angluin et al. initially supposed that population protocols run in populations of agents that are passively mobile and that, consequently, each pair of agents will eventually meet and interact. Thus, they assumed that the interaction graphs are complete so that all pairs interactions are permissible. Hence, the computational power of population protocols was characterized for those running over complete interaction graphs. Yet, the authors proved in [4] that, as long as the interaction graph is still connected, restricting the interactions does not affect the computational power of the population protocols. In fact, any predicate that is stably computable in a complete interaction graph is also stably computable in any weakly-connected interaction graph.

2.2 Variants of the Population Protocol Model

In the sequel, we are going to present three variants of the population protocol model also proposed by Angluin et al. The first restricts the bidirectional communication to unidirectional ones. The second, is a slightly modification of the original model that focuses on network graphs properties. The last one supposes the existence of an already elected leader in the population.

We should mention that for these variants the condition of stable computation does not require anymore that all agents output 0 or 1. It suffices that the output satisfies some specified requirement that depends on the protocol and that can be for example: only one agent outputs 1, while all the rest output 0.

2.2.1 One Way Population Protocols

The population protocol model supposes that the communication between a pair of interacting agents is bidirectional and simultaneous. This model may sometimes be unrealistic for some networks with radio communications.

Thus, Angluin et al. proposed in [9] the one way population protocols. They are a modified version of the population protocols where the communications are not bidirectional anymore but only one way. An interaction is hence split into separate send and receive events, and the transition function is also restricted. In fact, given an interacting pair, the responder changes its state according to the received state of the initiator. However the state of the initiator, if it is updated, will not depend on that of its responder as it does not know it.

There are different versions of the one way population protocols model. An initiator may be able to detect that it has sent a message and accordingly changes its state. This

is called the transmission model. However, in the observation model, the sender is not able to detect this event and then becomes unable to change its state. In this case, the sender is passively observed by the receiver.

Also, in the one way population protocol model, the delivery can be immediate so that the send and receive events happen simultaneously, as it can be delayed (or queued) where the send and receive may imply delayed messages.

An example of a one way population protocol follows.

Example 2.2. The 3−States Approximate Majority Protocol. Suppose that in a given population $\mathcal{P}$ there are two candidates, x and y, for some election. Some agents of the population may decide to vote for x, others may choose y, while the rest of the population may be undecided and can choose to vote blank. We should mention that the number of agents choosing x, and also those choosing y, should be > 0.

To decide what the majority voted for, Angluin et al. proposed the following 3-State Approximate Majority protocol [8]. It is a one way population protocol consisting of the 6-tuple $(X_{AM}, Y_{AM}, Q_{AM}, I_{AM}, O_{AM}, \delta_{AM})$ where:

- $X_{AM} = Q_{AM} = \{x, y, b\}$ where b denotes the blank vote,
- $Y_{AM} = \{x, y\}$,
- $I_{AM}(x) = x$, $I_{AM}(y) = y$, $I_{AM}(b) = b$,
- $O_{AM}(x) = x$ and $O_{AM}(y) = y$,
- $\delta_{AM}(x, b) = (x, x)$, $\delta_{AM}(x, y) = (x, b)$, $\delta_{AM}(y, b) = (y, y)$, and $\delta_{AM}(y, x) = (y, b)$.

The input of each agent in this population corresponds its choice for this vote. The inputs are themselves the states of the agents as the input function I_{AM} is the identity function. Then pairwise interactions start. When an initiator is with state x or y, it attempts to convert any responder that it meets to adopt its state. It can succeed to immediately convert an undecided responder. However, it can only reduce an opposing responder to undecided status.

The protocol stabilizes when a consensus on one of the values x and y is reached. Then, all the agents output x or y. If initially the number of agents with state x exceeded the number of agents with state y, the consensus will be on x and vice versa.

2.2.2 Network Graphs Protocols

When the finite state agents of a population communicate, they form a network represented by the interaction graph supposed to be at least weakly connected. Angluin et al. proposed to study graph properties of these networks. They defined a graph property as a function p from a graph G to the set $\{0, 1\}$ where $p(G) = 1$ if and only if G has the corresponding property. They then introduced protocols designed to study such graph properties: the network graphs protocols [3].

A network graph protocol is a population protocol where all the agents start with the same initial state. A graph property is stably computable if all the agents eventually converge to the correct output. The authors described some stably computable properties i.e: whether the communication graph is a directed star, or it contains a directed cycle, or it has degree bounded by some constant, etc.

Example 2.3. The In-Degree Protocol. We here present the example of the network graph protocol that can decide whether a graph G has a vertex of in-degree greater than 1. This protocol was introduced by Angluin et al. in [3].We call this protocol the In-Degree protocol and it consists of (X, Y, Q, I, O, δ) where:

- $X = \{-\}$,
- $Y = \{0, 1\}$,
- $Q = \{-, I, R, T\}$,
- $\forall x \in Q$, $O(x) = 1$ if $x = T$, and 0 otherwise,
- $\delta : \begin{cases} (-,-) \mapsto (I,R), \\ (I,R) \mapsto (-,-), \\ (-,R) \mapsto (T,T), \\ (T,x) \mapsto (T,T),\ \forall x \in Q, \\ (x,T) \mapsto (T,T),\ \forall x \in Q. \end{cases}$

All the vertices (agents) start with state $-$. If every vertex in the graph has in-degree at most 1 then only the first two transitions rules occur. Consequently, the protocol will stabilize to a configuration where all agents output 0. However, if there is some vertex with in-degree at least 2, then the state T appears and will be spread. In such case, when the protocol stabilizes, all the agents output 1.

Angluin et al. also proposed two models that are slight modifications of the network graphs protocols: the network graphs protocols with stabilizing inputs, and the non deterministic population protocols [3].

Network Graphs Protocols with Stabilizing Inputs

In this variant of network graphs protocols, the inputs of the agents of the population are not initially fixed. They vary for a while and then finish by stabilizing. The only modification regarding the original model concerns the transition function that becomes as follows:

$$\delta : (Q \times X) \times (Q \times X) \to (Q \times Q)$$

Thanks to this variant, the composition of protocols becomes possible. Indeed, let A and B be two protocols such that: A is a network graph protocol whose outputs are from a finite alphabet X, and B is a network graph protocol with stabilizing inputs that are also from X. Then, when composing A with B, the stabilizing outputs of A will be the stabilizing inputs of B.

Non Deterministic Population Protocols

In non deterministic population protocols, the transition function δ is non deterministic. It describes more than one possible transition for a given pair of states as it is shown in the example below.

Example 2.4. The d−Coloring Protocol. Coloring a graph consists on attributing colors to its vertices. A graph is d-colorable if each of its nodes can be colored with one of the d colors such that no two neighbor nodes have the same color.

The following non deterministic protocol stabilizes to a correct d-coloring if the network graph is d-colorable. Otherwise, if the network graph is not d-colorable, this protocol will never stabilize and the nodes will never stop changing their colors.

- $X = \{0\}$,
- $Y = Q = \{0, 1, ..., d-1\}$,
- $I(x) = x$ and $O(x) = x$, $\forall x \in \{0, 1, ..., d-1\}$,
- $\delta(i, i) = (j, k)$ where j and k are randomly chosen from the set $\{0, 1, ..., d-1\}$.

The authors of [3] stated that the nondeterminism of the transition function does not increase the class of stably computable predicates.

2.2.3 Fast Population Protocols with a Leader

Fast population protocols are population protocols where initially there is a leader supposed to be already elected in the population [7]. To be able to design a fast population protocol, a leader should first be designated.

We recall that leader election consists on designating a unique process as a leader. It is a fundamental problem not only in population protocols, but in distributed systems in general. The population protocol, called **Leader Election** protocol, that elects only one leader in a population was introduced by Angluin et al. in [3]. It consists of the following 6−tuple (X_{LE}, Y_{LE}, Q_{LE}, I_{LE}, O_{LE}, δ_{LE}):

- $X_{LE} = \{L\}$,
- $Y_{LE} = \{L, F\}$,
- $Q_{LE} = \{L, F\}$,
- $I_{LE}(x) = x$ and $O_{LE}(x) = x$, $\forall x \in \{L, F\}$,
- $\delta_{LE}(L, L) = (L, F), \delta_{LE}(L, F) = (F, L)$ and $\delta_{LE}(F, L) = (L, F)$.

In the classical leader election algorithms in distributed computing, all the processes of the distributed system initially start with the same state which is “simple candidate”. Then, they become either “elected (L)” or “failed (F)”. Also, once a node updates its state to ”elected”, it never changes anymore.

Nevertheless, this population protocol presents a different approach as initially all agents start by being “leader”. Furthermore, when this protocol stabilizes with only one “leader” state in the whole population, this state can be a swapping one.

During a pairwise computation, if both agents are with state L, the initiator preserves its state while the responder becomes F. Otherwise, if one agent is with state L and the other F, they just interchange their states.

When the protocol stabilizes, it guarantees that there is only one agent with the state L. However, this leader state may jump from one agent to another from one computation step to another.

This population protocol works in any weakly connected interaction graph. Yet, if we suppose that this last one is complete, we can restrict δ_{LE} to only one transition rule which is $\delta_{LE}(L, L) = (L, F)$. We can thus guarantee that the leader state will not be a moving one.

Now, once a leader is appointed, it can proceed in coordinating the agents of the population to compute a new protocol. This represents the fundamental assumption of the fast population protocols. In fact, such coordination may lead to a faster stabilization of some population protocols: that is less computation steps are needed to reach a protocol's stabilization. This was proved by Angluin et al. [7].

2.3 Models Extending the Population Protocols

Once the population protocol model was introduced and its computational power exactly characterized, some new open questions arised. How some modifications such as: according a state to the edge linking the interacting pair, according more memory space to the agents, or giving identities to the agents of the population, ... may affect the original model as well as its computational power. We here present some models that extend the population protocol model and that also provide some answers to these questions.

2.3.1 The Mediated Population Protocols

The mediated population protocols extend the basic population protocols by adding states to the edges of the interaction graph which is translated into equipping each communication link with a buffer of constant storage capacity. This model preserves the anonymity and the uniformity of the original population protocol model.

A more formal and detailed description of the mediated population protocol model follows [29].

Definition 2.9. (**Mediated Population Protocol**) A mediated population protocol consists of $(X, Y, Q, S, I, O, \delta, \iota, \omega)$ where:

- X: a finite input alphabet,
- Y: a finite output alphabet,
- Q: a finite set of agent states,
- S: a finite set of edge states,
- I: $X \to Q$: an agent input function mapping inputs to agents states,
- O: $Q \to Y$: an agent output function mapping agents states to outputs,
- δ: $Q \times Q \times S \to Q \times Q \times S$: a transition function,
- ι: $X \to S$: an edge input function mapping inputs to edge states,

- ω: $S \to Y$: an edge output function mapping edge states to outputs.

  A possible additional component, but which is not mandatory, of a mediated population protocol is an output instruction r. This output instruction specifies how the output of this protocol should be interpreted.

As an example of the mediated population protocols, we present the VarProduct protocol [29].

Example 2.5. The VarProduct Protocol. Let $\mathcal{P}$ be a population whose interaction graph is complete and where each agent can have as initial input: a, b, or c. Let N_i denote the size of the set of agents with state i. We consider the following predicate: $N_c = N_a \times N_b$. As already mentioned in a previous section (see Section 2.1.2), the Presburger arithmetic does not include multiplication. As a consequence, there is no population protocol that can stably computable this predicate.

Chatzigiannakis et al. introduced then the following VarProduct mediated population protocol $(X, Y, Q, S, I, O, \delta, \iota, \omega)$ that can stably compute this predicate:

- $X = \{a, b, c\}$, $Y = \{0, 1\}$,
- $Q = \{a, \bar{a}, b, c, \bar{c}\}$,
- $S = \{0, 1\}$,
- $I(x) = x$, $\forall x \in X$,
- $O(a) = O(b) = O(\bar{c}) = 1$ and $O(c) = O(\bar{a}) = 0$,
- $\iota(x) = 0$, $\forall x \in X$,
- $\omega(x) = x$, $\forall x \in S$,
- $\delta(a, b, 0) = (\bar{a}, b, 1)$, $\delta(c,\bar{a}, 0) = (\bar{c}, a, 0)$, and $\delta(\bar{a}, c, 0) = (a,\bar{c}, 0)$.

The output instruction r of this protocol can be: "If there is at least one agent with output 0, then reject, else accept."

As the interaction graph is complete, $N_a \times N_b$ equals the number of edges leading from agents in state a to agents in state b. Thanks to edge states, an agent with state a can remember an agent with state b that it has already met: the edge linking them is marked with state 1 instead of 0. For each agent b that it has met, an agent with state a, and that becomes $\bar{a}$, tries to erase an agent with state c. Consequently, if initially $N_c > N_a \times N_b$, then the protocol stabilizes to a configuration where there is at least one agent with state c that remains and whose output is 0. Otherwise, if $N_c < N_a \times N_b$, then the protocol stabilizes with at least one agent with state $\bar{a}$ in the configuration and whose output is also 0. Then, $N_c = N_a \times N_b$ is true only if there no agent with state c or $\bar{a}$ that remains, so that all agents output 1.

Chatzigiannakis et al. hence proved that the model of mediated population protocols has a stronger computational power compared to the population protocol model.

The model of mediated population protocols was also extended, like the population protocols, to study graph properties. In this context, the graph decision mediated population protocols (GDMPP) and the mediated graph protocols (MGP) were designed [28].

2.3.2 The Passively Mobile Machines Protocols

The constant memory of an agent in population protocols is a constraining characteristic of this model. A new model relaxing this constraint was introduced in [27]: the passively mobile machines protocols. This model extends the population protocols by allowing the agents to have greater memory space. An agent is not an automaton anymore, but a Turing machine.

We provide a formal description of this new model.

Definition 2.10. (**Passively Mobile Machines Protocol**) A passively mobile machines protocol is a 6-tuple $(X, \Gamma, Q, \delta, \gamma, q_0)$ where:

- X is the finite input alphabet, where $B \notin X$ (B is the blank symbol),
- Γ is the finite tape alphabet, where $B \in \Gamma$ and $X \subset \Gamma$,
- Q is the finite set of states,
- $\delta : Q \times \Gamma^4 \longrightarrow Q \times \Gamma^4 \times \{L, R\}^4 \times \{0, 1\}$ is the internal transition function,
- $\gamma : Q \times Q \longrightarrow Q \times Q$ is the external transition function, and
- $q_0 \in Q$ is the initial state.

In the passively mobile machines protocol model, each agent is supposed to be equipped with:

- A sensor to sense a specified parameter from the environment. The sensed value will be considered as the input of the agent.
- Four read/write tapes which are: the incoming message tape, the outgoing message tape, the working tape, and the output tape. These tapes are assumed to be unbounded only to the right.
- A control unit where the state of the agent is stored. This unit applies the transition functions.
- Four heads, one for each tape. A head is able to read from and write to the cells of the tape. It is also able to remain stationary or to move either to the left or to the right.
- A binary working flag that can be set to 0 when the agent is ready for interaction or to 1 when this last is working internally.

All agents start with the state q_0, with a working flag set to 1 and with tapes containing only the blank symbol. Then, they all get their inputs from their environment described with symbols from the alphabet X. This input symbol will be written in the working tape of the agent. As the working flag is set to 1, the control unit can apply the internal transition function. This unit reads its own state as well as the symbols under the heads of the four tapes. Then, after updating all of them, moves to left/right or keeps stationary each head of the tapes, and finally sets the working flag to 0 or 1.

The pairwise interactions in this model depends on the working flag of the agents. In fact, two agents can interact only if their respective working flags are set to 0. When an interaction between two agents takes place, the external transition function is applied to update the states of these interacting pair. The outgoing message of each of these two agents is copied in the incoming message tape of the other (replacing, thus, the previous content of this tape). Then, the working flags of this pair of agents are set to 1 as the internal transition function should be applied.

PALOMA

One particular case of the passively mobile machines protocols is the model of passively mobile logarithmic machines (PALOMA) [26]. In this model, each agent is equipped with $O(\log(n))$ memory space, where n is the size of the population. Chatzigiannakis et al. proved that there is a PALOMA protocol that stably computes the predicate $N_c = N_a \times N_b$. Also, assigning unique identities, from the set $\{0, 1, \ldots, n-1\}$, to the agents of the population is also possible with this model. In the corresponding protocol, the agents start with the identity 0. When an interaction takes place, the initiator compares its id with the responder id. If they are equal, the initiator increments its id. As the interaction graph is supposed to be complete in this model, all agents will eventually meet. And thanks to the fairness of the execution of the protocols, each agents will finally have a unique id from $\{0, 1, \ldots, n-1\}$.

2.3.3 The Community Protocols

In a community, each individual has a distinguished name or identity. This inspired Guerraoui et al. in their work in [41] where they proposed a new model extending the population protocols called the model of community protocols. In this model, an agent can store, in addition to $O(1)$ bits of states, a unique identifier, and $O(1)$ other agents identifiers. These identifiers are in a read only mode. They are used only for comparison and thus never modified.

Let U be an infinite ordered set containing all possible identifiers and let $\perp$ be the null identifier. Then, a community protocol can formally be described as follows [41]:

Definition 2.11. (**Community Protocol**) A community protocol consists of an 8-tuple $(X, Y, B, d, I, O, Q, \delta)$, where:

- X is the finite input alphabet,
- Y is the finite output alphabet,
- B is the finite set of basic states,
- d is a non negative integer representing the number of identifiers that can be recorded by an agent,
- $I : X \longrightarrow B$ is the input function mapping input symbols to basic states,
- $O : B \longrightarrow Y$ is the output function mapping the basic states to output symbols,

- $Q = B \times (U \bigcup \{\perp\})^d$ is the set of agent states, and
- $\delta : Q \times Q \longrightarrow Q \times Q$ is the transition function.

The initial state of each agent in a population running a community protocol is of the form $(b_i, id_i, \perp, \perp, \ldots, \perp)$. $b_i \in B$ is the initial basic state of this agent and it equals $I(x_i)$ with x_i the input of this agent. id_i is the unique identifier of this agent. Then, the remaining components of this state are $d - 1$ repetitions of the symbol $\perp$.

Community protocols preserve the uniformity property as their design do not depend on the size of the population. However, they are clearly not anonymous.

Guerraoui et al. proved that their model has a stronger computational power than the population protocols. They also proved that it is robust and can tolerate the presence of a constant number of agents with Byzantine failures. An agent that has a Byzantine failure is an agent that can behave arbitrarily. It can pretend being in any state when interacting with other agents.

2.4 Conclusion

We presented in this chapter the computational model of population protocols. Run in a population of anonymous agents with finite set of states, a population protocol computes a predicate of the distributed inputs of these agents, gathered from their environment. Interactions between pairs of agents allow them to exchange their states and to update them according to some given rules. The result of the computation can be extracted from the outputs of the agents of the population when the protocol stabilizes. Population protocols have two basic properties: uniformity and anonymity. We gave an overview of the variants of this model, as well as models that enhance it, that all preserve the uniformity property but not the anonymity. These models also, except the one way population protocols, are all based on pairwise interactions like the population protocols.

In the next chapter, we are going to compare the population protocol model (and some of its extensions) with two other models and establish a bridge that allows us to describe it with each one of them. The first one is a model that also uses abstract communications. However, the second one is a model with explicit communications.

Chapter 3

A Bridge between Population Protocols, Tasks with Graph Relabeling Systems and Anonymous Asynchronous Message Passing

The population protocols, as they are based on local interactions and computations, belong to local computation systems. In this chapter, we introduce another model of local computation systems, and more specifically of local computations in graphs, which is tasks with graph relabeling systems [48, 47]. Population protocols, as well as tasks with graph relabeling systems, are models using abstract communications. We present in this chapter a comparative study between these two models and establish some analogy between them. This allows us to prove the possibility of mapping the execution of a (mediated) population protocol to a realization of a task according to a graph relabeling system. However, we prove that the reverse is possible only under some conditions.

On the other hand, population protocols were designed for networks of tiny objects such as sensor networks. Hence, we think that supposing abstract communications is too theoretical for such networks. We thus propose to describe the computation of a population protocol in a population of anonymous agents, as a distributed algorithm running in an anonymous asynchronous system based on message passing.

3.1 From Population Protocols to Tasks with Graph Relabeling Systems

A distributed system is a collection of interacting processes that collaborate to reach a global result. These interactions can be based on a shared memory model where the processes perform read/write primitives on shared registers. Another interaction model, is the message passing model where the processes exchange messages, through the communication links, using send/receive primitives. A third basic interaction model, is the local computations model on which are based the local computations systems [24].

We will focus, in this section, on this last model, and more specifically on the local computations in graphs with tasks and graph relabeling systems. This model present, in fact, some similarities with the (mediated) population protocols that we found interesting to investigate.

3.1.1 Local Computations Systems

Local computations systems are described as networks of processors that have only access to local resources. The local resources consist of the state of the process, the states of its neighbors and those of the links between them. They are the result of the restricted local interactions of each process with its direct neighbors. In a local computation system, all processors execute the same program.

Accordingly, the population protocol model, as well as the models extending it, belongs to this class of systems that have been the object of different studies. Von Neumann introduced in [68] the cellular automata model where the computations are made by a collection of synchronously communicating finite automata forming a network, such as a grid, which is highly symmetric and regular (no mobility). According to the states of its neighbors, an agent can update its state. A similar model, that does not require the regular structure of the network, is proposed by Rosenstiehl et al. in [64]. A computation step also consists on computing the next state of each process according to its current state and those of its neighbors. Litovsky, Métivier, Sopena and Zielonka [48, 47] proposed

a model based on graph relabeling (rewriting) systems where a distributed computation is described as a transformation of the graph representing the distributed system.

We focus on this last model which is a tool for coding distributed algorithms [46] and proving their correctness [15], used even to recognize some graph properties and families of graphs [39]. This model presents some correspondences with the population protocols model that we would later explore.

3.1.2 Local Computations in Graphs with Graph Relabeling Systems

In the model of Litovsky et al., a network is represented as a finite connected graph with some level of abstraction of the distributed system model. The processes are represented by the set of vertices and the direct communication links are represented by the set of edges. The local state of each process (respectively communication link) is denoted by a label assigned to the corresponding vertex (respectively edge). These local states associated to each vertex and edge describe the global state of the network, denoted labeling of the graph.

Accordingly, a labeled graph can formally be described as follows [46]:

Definition 3.1. ($L-$**labeled graph**) A $L-$labeled graph (or a graph labeled over L) is a graph where the vertices and the edges have labels from a possible infinite alphabet L. It is denoted by (G, σ) where $G = (V, E)$ is a graph and $\sigma : V \cup E \to L$ is the labeling function. G is called the **underlying graph** of (G, σ) and σ is a **labeling** of G.

A state transformation of a network represented by a graph G corresponds to a **relabeling** on this graph: it is any pair of global states $((G, \sigma), (G, \sigma'))$ of labeled graphs over the same underlying graph G. A **relabeling relation** on G is a set of relabellings on G.

Based on these concepts, Litovsky et al. introduced the graph relabeling systems to describe local computations in graphs.

Graph Relabeling Systems

Local interactions in a local computation system can be traduced by graph relabeling rules. A graph relabeling rule illustrates the possible modification of the labels attached to the vertices and edges of a graph if they satisfy a given description. Formally, it is defined as follows [46] :

Definition 3.2. (**Graph Relabeling Rule**) A graph relabeling rule is a triple $R = (G_R, \sigma_R, \sigma'_R)$ such that (G_R, σ_R) and (G_R, σ'_R) are two labeled graphs. The labeled graph (G_R, σ_R) is the left-hand side of R and (G_R, σ'_R) is the right-hand side.

An algorithm can be encoded by means of local relabellings. These relabellings are according to the graph relabeling rules of the graph relabeling system associated to this algorithm. A graph relabeling system can be formally described as follows [46]:

Definition 3.3. (**Graph Relabeling System**) A graph relabeling system is a triple $\mathcal{R} = (\mathcal{L}, \mathcal{I}, \mathcal{P})$ where $\mathcal{L}$ is a set of labels, $\mathcal{I}$ a subset of $\mathcal{L}$ called the set of initial labels and $\mathcal{P}$ a finite set of relabeling rules.

A computation step, resulting from a local interaction, corresponds to a relabeling step having the following definition [46]:

Definition 3.4. (**Relabeling Step**) Let $\mathcal{R} = (\mathcal{L}, \mathcal{I}, \mathcal{P})$ be a graph relabeling system. A $\mathcal{R}-$relabeling step is a $5-$tuple $(G, \sigma, R, \varphi, \sigma')$ such that: $R = (G_R, \sigma_R, \sigma'_R)$ is a relabeling rule in $\mathcal{P}$, and φ is both an occurrence of (G_R, σ_R) in (G, σ) and an occurrence of (G_R, σ'_R) in (G, σ').

In other words, a relabeling step consists on applying the relabeling rule R that modifies the labels of the elements of $\varphi(G_R, \sigma_R)$ according to σ'_R. Consequently, the graph G has a new labeling σ', obtained from σ.

A computation (or an execution) corresponds to a relabeling sequence. A computation stops when no relabeling rule of the graph relabeling system is applicable anymore [46]:

Definition 3.5. ($\mathcal{R}-$**irreducible**) A labeled graph (G, σ) is $\mathcal{R}-$irreducible if there is no occurrence of (G_R, σ_R) in (G, σ) for every relabeling rule R in $\mathcal{P}$.

Characteristics of Local Computations with Graph Relabeling Systems

Computations with graph relabeling systems, as they belong to local computations in graphs, satisfy the following conditions [15]:

1. They do not affect the structure of the underlying graph, but only the labeling of its vertices and edges.
2. The locality of these interactions implies that each relabeling step can only change the labellings of a connected subgraph with a fixed size in the underlying graph.
3. The local context of the relabeled graph determines if a relabeling rule is applicable or not, which means that these computations are locally generated.

Local Interaction Rules and Local Synchronizations

A local interaction in a local computation system may involve a process with only one of its direct neighbors, or with all its direct neighborhood. Any local computation step needs a local synchronization between the interacting entities. And as a relabeling rule in a graph relabeling system stands for a local interaction, a local synchronization is needed to synchronize the components of the subgraph of the underlying labeled graph where this relabeling rule will be performed. The model of Litovsky et al. present three types of local computations (LC) that need three different types of local synchronizations [15]:

- **LC0** This local computation is a relabeling step according to a relabeling rule that involves a node with one of its neighbors and the edge linking them. Consequently, at least one of the labels of these elements will be modified. This local computation needs the synchronization of this node with its neighbors.
- **LC1** A $LC1$ computation is a relabeling step that involves a star formed by a center node and all its direct neighbors. The label of this center node as well as the labels of the edges of this star can be modified. However, the labels of the leaves of this star remain unchanged. A $LC1$ synchronizes a ball of radius 1, that is a center node and all its neighbors at distance 1.

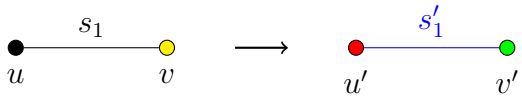

Figure 3.1: $LC0$ relabeling rule updating the label of the edge (u, v) from s_1 to s'_1, and modifying the labels of the vertices respectively from u and v, to u' and v'.

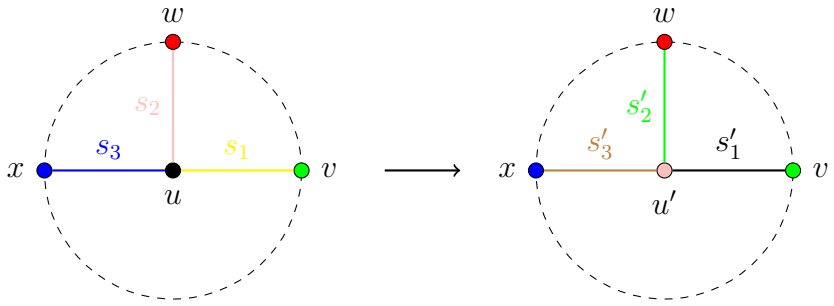

Figure 3.2: $LC1$ relabeling rule updating the label of the center node, labeled u, and all its incident links labeled s_1, s_2 and s_3, but without altering the labels of the leaves nodes x, w and v.

- **LC2** This local computation needs to synchronize a ball of radius 2: a center node and all its neighbors at distance at most 2. The application of the relabeling rule of this local computation can update the state of the center node, the states of its direct neighbors and also the states of the edges linking the center node to these neighbors.

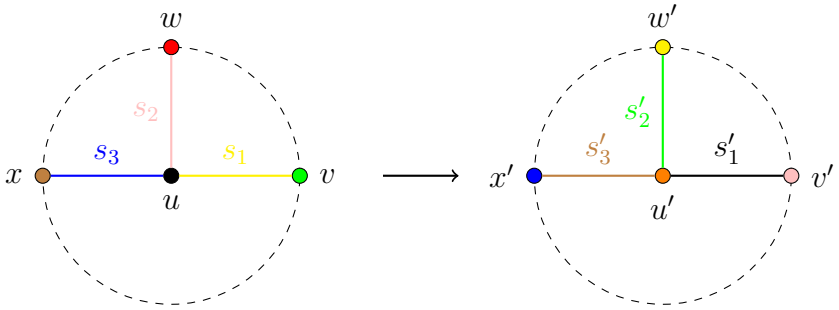

Figure 3.3: $LC2$ relabeling rule involving all components of the star whose center node is labeled u, and modifying all their labels.

Local Control Mechanisms on Graph Relabeling Systems

Local control mechanisms can be introduced to graph relabeling systems in order to restrict the applicability of the relabeling rules [45]. As a control mechanism, some priority relation on the set of relabeling rules can be defined in **graph relabeling systems with priorities** [20].

Also, some forbidden contexts can be specified on relabeling rules. A rule with forbidden context can not be applicable on some occurrence if and only if this latter is included on an occurrence of the forbidden context. This is the case of **graph relabeling systems with forbidden contexts** [46].

Example 3.1. Distributed Computation of a Spanning Tree with a Graph Relabeling System. We consider the example of a graph relabeling system, introduced in [46], encoding a distributed algorithm that computes a spanning tree on a graph. We denote this graph relabeling system by $\mathcal{R}_{\mathcal{ST}} = (\mathcal{L}_{ST}, \mathcal{I}_{ST}, \mathcal{P}_{ST})$ defined by $\mathcal{L}_{ST} = (A, N, 0, 1)$, $\mathcal{I}_{ST} = (A, N, 0)$, and $\mathcal{P}_{ST} = \{R_{ST}\}$ with R the relabeling rule described in Figure (3.4).

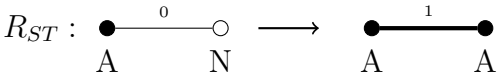

Figure 3.4: The spanning tree relabeling rule R_{ST}

Figure 3.5 illustrates a sample of a relabeling sequence with $\mathcal{R}_{ST}$ computing a spanning tree of a graph.

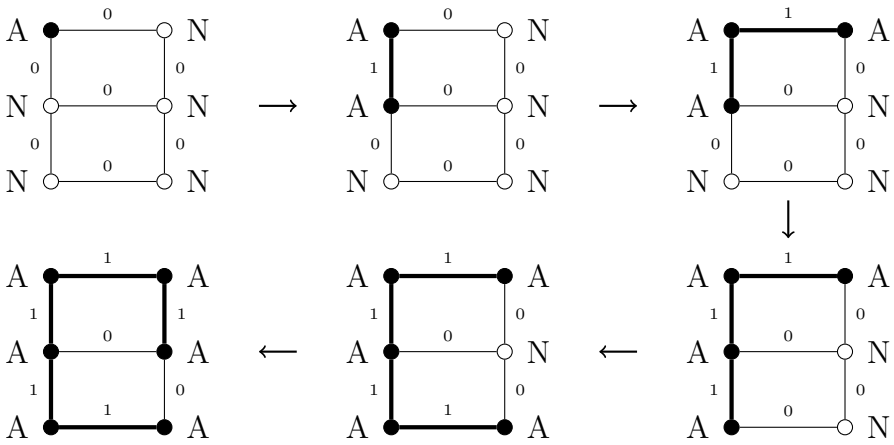

Figure 3.5: A distributed computation of a spanning tree with $\mathcal{R}_{\mathcal{ST}}$

Initially, all the nodes are labeled N, except one node which is labeled A, and all the edges are labeled 0. An elementary computation step consists of an application of the rule R_{ST}. It rewrites a label of a node labeled N, linked to a node labeled A by an edge labeled 0, to a label A. The label of the involved edge is also modified, it becomes 1. Parallel computation steps can take place simultaneously as long as the interacting pairs are disjoint. We can take the example of the 5th relabeling step and the 6th relabeling step in Figure 3.5 that can be executed concurrently. When the relabeling rule R_{ST} is not applicable anymore, the final result is obtained. It is the spanning tree formed by the edges labeled 1 and whose root is the initial node labeled A.

3.1.3 Tasks with Graph Relabeling Systems

A distributed **task** consists of a specification and a domain [40]. The **specification** is a description of what should be computed. It is a graph relabeling relation. The **domain** is a family of labeled graphs over which the local computations should lead to a correct result with respect to the specification.

A formal definition of a task corresponds to what follows [23]:

Definition 3.6. (**Task**) Let $\sum_G^L$ represent the type of $L-$labellings on a graph G. Let L_i (respectively L_o) be a type for input (respectively output) labels. A task T is a function that associates to any graph G a relation T_G between $\sum_G^{L_i}$ and $\sum_G^{L_o}$. The domain of T is the set of L_i labeled graphs (G, σ_i) such that $((G, \sigma_i), (G, \sigma_o)) \in T_G$ for some σ_o.

Realization of a Task

Given a distributed algorithm, a task can describe the problem this algorithm should solve. On the other hand, and as we mentioned in a previous section, a graph relabeling system can encode this distributed algorithm by means of local relabeling. A realization of a task is the link established between the task and the graph relabeling system associated to this distributed algorithm.

Let T be a task with labellings types L_i and L_o. Let (G, σ_i) be any labeled graph in the domain of T, and let $\mathcal{R}$ be a graph relabeling system over the labeling type $\mathcal{L}$. A computation that realizes T on (G, σ_i) according to $\mathcal{R}$ consists on the following three successive phases:

1. **Initialization** Every vertex and edge receives its initial state according to the input σ_i and to the local initialization function $\lambda : L_i \to \mathcal{L}$.

2. **Relabellings** Relabellings are made according to the relabeling rules of $\mathcal{R}$. They stop when the labeled graph becomes $\mathcal{R}-$irreducible.

3. **Extraction** Output states are extracted uniformly over L_o according to the local extraction function $\pi : \mathcal{L} \to L_o$.

we consider the following example of a realization of a task according to a graph relabeling system.

Example 3.2. Election in a Tree with Initial Knowledge of the Vertices Degrees. Let G be any network graph whose structure is a tree and where the processes initially know their degrees: each vertex of the tree is labeled with its degree. We consider a realization of a task that aims to elect a single vertex to be the leader in this network [23]. Therefore, let $\mathcal{R}_{ET} = (\mathcal{L}_{ET}, \mathcal{I}_{ET}, \mathcal{P}_{ET})$ be a graph relabeling system where:

- $\mathcal{L}_{ET} = \{None\} \cup \{Some\,(i) | \, i \in \mathbb{N}\}$,
- $\mathcal{I}_{ET} = \{Some\,(i) | \, i \in \mathbb{N}\}$,
- $\mathcal{P}_{ET} = \{R_{ET}\}$, with R_{ET} the relabeling rule illustrated in Figure 3.6.

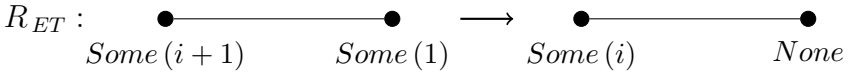

Figure 3.6: The $LC0$ rule R_{ET} for election in a tree with initial knowledge of vertices degrees

Let $T_{election}$ be a task, whose realization according to $\mathcal{R}_{ET}$ over the labeling type $\mathcal{L}_{ET}$, consists on electing a leader vertex in the graph G. The realization of $T_{election}$ can be described as follows:

- The labeling types are fixed as follows: the input labeling type is $L_i = \mathbb{N}$ representing the possible values of the initial degrees of the vertices, and the output labeling type is $L_o = \{B, E\}$.
- The initialization phase takes place according to the following input function:

$$\begin{array}{rccc} \lambda: & L_i & \to & \mathcal{L}_{ET} \\ & i & \mapsto & Some\,(i) \end{array}$$

- The relabellings are made according to the relabeling rule R_{ET}.
- The extraction phase depends on the following extraction function π:

$$\pi(None) = B \text{ and } \pi(Some\,(0)) = E.$$

Figure 3.7 represents an example of a realization of this task in a tree whose vertices are initially labeled with their degrees. Based on these initial labels, and according to the input function λ, new labels from the labeling type $\mathcal{L}_{ET}$ are attributed to the vertices. Then, relabeling steps take place according to the relabeling rule R_{ET}. The vertices that were involved in a relabeling step, as well as the edge linking them, are represented in red in this figure. The relabellings stop when the relabeling rule is not applicable anymore. If so, all the vertices are thus labeled $None$, except one which is labeled $Some(0)$. This latter becomes labeled E according to the extraction function π: this is the elected vertex. However, the extraction function π attributes the label B to all the remaining vertices labeled $None$. They are consequently considered as defeated.

3.1.4 From a Computation of a Population Protocol to a Realization of a Task

We propose in this section to establish a bridge between the population protocol model and tasks with graph relabeling systems. In fact, we noticed some similarities between these two models as they are both local computation models with abstract communication. They also, both, consider the model of pairwise interactions. We thus want to investigate these similarities and present a comparison study that may allow us to establish a link between these two models.

We would mention that the approach presented all along this section applies, not only for the population protocol model, but also for the models extending it. However, in the

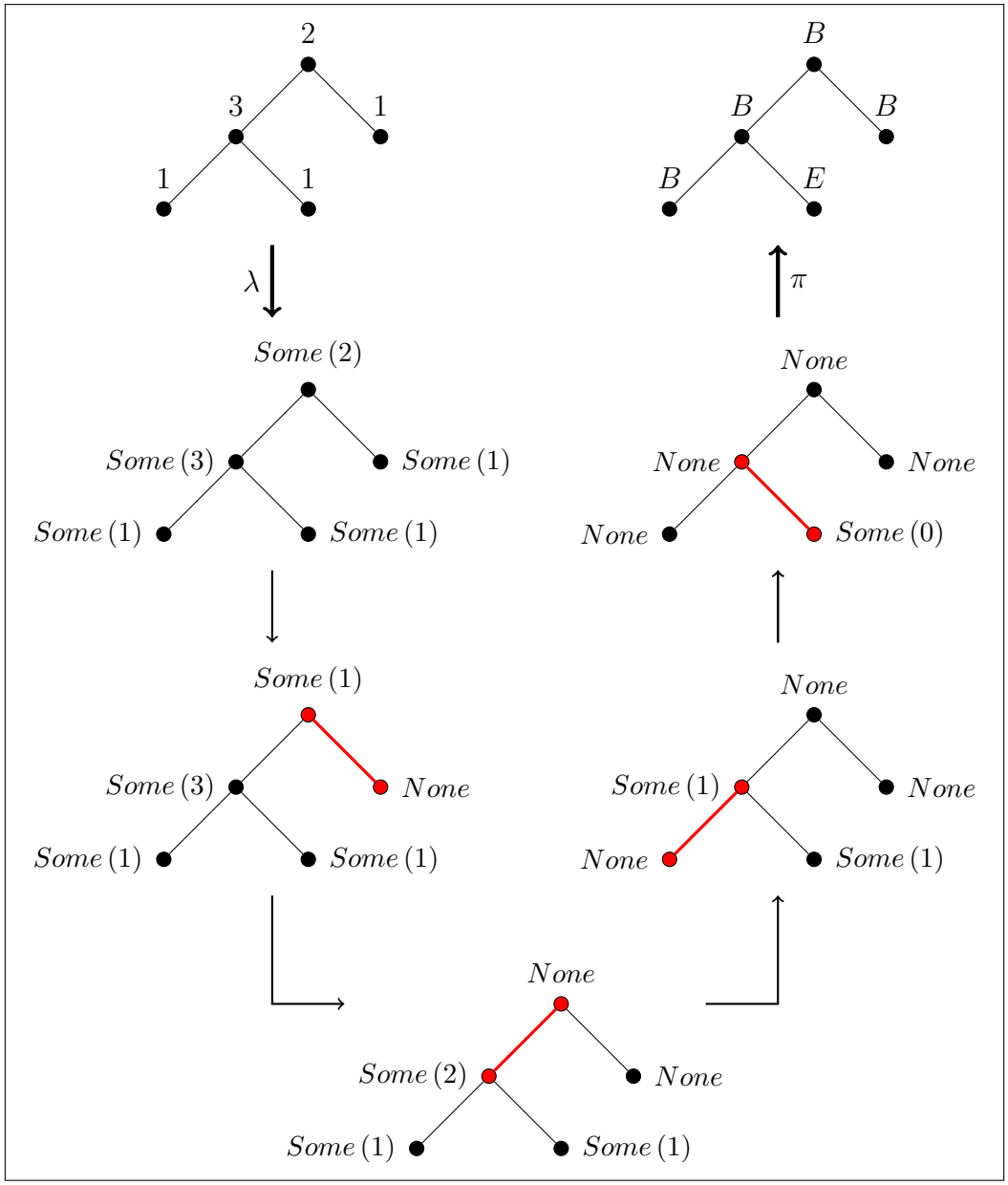

Figure 3.7: Realization of the task $T_{election}$ in a tree with initial knowledge of vertices degrees.

sequel, we will focus only on the cases of the populations protocols and the mediated population protocols. Compared to the population protocols, the mediated population protocols assign states, not only to the agents of the population, but also to the communication links. Therefore, additional details are going to be needed for this model. They will be denoted afterward between () or [].

The Population Protocol Model as Local Computations in Graphs

We aim to prove, in this section, that the computation of a (mediated) population protocol can be considered as local computations in graphs. We thus first start by representing

the interaction graph, over which the protocol runs, as a labeled graph. We then describe the effective pairwise interactions as local relabellings on this graph. Finally, we ensure that the execution of this (mediated) population protocol satisfy the properties of local computations in graphs.

Let $\mathcal{A} = (X, Y, Q[, S], I, O, \delta[, \iota, \omega])$ be a (mediated) population protocol running in a population $\mathcal{P}$. We recall that a computation of a (mediated) population protocol takes three basic phases:

1. **Initialization** The inputs are mapped to states according to the agent input function I: $X \to Q$ (and to the edge input function ι: $X \to S$).
2. **Pairwise Interactions** Interactions between pairs of agents take place and can lead to the modification of their states according to the transition function δ: $Q \times Q\,[\times S] \to Q \times Q\,[\times S]$.
3. **Outputs Extraction** The states are mapped to outputs according to the agent output function O: $Q \to Y$ (and the edge output function ω: $S \to Y$).

Let $G = (V, E)$ be the communication graph over which $\mathcal{A}$ runs. V is the set of vertices representing the set of agents of the population $\mathcal{P}$. E denotes the set of the edges representing all the permissible communication links between the pairs of agents from $\mathcal{P}$. This graph is a representation of the network formed by the agents of the population $\mathcal{P}$ with an abstraction over its characteristics.

We propose to represent this communication graph as a labeled graph. Therefore, we add labels to its vertices (and respectively to its edges) that describe the states of the agents (respectively the states of the communication links) of the population $\mathcal{P}$. The labeling type will depend on the phase of the execution of the protocol $\mathcal{A}$. Also, the labeling of this graph can be updated after a computation step.

Initially, the labels attributed to the nodes (and to the edges) of the interaction graph G are the inputs of the agents (and the inputs of the communication links) of $\mathcal{P}$. The graph G becomes thus $X-$labeled, and denoted by (G, σ_X) with $\sigma_X : V[\cup E] \to X$.

Then, once the initialization phase has taken place, this labeled graph becomes (G, σ_Q) with $\sigma_Q : V[\cup E] \to Q[\cup S]$. Any interaction happening during this second phase, may only lead to an update of the states of the interacting agents (and the communication link connecting them). This is traduced by an update of the labels of the corresponding nodes (and the edge linking them) in the graph. This update will associate to the communication graph a new labeling function over $Q[\cup S]$.

Finally, when the protocol $\mathcal{A}$ reaches stabilization and during the outputs extraction phase, the labeled interaction graph can be considered as an $Y-$labeled graph (G, σ_Y) with $\sigma_Y : V[\cup E] \to Y$.

Let u, v, u' and v' be any agent states from Q (and let s and s' be any edge states). Any transition rule $(u, v\,[, s]) \mapsto (u', v'\,[, s'])$ from δ executed over the $Q[\cup S]-$labeled communication graph can be translated to a relabeling rule $R_{uv} = (G_{uv}, \sigma_{uv}, \sigma'_{uv})$ where:

- G_{uv} is a restriction of the $Q[\cup S]-$labeled interaction graph to the interacting pair of nodes, with respective states u and v, and the edge linking them.

- σ_{uv}, respectively σ'_{uv}, is the restriction of the labeling function $\sigma : V[\cup E] \to Q[\cup S]$, respectively $\sigma' : V[\cup E] \to Q[\cup S]$, to the labellings of the nodes that initially were with states u and v (and the edge linking them).

Figure 3.8 illustrates the relabeling rule R_{uv}. Note that, while labeling the interaction graph, the direction of its edges were left untouched to preserve symmetry breaking. Hence, a transition rule applied for the ordered pair of agents (u, v) can be expressed as a relabeling rule over the directed edge (u, v) in G.

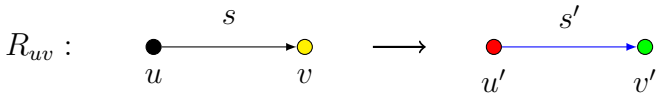

Figure 3.8: A relabeling rule representing the transition $(u, v, s) \mapsto (u', v', s')$

We define a function ϕ that associates to each transition rule from the transition function δ of the protocol $\mathcal{A}$ a corresponding relabeling rule.

This function ϕ is described as follows:

$$\begin{array}{ccccc} \phi & : & \delta & \to & P \\ & & ((u, v\,[, s]), (u', v'\,[, s'])) & \mapsto & R_{uv} = (G_{uv}, \sigma_{uv}, \sigma'_{uv}) \end{array}$$

with P a set of relabeling rules.

The (mediated) population protocol model is based on pairwise interactions. Therefore, any computation step in such protocol involves only the interacting pair of agents (and the communication link connecting them). Accordingly, any transition rule from δ can be traduced by a $LC0$ interaction.

We can notice that, any computation step in the execution of the protocol $\mathcal{A}$ has the following characteristics:

- It does not affect the structure of the communication graph.
- It can only lead to some changes of the states of the interacting pair of agents (and the communication link between them), and consequently lead to the modification of the labellings of the pair of vertices representing these two agents (and the edge linking them) in the underlying labeled interaction graph.
- It is locally generated as only the current states of the interacting pair of agents (and the communication link connecting them) determine if a transition rule is applicable or not.

We can hence deduce that the computation of any (mediated) population protocol respects the three characteristics of local computations in graphs (see Section 3.1.2). And accordingly, a (mediated) population protocol model can be described as local computations in graphs.

Mapping an Execution of a Population Protocol to a Realization of a Task

We now focus on how an execution of a (mediated) population protocol over an interaction graph can correspond to a realization of a task on a labeled graph according to some graph relabeling system.

We recall that the computation of a (mediated) population protocol consists on three phases: initialization, pairwise interactions, and outputs extraction. As already mentioned, if we describe this computation as local computations in graphs, we can consider that the pairwise interactions phase is a relabellings phase. These phases present a great degree of similarity with those needed for a realization of a task, and that are: initialization, relabellings and extraction. In fact, the computation of a (mediated) population protocol, as well as the realization of a task, needs an initialization phase to map the inputs to states. Then, a relabeling phase where interactions lead to the application of some rules. These rules may result in an update of the states and, consequently, in an update of the labels of the graph over which the computations take place. And, a last phase, where outputs can be extracted from the states.

Given this analogy, and the approach we presented in the previous section of representing a (mediated) population protocol as local computations in graphs, we propose to describe the computation a (mediated) population protocol over an interaction graph as a realization of a task with a graph relabeling system on this graph.

Accordingly, let $\mathcal{A} = (X, Y, Q[, S], I, O, \delta[, \iota, \omega])$ be a (mediated) population protocol running in a population $\mathcal{P}$ with an interaction graph $G = (V, E)$. Let T be a task with input labeling type X, and output labeling type Y. Let (G, σ_X) be a labeled graph, whose underlying graph G is the interaction graph of $\mathcal{A}$, and whose labeling function is $\sigma_X : V[\cup E] \to X$. The labeled graph (G, σ_X) is in the domain of T. Finally, let $\mathcal{R} = (Q, Q_i, P)$ be a graph relabeling system over the labeling type Q with: $Q_i = I(X)[\cup\, \iota(X)] \subset Q$, and $P = \phi(\delta)$.

The execution of the protocol $\mathcal{A}$ over the interaction graph G of the population $\mathcal{P}$ can be mapped to a realization of the task T, on the labeled graph (G, σ_X), according to the relabeling system $\mathcal{R}$. This realization is as follows:

- The initialization phase is done according to the initial input σ_X and to the local initialization function $\lambda : X \to Q\,[\cup S]$, with:

$$\forall x \in X,\ \lambda(x) = \begin{cases} I(x), & \text{if } x \text{ is the input of an agent,} \\ \iota(x), & \text{if } x \text{ is the input of an edge.} \end{cases}$$

- The relabellings are then made according to the relabeling rules P of $\mathcal{R}$.

- When the protocol stabilizes, the outputs can be extracted over the labeling type Y according to the local extraction function $\pi : Q\,[\cup S] \to Y$, with:

$$\forall q \in Q\,[\cup S],\ \pi(q) = \begin{cases} O(q), & \text{if } q \in Q, \\ \omega(q), & \text{otherwise.} \end{cases}$$

Table 3.1 summarizes this approach by attributing to each element of the execution of a (mediated) population protocol, its correspondence in the realization of the associated

task.

Correspondence	Computation of a (Mediated) Population Protocol	Realization of a Task with a Graph Relabeling System
Inputs	Input alphabet X	Input labeling type $L_i = X$
States	States sets Q (and S)	Labeling type $\mathcal{L} = Q(\cup S)$
Outputs	Output alphabet Y	Output labeling type $L_o = Y$
Initialization	Input functions $I : X \to Q$ (and $\iota : X \to S$)	Initialization function $\lambda : L_i \to \mathcal{L}$
A Computation Step	A transition from the transition function $\delta : Q \times Q(\times S) \to Q \times Q(\times S)$	A rule from the set of relabeling rules $P = \phi(\delta)$
Output Extraction	Output functions $O : Q \to Y$ (and $\omega : S \to Y$)	Extraction function $\pi : \mathcal{L} \to L_o$
Global State of the Network	Configuration	Labeling

Table 3.1: Mapping an execution of a (mediated) population protocol to a realization of a task with a graph relabeling system: Correspondence between the different elements

Example 3.3. Mapping the Computation of the Threshold Population Protocol to a Realization of a Task with a Graph Relabeling System. Let $\mathcal{P}$ be a population whose interaction graph is $G = (V, E)$. We suppose that the agents of $\mathcal{P}$ are executing the population protocol $Threshold = (X, Y, Q, I, O, \delta)$, that we presented in the previous chapter (see Section 2.1).

Let T_{Th} be a task with input, and respectively output, labeling type L_i and L_o such that:

- $L_i = X = \{0, 1\}$.
- $L_o = Y = \{0, 1\}$.

We consider the labeling function $\sigma_X : V \to X$ which is applied to the graph G to obtain the X-labeled graph (G, σ_X) in the domain of T_{Th}.

We also consider the graph relabeling system $\mathcal{R}_{Th} = (Q, Q_i, P)$ with: $Q_i = I(X) = \{q_0, q_1\} \subset Q$, and $P = \phi(\delta) = \{R_1, R_2\}$. The rules R_1 and R_2 are illustrated in Figure 3.9, and represent the only two effective transitions of the protocol $Threshold$.

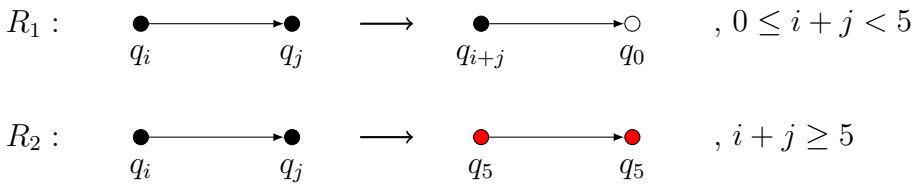

Figure 3.9: The relabeling rules of the relabeling system $\mathcal{R}_{Th}$

The execution of the population protocol $Threshold$ can be mapped to the realization of the task T_{Th} over the labeled graph (G, σ_X) according to the graph relabeling system $\mathcal{R}_{Th}$, where:

- The initialization is done according to σ_X and to the local input function λ with $\lambda = I$,
- Relabellings are done according to the relabeling rules R_1 and R_2 of $\mathcal{R}_{Th}$, and
- The outputs extraction is made according to the local output function $\pi = O$.

Now, as an example, we suppose that the population $\mathcal{P}$ consists of 8 agents, and that its interaction graph $G = (V, E)$ is a complete graph. Let the input assignment of this population be described by the vector $(1, 0, 1, 1, 1, 1, 0, 1)$. We illustrate a computation of the $Threshold$ protocol in this population, as a realization of the task T_{Th}, in Figure 3.10.

Initially, the interaction graph G is labeled with the input alphabet X, according to the input assignment of this computation, so it becomes in the domain of the task T_{Th}. Consequently, each vertex is either labeled 0 or 1. Then, applying the input function λ associates to each vertex a new label. If the vertex was labeled 0, then the new label is q_0. Otherwise, the new label is q_1. The relabellings phase starts then, standing for the pairwise interactions in the computation of the protocol. The first relabeling step, illustrated in Figure 3.10, is applied over the directed edge colored red, and whose both

extremities are labeled q_1. The relabeling rule R_1 is applied, updating the label of the source of the directed edge to q_2 and the label of its target to $q0$. Relabeling steps continue, according to the relabeling rule R_1, until the label q_5 appears. There, all the relabeling steps that take place are according to the relabeling rule R_2, so all the vertices become labeled q_5. The output extraction can thus be done: all the vertices labels are 1. The output assignment of the computation of the protocol *Threshold* can consequently be deduced from this output extraction. The value of the predicate $N_{SB} \geq 5$ is 1 (indeed, we can notice that in the input assignment, we have 6 agents with inputs 1, that is 6 agents that are sick).

3.1.5 From a Realization of a Task to a Computation of a Population Protocol

We proved that a mapping of a computation of a (mediated) population protocol to a realization of a realization of a task with a graph relabeling system is possible. Now, we are going to investigate whether the reverse is also possible.

Tasks with Graph Relabeling Systems versus Population Protocols: the Key Differences

The mapping of a computation of a (mediated) population protocol to a realization of a task, introduced above, is an analogy based on the similarities and correspondences that exist between these two models. However, this does not exclude the fact that there are some differences that distinguish one model from the other, and that will particularly make the mapping of a realization of task with a graph relabeling system to a computation of a (mediated) population protocol not always possible.

Table 3.2 gives an overview of the basic differences existing between these two models, and that we are going to detail in the sequel:

Differences	Task's Realization with Graph Relabeling System	(Mediated) Population Protocol's Computation
Memory Constraint	×	✓
Symmetry Breaking	×	✓
Star Synchronization	✓	×
Termination	✓	×
Termination Detection	✓	×
Local Control Mechanisms	✓	×

Table 3.2: A realization of a task with a graph relabeling system versus a computation of a (mediated) population protocol: Key differences.

- **Memory constraint** In the population protocol model, and all its extensions, the memory space accorded to the agents (and also to the edges) is specified varying

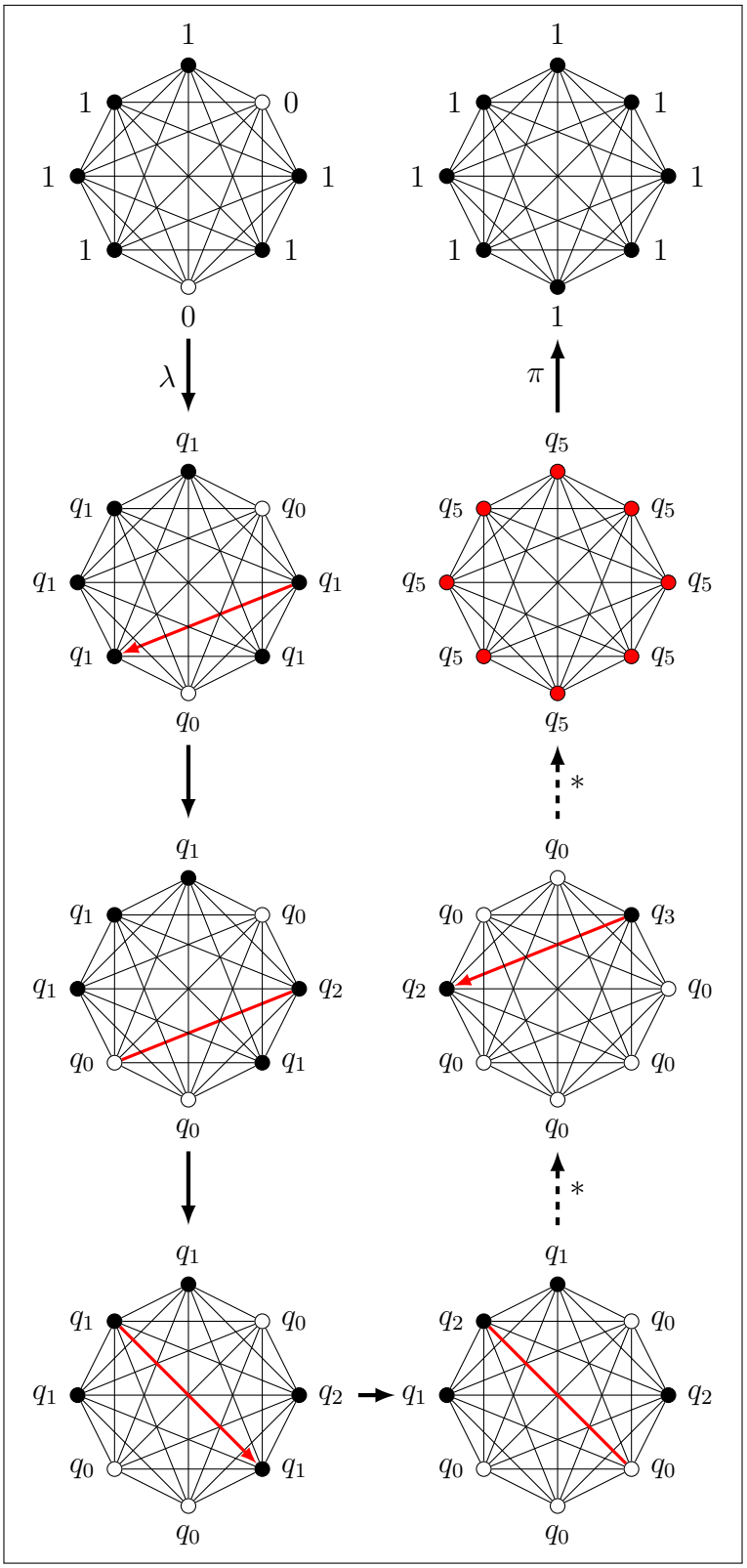

Figure 3.10: A realization of the task T_{Th}

from $O(1)$ to an unlimited space. This constraint is one of the basic elements that may influence the computational powers of these models. However, no constraint is specified concerning the memory space accorded to the nodes or edges in the case of tasks with graph relabeling systems, which implies that there is no restriction on this space.

- **Asymmetry** A pair of interacting agents, in a (mediated) population protocol, consists of one agent playing the role of the initiator, and a second one playing the role of the responder. This distinction breaks the symmetry between these two anonymous entities. Such asymmetry is not specified when applying a relabeling rule of a graph relabeling system.

- **Restriction of the interacting entities** The (mediated) population protocols are based on pairwise interactions. Therefore, the transition rules of these protocols are described by the pair of states of the two interacting agents (and the state of the edge linking them). We established, in the previous section, that these transition rules can be represented by $LC0$ relabeling rules in graph relabeling systems. However, relabeling rules in graph relabeling systems are not restricted to $LC0$. They cover more interaction models that allow a node to interact with all its direct neighborhood. These interactions are described by $LC1$ or $LC2$ (see Section 3.1.2) rules.

- **Stabilization versus termination** Population protocols do not halt but they stabilize. Even when stabilization is reached, the agents of the population continue interacting without changing their outputs (or even their states in case of state stabilization). We can consider the case of the *Threshold* population protocol with $X \geq 5$. This protocol stabilizes to a configuration where all the agents are with state q_5. And, no matter the interactions that later take place, these states do not change. However, agents can not detect that they terminate and continue interacting. This corresponds to an implicit termination in distributed systems. On the other hand, a task with graph relabeling system $\mathcal{R}$ terminates if $\mathcal{R}$ is noetherian: that is there is no more applicable relabeling rule and the graph becomes $\mathcal{R}-$irreducible. Graph relabeling system can even be designed to be able to detect this termination with: local detection of local termination, local detection of global termination, *etc.*

- **Control mechanisms on the interactions** Graph relabeling systems offer some control mechanisms on the interactions by describing relabeling rules with priorities or with forbidden contexts. The application of a relabeling rule, in a relabeling system with priorities, depends on some global knowledge about the network. This knowledge concerns the existence or not, in the labeled graph, of an occurrence of another relabeling rule that has the priority. Also, the application of a rule with a forbidden context, requires some awareness about the states of the neighbors of the nodes involved in this relabeling rule. This is possible thanks to the interactions such as $LC1$ and $LC2$ that allows a node to communicate with all its neighborhood. Unlike the graph relabeling systems, the (mediated) population protocol model does not define such control mechanisms in its transition rules. This is due to the fact that, each entity in this model, only knows its current state and the state of the agent it communicates with (and the state of the edge linking them). It has no

idea about the states of the rest of the population, not even those of its closest neighbors. Consequently, there is no possibility to check if a given priority or a forbidden context is respected while a pairwise interaction takes place.

Using these control mechanisms is one of the elements that make possible the design of graph relabeling systems with termination detection. We are going to illustrate this through the example of a leader election over a complete graph realized with the two models: a task with a graph relabeling system, and a population protocol.

For this computation with the first model, we start by considering the following graph relabeling system $\mathcal{R}_{EC}$= (L_{EC}, I_{EC}, P_{EC}) with: $L_{EC} = \{N, F, E\}$, $I_{EC} = \{N\}$ and $P_{EC} = \{R_1, R_2\}$ [14]. The rules of P_{EC} are as illustrated in Figure 3.11.

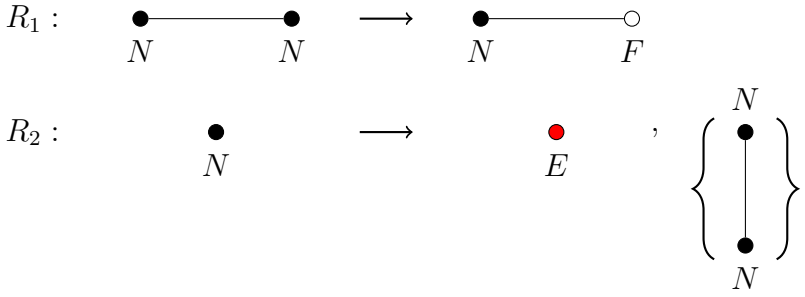

Figure 3.11: The relabeling rules of the graph relabeling system $\mathcal{R}_{EC}$

We consider a task T_{EC} with an input labeling $L_i = I_{EC}$ and an output labeling $L_o = \{F, E\}$. Both, the initialization function and the output extraction function of T_{EC} are the identity function. Let K_n be a L_i-labeled complete graph in the domain of T_{EC}. A realization of T_{EC} over K_n with respect to the graph relabeling system $\mathcal{R}_{EC}$ leads to the election of a unique vertex. Thanks to the forbidden context, the last vertex labeled N knows that it is the only remaining N in the graph and becomes E. The relabeling phase thus terminates and this elected vertex E can locally detect the global termination of the computation.

Now, we focus on this computation with the second model. We therefore consider the Leader Election population protocol presented in the previous chapter (see Section 2.2.3) running over a complete graph. When the protocol stabilizes, the elected agent is also the only remaining agent with state L. However, as no knowledge about the direct neighborhood is allowed, this agent is not able to detect that it is the only remaining L in the population. Consequently, it is enable to locally detect any termination.

We can thus notice, how the possibility, or not, of using control mechanisms has an effect on the termination detection.

Mapping a Realization of a Task with a Graph Relabeling System to a Computation of a Population Protocol

In spite of the differences between these two models presented in the previous section, the mapping approach of a (mediated) population protocol to a task with graph relabeling system established above, is still valid.

In fact, any (mediated) population protocol can be translated to a task with a graph relabeling system with only $LC0$ relabeling rules and with no priorities or forbidden context. The graph relabeling system will be noetherian only if the population protocol has a state-stabilization. By preserving the direction of the edges of the interaction graph, we preserve the order of the pair of interacting nodes at each computation step. Consequently, symmetry breaking is not altered. In fact, in a directed edge of a $LC0$ relabeling rule, the source represents the initiator and the target represents the responder.

However, this comparative study implies that the mapping of a realization of a task with a graph relabeling system to a computation od a (mediated) population protocol is not always possible. As a matter of fact, a task with a graph relabeling system can be mapped to a (mediated) population protocol, only if the following conditions are respected:

1. The graph relabeling rules of the graph relabeling system should be described as $LC0$ rules only (no $LC1$, or $LC2$, or single node relabeling rules) with no forbidden contexts or priorities. If the $LC0$ relabeling rule is described over an undirected edge, the left hand side endpoint will stand for the initiator, and the other for the responder.

2. The labellings of the vertices (and those of the edges) described by the input and output labeling type of the task, and by the set of labels of the graph relabeling system, should be of constant size.

Accordingly, let T be a task with input, respectively output, labeling types L_i and L_o (including labels of constant size only). Let (G, σ_{L_i}), with $G = (V, E)$, be a L_i-labeled graph in the domain of T where $\sigma_{L_i} : V \cup E \to L_i$. Let $\mathcal{R} = (\mathcal{L}, \mathcal{I}, P)$ be a graph relabeling system such that: the elements of $\mathcal{L}$ are of constant size, and the relabeling rules forming P are $LC0$ rules with no forbidden contexts or priorities.

We consider the realization of the task T, on the labeled graph (G, σ_{L_i}), with respect to the relabeling system $\mathcal{R}$. This realization can be mapped to a mediated population protocol if labels are attributed to the edges in the initial labeled graph, or in the relabeling rules. Otherwise, it is mapped to a population protocol. This mapping associates to, each element of this realization, its correspondence in the execution of a (mediated) population protocol $\mathcal{A} = (X, Y, Q[, S], I, O, \delta[, \iota, \omega])$. These correspondences are as described in Table 3.1.

Hence, a realization of the task T, on the labeled graph (G, σ_{L_i}) with respect to $\mathcal{R}$, can be described by a computation of the protocol $\mathcal{A}$ in a population whose interaction graph is G (the underlying graph of the labeled graph (G, σ_{L_i})) and with:

- $X = L_i$, $Y = L_o$,
- $Q = \mathcal{L}_v$, $S = \mathcal{L}_e$, with $\mathcal{L}_v$ the labels from $\mathcal{L}$ assigned to the vertices in the labeled graph, $\mathcal{L}_e$ the labels from $\mathcal{L}$ assigned to the edges in the labeled graph and $\mathcal{L} = \mathcal{L}_v \cup \mathcal{L}_e$,
- $I : L_i \to \mathcal{L}_v$, such that: $\forall\, x \in L_i$, $I(x) = \lambda(x)$,
- $\iota : L_i \to \mathcal{L}_e$, such that: $\forall\, x \in L_i$, $\iota(x) = \lambda(x)$,

- $\delta = \phi^{-1}(P)$,
- $O : \mathcal{L}_v \to L_o$, such that: $\forall\, y \in \mathcal{L}_v$, $O(y) = \pi(y)$, and
- $\omega : \mathcal{L}_e \to L_o$, such that: $\forall\, y \in \mathcal{L}_e$, $\omega(y) = \pi(y)$.

We can consequently conclude that representing a realization of a task with a graph relabeling system as a computation of a (mediated) population protocol is possible under some conditions. Whereas, representing the execution of a (mediated) population protocol as a realization of a task with a graph relabeling system is unconditioned.

3.2 From Population Protocols to Anonymous Asynchronous Message Passing

Angluin et al. [5, 4] do not specify, while describing the pairwise interactions in their model, how the communicating pairs are synchronized to simultaneously and instantaneously exchange their respective states, how the states are actually exchanged, etc. Tasks with graph relabeling systems are like the population protocols in the sense that they also make abstraction about this aspect of the communication between the processes. However, works in [14, 15, 51] established a bridge from graph relabeling systems to asynchronous message passing. They describe each synchronization needed for the application of a relabeling rule ($LC0$, $LC1$ or $LC2$) as a procedure based on message passing.

We proceed with a similar reasoning. Indeed, as population protocols were designed for networks such as sensor networks, we suggest to describe them through another model that is less theoretical and abstracted and that is more "natural" and suitable for such networks, with only some minimalistic hypotheses. This model consists on anonymous asynchronous message passing in networks with port numbering.

3.2.1 The Message Passing Model

A distributed system is formed by a collection of processes and a communication subsystem [67]. In case of message passing model, the communication subsystem consists on exchanging messages between the processes via the communication links. The processes use therefore specific primitives: send and receive. The send primitive enables a process to send a given message to a destination and the receive primitive allows a process to receive a message from a source. The sources and destinations of the messages can be named directly using names (or identities) of the processes, this is called direct naming. They can also be indirectly named using channels and ports numbers. A direct (respectively indirect) naming can be symmetrical where both the sender and receiver specifies the corresponding process (respectively channel), or asymmetrical where the receiver is able to receive messages from any process (respectively channel).

There are two types of message passing systems: synchronous and asynchronous [12]. When it is synchronous, the execution is partitioned into rounds. In each round, messages are sent and received by the corresponding processes and according to the received messages, each process executes a corresponding computation step. However, if the system is

asynchronous, then there is no fixed upper bound of the delivery time of messages, and also of the time separating two consecutive computation steps of a process.

A distributed system based on message passing is usually represented by a simple connected graph where an undirected edge represents a bidirectional communication link while a directed edge represents a unidirectional link. Now, if we detail a bidirectional link existing between any two processes u and v, in case of asynchronous message passing, then it will be as depicted in the following picture (Figure 3.12):

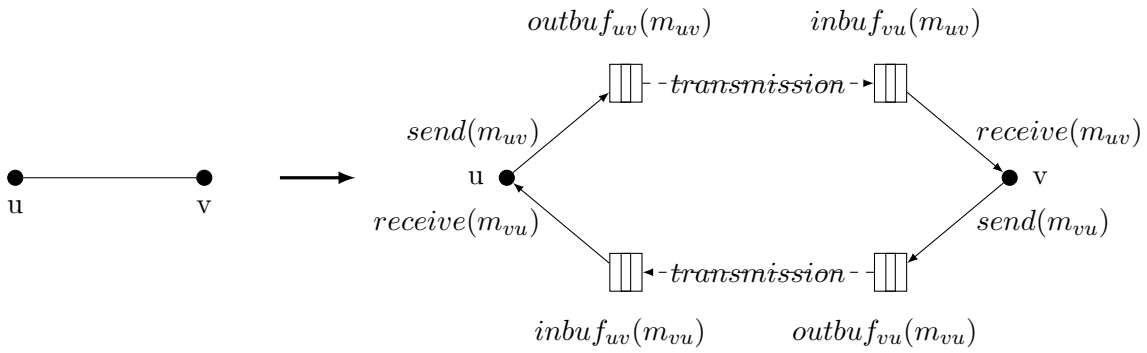

Figure 3.12: Zoom on a bidirectional communication link in asynchronous message passing system

A bidirectional link, between two processes u and v, allows u (respectively v) to send messages to v (respectively u) and receive messages from it.

When u sends a message m_{uv} to the node v, this message is placed in the out-buffer $outbuf_{uv}$ of the process u. The out-buffer $outbuf_{uv}$ contains the messages sent by u to v and that are still not delivered yet. Once, the message delivered to v, it is placed on the in-buffer $inbuf_{vu}$ that contains the messages coming from u and not yet processed by v. When v receives the message m_{uv}, by dequeuing it from $inbuf_{vu}$, it executes the corresponding computation step. The buffers are supposed to behave as FIFO queues.

In synchronous systems, we will suppose that the buffers are not needed anymore. We suppose that the transmission is instantaneous and that each sent message is supposed to be received and processed in the same round.

3.2.2 Anonymous Message Passing with Port Numbering

In anonymous networks, processes do not have unique identifiers and thus become indistinguishable. Consequently, in an anonymous network based on message passing, a process can not address a message to a specific neighbor based on its id. There are some studies that proposed an approach to break this symmetry by according names to the ports of each process. Accordingly, a process can address a message to a given neighbor by specifying, not its corresponding id, but the port's name associated to the link connecting them.

Port Numbering of Graphs

We here consider two existing works from literature. Each depends on the graph that models the network.

Port Numbering of Undirected Graphs In [70], Yamashita et al. consider anonymous networks represented by undirected, simple, connected graphs. They propose to attribute names to the ports of these networks according to a **local edge labeling** described as follows:

Definition 3.7. (**Local Edge Labeling**) Let $G = (V, E)$ be a simple, connected, undirected graph. Let v be any vertex in V and let f_v be a bijection from the set of edges, incident to v, to the set of positive integers $\{1, 2, \ldots, d(v)\}$. Accordingly, $f_u(u, v) = x$ implies that x is the name of the port of u that corresponds to the link (u, v). A local edge labeling, or port numbering, of G is the set of functions $f = \{f_v \,|\, v \in V\}$. This labeling may assign two different numbers to the two endpoints of a given link $(u, v) \in E$, which corresponds to $f_u(u, v) \neq f_v(u, v)$.

Figure 3.13 is an example of port numbering of an undirected graph.

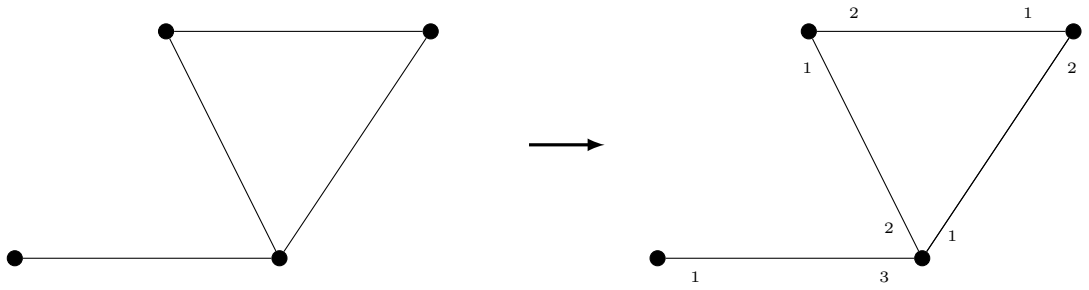

Figure 3.13: Local port numbering of an undirected graph

Port Numbering of Directed Graphs In [21], Boldi et al. consider the case of anonymous networks that are represented by directed graphs. They introduce different models of port numbering to allow each process to distinguish which port is associated to a given link.

In fact, the processors can have **output port awareness** traduced by a local output labeling of their output arcs according to the following definition.

Definition 3.8. (**Output Port Awareness**) Let $G = (V, A)$ be a simple, connected directed graph. For each vertex $v \in V$ with output degree $d^+(v)$, a local output labeling denoted $f_{out(v)}$ assigns to each outgoing arc of v a distinct number from the set $\{1, 2, \ldots, d^+(v)\}$.

Similarly, processors can also have **input port awareness**, which is a local input labeling $f_{in(v)}$ assigning to each input arc of any vertex v from G with input degree $d^-(v)$, a distinct number from the set $\{1, 2, \ldots, d^-(v)\}$. An example is illustrated in Figure 3.14.

Boldi et al. also introduced the model of **complete port awareness**. This model corresponds to the case where:

- the graph representing the network is symmetric,
- the processors have both input, and output, port awareness, and

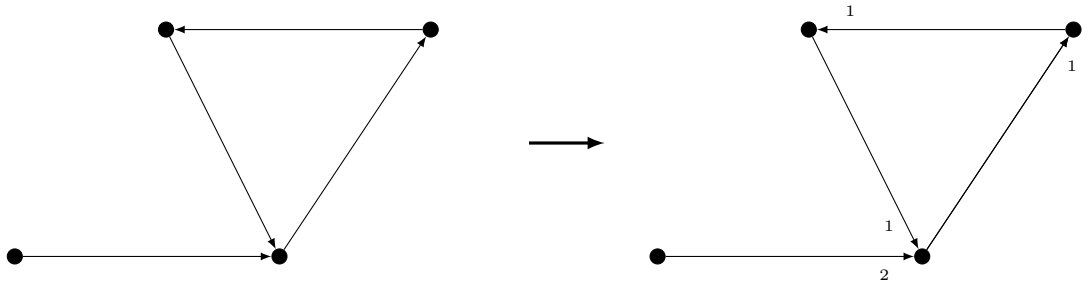

Figure 3.14: Attributing input port awareness to a directed graph

- each process assigns the same labeling to each pair of symmetric input/output arcs, that is: for any two neighbors nodes u and v, $f_{out(u)}(u, v) = f_{in(u)}(v, u)$.

Figure 3.15 shows an example of attributing complete port awareness to the processes of a network.

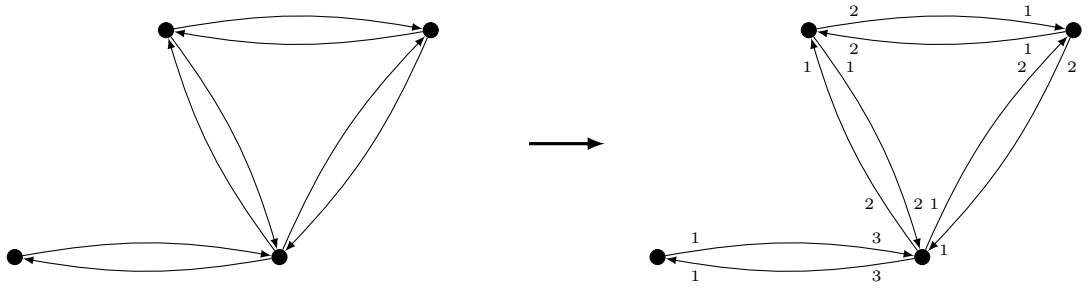

Figure 3.15: Attributing complete port awareness

We can notice that the model of Yamashita et al. [70], of local port numbering, corresponds to the complete port awareness model.

In fact, as depicted in Figure 3.16, any undirected graph $G = (V, E)$ corresponds to the symmetric digraph $Dir(G) = (V(Dir(G)), A(Dir(G)))$ where: $V(Dir(G)) = V$, and $\forall v \in V(Dir(G))$, $d^-(v) = d^+(v)$, and for each edge (u, v) from G two arcs (u, v) and (v, u) in $A(Dir(G))$ are associated such that: $s(u, v) = u$ and $t(u, v) = v$, and $s(v, u) = v$ and $t(v, u) = u$. For any two neighbors u and v from V, $f_u(u, v) = x$, with $x \in \{1, 2, \ldots, d(u)\}$, implies that $f_{out(u)}(u, v) = f_{in(u)}(v, u) = x$.

3.2.3 The Population Protocol Model versus the Anonymous Asynchronous Message Passing Model

We should mention that, in the sequel and in the context of population protocols, a sender can stand for the initiator of an interaction, as well as for the responder. The same holds for a receiver. Indeed, the initiator and the responder, both, play these two roles at the same time as, when exchanging their states, each sends its state and receives the state of the other agent. In this same context, messages stand for the exchanged states.

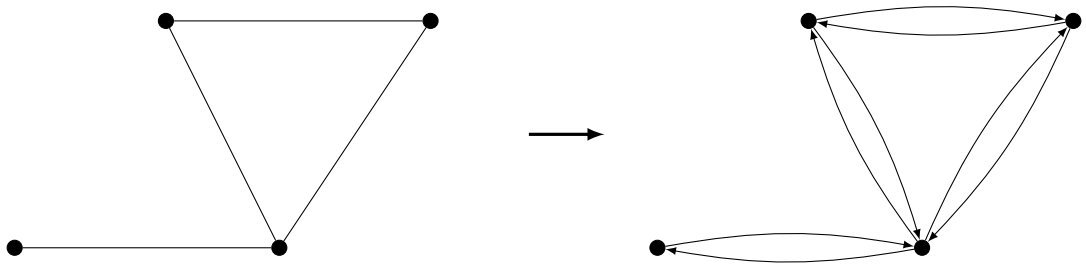

Figure 3.16: From an undirected to a directed graph

In the population protocol model, as well as in the anonymous asynchronous message passing model, the agents and the processes do not have unique identifiers. However, in the anonymous asynchronous message passing model, in spite of the anonymity of the processes, a node can distinguish its neighbors thanks to port numbering. Consequently, a node can address a message to a specific neighbor. This is not the case of the population protocol model. In fact, with respect to this aspect of interaction, the population protocol model is weaker compared to the anonymous asynchronous message passing model. It supposes that a sender is not able to direct a message to a specific receiver, and the receiver also has no idea about the source (the sender) of the message it receives.

On the other hand, regarding another aspect of the communication, the population protocol model is stronger as it supposes that the interaction between the communicating pair of agents is instantaneous and bidirectional, in a population that communicates asynchronously.

When Angluin et al. [9] introduced the one way population protocols model (see Section 2.2.1), they presented it as being the compromise between these two models. The one way population protocol model is a restriction of the pairwise interactions of the population protocols to one way communication. As a result, an initiator becomes unable to update its state according to the state of its responder. This model is also an adaptation of the anonymous asynchronous message passing model where processes have no control to choose the destination of their sent messages. Angluin et al. also supposed that, in this model, the initiator corresponds to the sender of a message and the responder stands for the receiver.

3.2.4 From a Computation of a Population Protocol to an Anonymous Asynchronous Distributed Algorithm based on Message Passing

In this section, we propose another solution which is also a compromise between the population protocol model and the anonymous asynchronous message passing. However, unlike the one way population protocols, it preserves the concept of bidirectional and simultaneous pairwise exchange of the population protocol. We mentioned above that Angluin et al. suppose that neither the initiator, nor the responder, has a control to direct its state to a specific neighbor. This reflects which is described in [9], and implicitly adopted in the structure of the population protocols: the strongly anonymous message passing. Though, these two interacting agents are supposed to mutually exchange their corresponding states. However, this is still a too theoretical assumption. Such mutual

bidirectional exchange requires a minimum knowledge, that allows a pair of agents to distinguish each other, during their interaction, and to direct their respective states to each other. We thus think about port awareness as being this required knowledge.

In the sequel, we propose to simulate the population protocols over an anonymous asynchronous system based on message passing with port awareness. First, we start by associating to any interaction graph of a population that runs a population protocol the corresponding communication graph that describes the information (messages) flow. Then, we describe the computation of a population protocol as an execution of an anonymous asynchronous distributed algorithm based on message passing over this obtained graph.

It is important to mention that the approach we present in this section holds, not only for the population protocol model, but also for all its extensions that preserve the pairwise interactions. We just need, in this case, to add some additional details that depend on the model.

The Network Communication Graph Associated to the Interaction Graph

Let $\mathcal{P}$ be a population that executes a population protocol $\mathcal{A}$ over a communication graph $G = (V, E)$. A possible interaction, between an initiator agent u and a responder agent v, is represented by the directed edge (u, v) in G. Yet, an interaction between this pair of agents implies that they both exchange their respective states. This supposes that information flows in both directions. Hence, the direction attributed to the edge (u, v) does not imply that the link is unidirectional, but is only required to break the symmetry between the interacting agents.

We propose now to represent the interaction graph as a graph that describes the messages flow in the network formed by the population. This graph will respect the convention introduced in Section 3.2.1 of the representation of networks based on message massing where: a directed edge represents a unidirectional communication link, while an undirected edge represents a bidirectional one. We call this graph: the network communication graph.

Consequently, we associate to any interaction graph $G = (V, E)$ the corresponding network communication graph $G' = (V', E')$. G' is a simple undirected graph where: $V' = V$, and for each arc $(u, v) \in E$, we associate the undirected edge (u, v) in E'.

As a result, any directed edge, linking an initiator to a responder in the graph G, is represented by an undirected edge in G' as depicted in Figure 3.17.

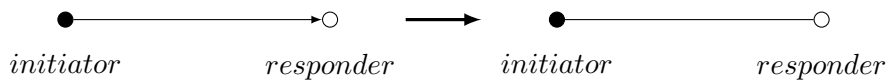

Figure 3.17: The representation of a directed edge from the interaction graph G in its associated network communication graph G'

Port Labeling with Roles of the Network Communication Graph While constructing the graph G' based on G, we should keep track of the initial directions of the edges of G that are essential in breaking the symmetry of each interacting pair of nodes.

For this purpose, we propose to assign labels to the ports of G'. We thus define, SB (SB standing for symmetry breaking) which is a local edge labeling of G' such that: for any vertex v from V', for any link incident to v, SB_v indicates the role associated to v if an interaction takes places over this link.

The existence of an arc (u, v) in E implies that, if u interacts with v then, u will play the role of the initiator and v the responder. Accordingly, $\forall\, u,\, v \in V'$, the labellings SB_u and SB_v of the incident edges of u and v in the graph G' are as follows:

- If $(u, v) \in E$ and $(v, u) \notin E$ then: $SB_u(u, v) = i$, and $SB_v(u, v) = r$. In Figure 3.18, we consider an arc (u, v) from G fulfilling this previous condition, and we illustrate its corresponding representation in the network communication graph G' with SB_u and SB_v labellings.

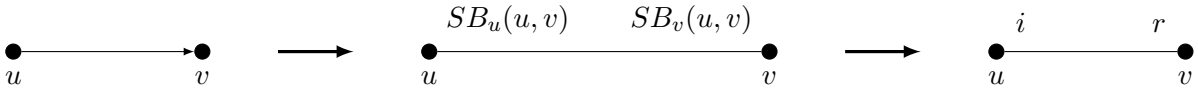

Figure 3.18: Representation of (u, v) from G in G' with SB_u and SB_v labellings

- Otherwise, if $(u, v) \in E$ and $(v, u) \in E$ then: $SB_u(u, v) = SB_v(u, v) = *$. The labeling $*$ implies that the vertex u, when interacting with v, can play either the role of the initiator or the responder. The same holds for v.

We show, in Figure 3.19, an example of an interaction graph G and its corresponding network communication graph G' with SB labellings of its ports.

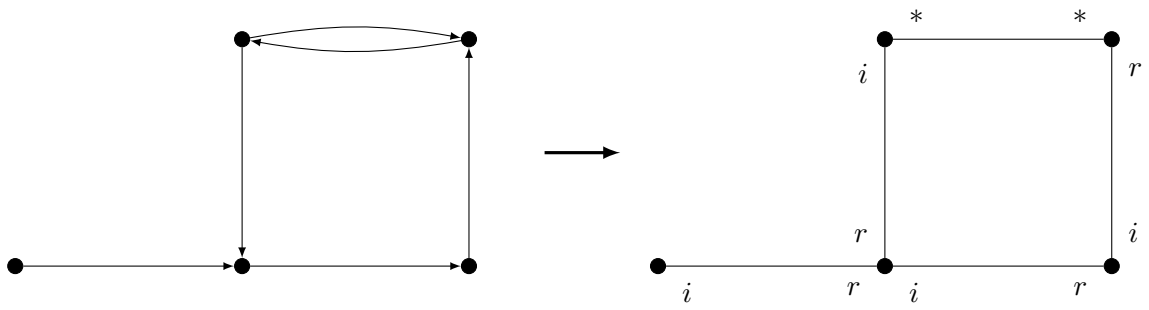

Figure 3.19: From the interaction graph G to its associated network communication graph G' labeled with SB

Port Numbering of the Network Communication Graph As we mentioned above, a less abstracted and more realistic execution of population protocols requires a minimum knowledge that enables an agent to distinguish the neighbor it will interact with. Hence, in addition to the SB labellings, we propose to assign numbers to the ports of the graph G' with a local edge labeling as described by Yamashita et al in [70].

Accordingly, $\forall\, v \in V'$, we define a local input labeling f_v that assigns, to each incident link of v, a distinct number from the set $\{1, 2, \ldots, d(v)\}$.

As a final result, the ports in G' are going to have both SB labellings and port numbering. Therefore, $\forall u \in V'$, $\forall v \in N_{G'}(u)$, we associate to each port of u the following labeling $(f_u(u,v), SB_u(u,v))$.

We continue now with the example presented in Figure 3.19, and assign, in addition to the roles labellings of the ports, port numbering. The obtained result is depicted in Figure 3.20.

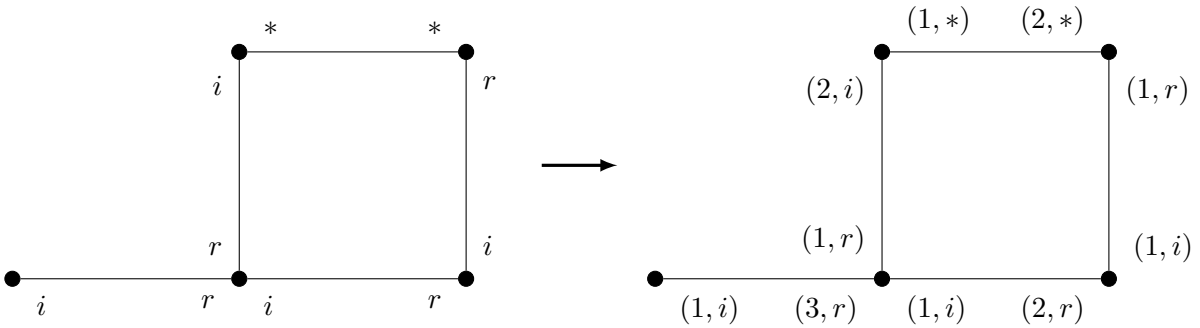

Figure 3.20: Assigning port numbers to the network communication graph G' labeled with SB

Remark 3.1. We would mention that the graph G' can also be represented as a symmetric digraph where: $V' = V$, and for each arc $(u,v) \in E$, we associate two arcs (u,v) and (v,u) in E', such that $s(u,v) = u$ and $t(u,v) = v$, and $s(v,u) = v$ and $t(v,u) = u$. Assigning numbers to its ports will be according to the to complete port awareness model of Boldi et al. [21].

Simulating the Pairwise Interactions

We now propose to map the computation of a population protocol to the execution of a distributed algorithm in an anonymous asynchronous system based on message passing.

Let $\mathcal{A} = (X, Y, Q, I, O, \delta)$ be any population protocol, and let $\mathcal{P}$ be any population executing $\mathcal{A}$ over an interaction graph $G = (V, E)$. If we proceed as described in the previous section, we can obtain the network communication graph $G' = (V', E')$, associated to the interaction graph G, and labeled with SB and port numbers. The graph G' is, in fact, the representation of the anonymous asynchronous system over which the distributed algorithm simulating $\mathcal{A}$ is going to run. This distributed algorithm is a collection of identical local algorithms that we denote by *Algo*.

The agents running the protocol $\mathcal{A}$ are the processes forming G' and executing the *Algo* algorithms. And, the messages that are going to be transmitted over G' are the states exchanged in $\mathcal{A}$. A local algorithm *Algo* should preserve the bidirectional and simultaneous exchange of these states in spite of the asynchrony of the network.

To be able to communicate simultaneously, the pair of interacting agents should be synchronized. In literature, there are procedures designed to synchronize processes in anonymous asynchronous systems based on message passing with port awareness. Further details about such procedures are going to be provided in the next chapter. But

meanwhile, we make an abstraction about the implementation details of these procedures. We just suppose that an algorithm *Algo*, run by an agent u, requires a synchronization procedure that we denote $TrySynchronization(u)$.

This function returns for each agent u two parameters. These parameters are different from 0, only if the agent u succeeds in having a synchronization with one of its neighbors. Let $c(u)$ denote the neighbor with which u is synchronized. The first output of $TrySynchronization(u)$ corresponds to $f_u(u, c(u))$, which is the port number associated to the edge linking this agent u with its neighbor $c(u)$. The second parameter corresponds to $SB_u(u, c(u))$, which is the role associated to the agent u during this interaction.

Now, once the pair of interacting agents is synchronized, roles are attributed according to the labeling SB. Both agents will play the role of a sender and of a receiver at the same time. The initiator and the responder will send their respective states to each other at the same time, and each of them will consequently receive the state of the other. Each of these processes will update its state according to the transition function and with respect to its role. Once the update is done, the current synchronization is ended and a trial for a new one is initiated.

Accordingly, the computation of a protocol $\mathcal{A} = (X, Y, Q, I, O, \delta)$ in a population $\mathcal{P}$ whose interaction graph is G, corresponds to the execution of the following local algorithm *Algo* (Algorithm 1) by each process u in the network communication graph G' associated to G. $\forall u \in V'$, $state_u$ denotes the state of the process u, standing for agent u, in this algorithm. The value of this state is initialized according to the input of the agent u in the population $\mathcal{P}$, and to the input function I of the protocol $\mathcal{A}$.

We can notice that, once the initialization is done, the remaining code in Algorithm 1 belongs to an infinite loop. In fact, an agent executing a population protocol does not halt, so the process representing it will infinitely execute this code.

Algorithm 1 The Algorithm *Algo*

Require: $TrySynchronization(u)$
1: $state_u \leftarrow I(input_u)$;
2: **loop**
3: $port \leftarrow 0$;
4: $role \leftarrow 0$;
5: $(port, role) \leftarrow TrySynchronization(u)$;
6: **if** $(port \neq 0)$ **then**
7: Send$(state_u, port)$;
8: Receive$(state_{c(u)}, port)$;
9: Apply a transition rule (if applicable) of the transition function of $\mathcal{A}$ with respect to the attributed role $role$.
10: **end if**
11: **end loop**

Example 3.4. Mapping the Computation of the Population Protocol Threshold to the Execution of a Distributed Algorithm in an Asynchronous Anonymous

System based on Message Passing. We propose to map the computation of the population protocol *Threshold* (see Section 2.1) to the execution of a distributed algorithm in an asynchronous anonymous system based on message passing with complete port awareness. This distributed algorithm, corresponding to this protocol, consists on a set of local algorithm *Algo*, described by Algorithm 2, and that are run by each process u in this system. A process u, as we mentioned above, stands for agent u in the population.

Algorithm 2 Algorithm Simulating the *Threshold* Population Protocol

Require: $TrySynchronization(u)$
1: $state_u \leftarrow I(input_u)$;
2: **loop**
3: $port \leftarrow 0$;
4: $role \leftarrow 0$;
5: $(port, role) \leftarrow TrySynchronization(u)$;
6: **if** $(port \neq 0)$ **then**
7: Send$(state_u, port)$;
8: Receive$(state_{c(u)}, port)$;
9: **if** $(role = i)$ **then**
10: **if** $(state_u + state_{c(u)} < 5)$ **then**
11: $state_u \leftarrow state_u + state_{c(u)}$
12: **else**
13: $state_u \leftarrow 5$
14: **end if**
15: **else**
16: **if** $(state_u + state_{c(u)} < 5)$ **then**
17: $state_u \leftarrow 0$
18: **else**
19: $state_u \leftarrow 5$
20: **end if**
21: **end if**
22: **end if**
23: **end loop**

We would just mention that, for a sake of clarity and for a simpler representation, we replaced the set of states Q of the *Threshold* population protocol, by $Q = \{0, 1, 2, 3, 4, 5\}$ where: the state 0 stands for the state q_0, 1 for q_1, etc.

Initially, a process u receives its state according to the input of agent u and to the input function I. This state can be either 0 or 1. Then, once the loop is initiated, the attributes *port* and *role* of the process u are initialized to 0. The function $TrySynchronisation(u)$ is executed. If the value of the parameter *port* returned by this function is different from 0, then *role* will also be different from 0. This implies that u is synchronized with a neighbor $c(v)$ with which is can interact via the link whose number is *port* and according to *role*. The process u sends its state $state_u$ via the link labeled *port*, and receives the state of its neighbor $state_{c(v)}$ via the same link. The process u, according to its role, checks if there is a transition rule, according to the transition function δ of the *Threshold* protocol, that should be applied. If the process u is the initiator in this interaction, and if the sum of the states is less than 5, its state receives the sum of these states, else its state becomes

5. However, if the process u is the responder in this interaction, and if the sum is less than 5, its state is reinitialized to 0, else it is updated to 5. This loop will be repeated infinitely. When the computation stabilizes, if there is at least 5 processes that started with state 1 (that is there at least 5 sick agents initially), the state 5 will be spread in the whole network. Otherwise, the state that represent the sum of the "ill" processes (which is for sure less than 5 in this case) will be a swapping state over the network.

3.3 Conclusion: Bridges between Models

Thanks to the comparative studies and mapping approaches presented in this chapter, we established some links between three different models that are: the (mediated) population protocols, tasks with graph relabeling systems and the anonymous asynchronous message passing. We summarize the result in Figure 3.21.

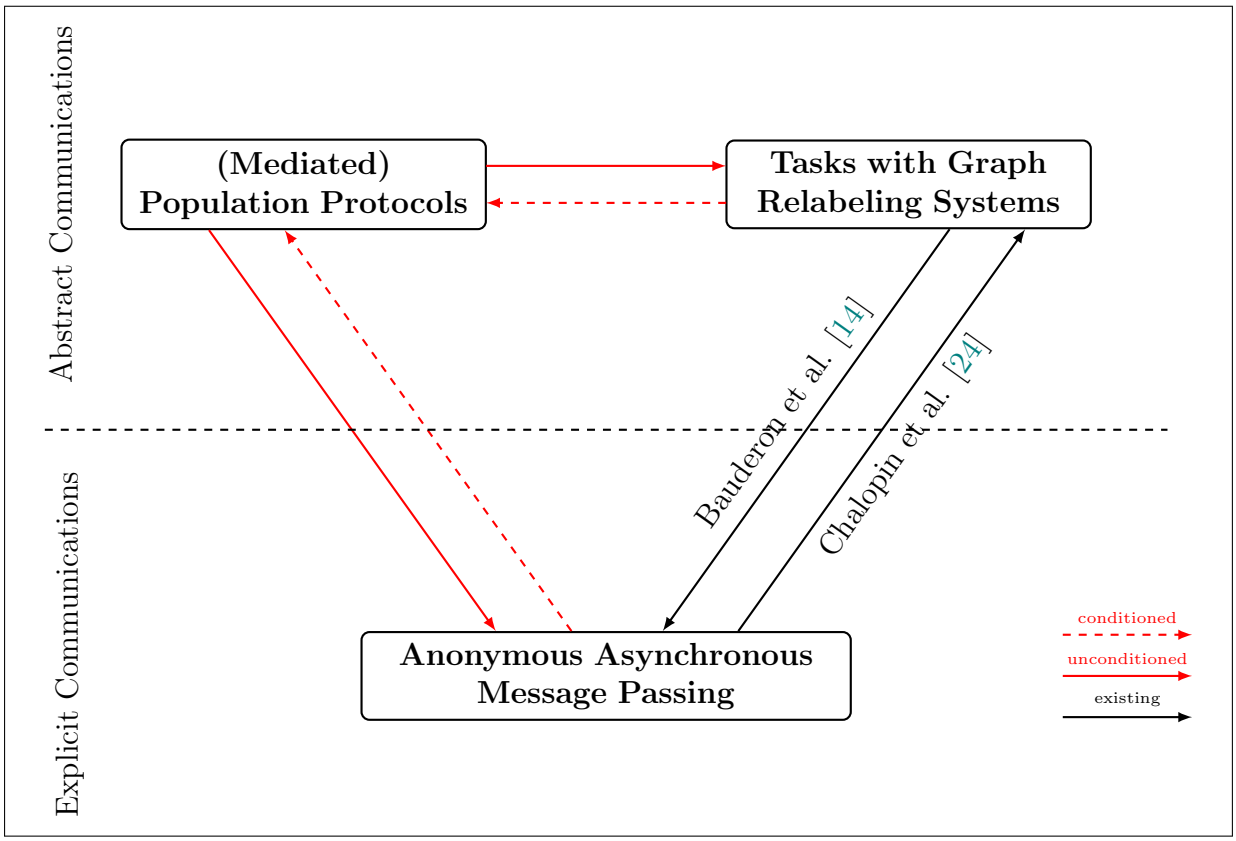

Figure 3.21: Possible Mappings

The bridge between the last two models, represented by the two black arcs in Figure 3.21, already exists in literature.

Indeed, in one hand, Bauderon et al. [14] have defined the link between the second and the third one by expressing distributed algorithms defined by graph relabeling systems with those based on anonymous asynchronous message passing.

On the other hand, Chalopin et al. [24] worked on expressing distributed anonymous

asynchronous algorithm based on message passing model as graph relabeling systems.

For our part, we first designed a mapping approach between the first and the second model. These two models, both, use abstract communication models. The red arc, linking the first model to the second one, traduces the possibility of mapping the computation of a (mediated) population protocol to a realization of a task with a graph relabeling system. Whereas, the reverse is not always possible, and we represent it in the figure by a dashed red arc. In fact, we proved that mapping a realization of a task to the computation of a (mediated) population protocol is only possible under some conditions.

Then, we established a bridge (represented by a red arrow) between the first and the third model that uses explicit communications. This allows us to represent the computation of a (mediated) population protocol as a distributed algorithm, based on message passing, and running in an anonymous asynchronous system. Hence, the possibility of a more realistic execution of such protocols can be illustrated. Concerning the mapping of a distributed asynchronous anonymous algorithm, based on message passing, to a computation of a (mediated) population protocol, we think it is possible under some specific conditions. However, we do not explore them in this work.

In the following chapter, we are going to take advantage from these established links between models, and more specifically, from the bridge between the (mediated) population protocols and the anonymous asynchronous message passing. In fact, we are going to propose some scheduling procedure designed for populations protocols and described through anonymous asynchronous distributed algorithms based on message passing.

Chapter 4

The Handshake Scheduler: a Distributed Fair Scheduler by Randomized Handshakes

A scheduler in a distributed system is a theoretical entity that simulates the environment and orchestrates the actions of the processes in the system. Likely, a scheduler in the context of population protocols decides which pair of agents is going to communicate at each step of the computation of the protocol.

Based on the mapping approach that we established in the previous chapter, we propose an implementation based on message passing of the Random Scheduler, a scheduler proposed by Angluin et al. for the population protocol model.

We also introduce, in this chapter, a new fair probabilistic scheduler designed for the population protocol model, and also for any other model extending it, as long as it preserves the pairwise interactions concept. We call this scheduler the Handshake Scheduler, and denote it by the *HS Scheduler*. It is based on randomized handshakes. Thus, unlike the already proposed schedulers, the *HS Scheduler* can allow, at each step of the computation, to more than only one pair of agents to communicate. In fact, it synchronizes pairs of agents forming matchings over the interaction graph of the population.

We start thus by giving an overview of the existing schedulers and fairness conditions in distributed systems, and especially in the context of population protocols. We focus on the Random Scheduler for which we propose an implementation based on message passing. We then give a definition of the model of the new scheduler, that we introduce, as well as its implementation. This implementation is based on a randomized handshake algorithm. We finally prove, through an analysis study, that this scheduler is fair with probability 1. These results appeared in [60] and in [61].

4.1 Schedulers in Distributed Systems

The execution of a distributed computing system may be influenced by many facts: the asynchrony of the processes, their mobilities, their energies, ... in short the environment as a whole. In fact, the environment has an impact on the selection of the processes that can be activated, on the order of their activation and also on the interactions that can take place between these different processes. In literature, these facts, as well as their caused nondeterminism, are captured by an abstraction which is the scheduler. The scheduler can also be called the demon, or the adversary, when it refers to a hostile environment.

A possible definition of a scheduler can be the following:

Definition 4.1. (**Scheduler**) Given a protocol that runs over a distributed system, a scheduler can be considered as a function that assigns to this protocol a subset of all its possible executions.

The scheduler orchestrates tasks and actions in a distributed system by determining, at each step of the computation, which enabled processes are going to be executed. A process is considered as enabled if, given its state, and sometimes the states of its neighbors, it is able to make a computation step (an action): it is ready to be activated.

Basically, there are two main types of schedulers for distributed systems [38]:

- **Central (or Sequential) Schedulers** A central scheduler allows to only one enabled process to perform a computation step and wait until it finishes before proceeding in choosing and activating another enabled process [30].

  A central scheduler serializes a distributed computation.

- **Distributed Schedulers** A distributed scheduler can pick any subset of the enabled processes to be privileged to make a computation step concurrently.

While presenting a taxonomy of the existing types of schedulers in [31], the authors introduced the following concept of k-centrality, that allows a more specified classification:

Definition 4.2. (**k-centrality**) A scheduler is k-central if it allows only the processes separated by at least k hops to be simultaneously scheduled.

Consequently, with respect to this definition, a **0-central** scheduler corresponds to the distributed scheduler as, whenever is the distance separating the enabled processes, they can be allowed to be executed simultaneously. Whereas, the **diam(G)-central**, where $diam(G)$ denotes the diameter of the graph G representing the distributed system, is the central (or the sequential) scheduler.

In addition to this classification, schedulers can also be characterized differently.

The distributed scheduler that picks all the enabled processes at each step is called the **synchronous** scheduler [59].

When the scheduler assigns, to each enabled process, a probability p to be chosen, it is called a **probabilistic** scheduler [17].

4.2 Fairness

The correctness of a distributed algorithm, running over a distributed system, usually depends on the presumed scheduler. To guarantee the success of the task of the distributed system, the scheduler should be **computation-friendly**: it should not avoid any possible step forever. Therefore, the scheduler should satisfy a fairness property [38].

The population protocols are distributed algorithms running over distributed systems (the populations). Hence, their executions should also satisfy some fairness condition. In this context, Angluin et al. [4], Fisher et al. [36], and Chatzigiannakis et al.[25] gave formal definitions of the different possible fairness conditions.

4.2.1 Fairness Condition According to Angluin et al.

When they introduced the population protocol model, Angluin et al. assumed a fairness condition that guarantees that any possible execution may occur. Consequently, any live-lock that can be caused by looped executions is avoided.

More formally, this fairness is characterized by the following condition [4]:

Definition 4.3. (**Fairness**) Let $EX = C_0, C_1, \ldots, C_i, \ldots$ be an infinite execution such that for each i, $C_i \rightarrow C_{i+1}$. EX is fair if for every possible transition $C \rightarrow C'$, if C occurs infinitely often in this execution, then C' occurs infinitely often.

4.2.2 Fairness Condition According to Fisher and Jiang

Fisher and Jiang presented in [36] more detailed definitions of the different possible fairness conditions for scheduling in population protocols. They based their approach on extended definitions of some basic elements in a population protocol.

First, the transition function is supposed to be as the one defined for population protocols with stabilizing inputs (see Section 2.2.2), and which is :

$$\delta : (Q, X) \times (Q, X) \to (Q, Q).$$

An input assignment is defined as being the mapping from V to X. An action is a pair (r, e) where: r is a transition from the transition function δ, and e is an edge from the interaction graph over which the protocol runs.

Now, let C and C' be two configurations. Let α be an input assignment, and let $e = (u, v)$ be an edge from the interaction graph. Let $\sigma = (r, e)$ be an action with $r = ((p, x), (q, y)) \to (p', q')$ and $e = (u, v)$.

We can say that σ is enabled in (C, α) if: $C(u) = p$, $\alpha(u) = x$, $C(v) = q$, and $\alpha(v) = y$. (C, α) goes to C' via σ, denoted $(C, \alpha) \xrightarrow{\sigma} C'$, if: σ is enabled in (C, α), $C'(u) = p'$, $C'(v) = q'$ and $C'(w) = w$, $\forall\, w \in V - \{u, v\}$.

If for some action σ, we have $(C, \alpha) \xrightarrow{\sigma} C'$, then we can say that: (C, α) goes to C' in one step, and that σ is a taken action.

According to these definitions, the authors presented, formally, the following fairness conditions of different strengths, with respect to an execution $EX = (C_0, \alpha_0)$, $(C_1, \alpha_1), \ldots, (C_i, \alpha_i), \ldots$:

Definition 4.4. (**Strong Global Fairness**) Let C and C' be two configurations, and let α be an input assignment such that $(C, \alpha) \to C'$. If $(C, \alpha) = (C_i, \alpha_i)$ for infinitely many i, then $(C_i, \alpha_i) = (C, \alpha)$ and $C_{i+1} = C'$ for infinitely many i.

In other words, this means that the step $(C, \alpha) \to C'$ is taken infinitely many times in EX.

Definition 4.5. (**Strong Local Fairness**) For every action σ, if σ is enabled in (C_i, α_i) for infinitely many i, then $(C_i, \alpha_i) \xrightarrow{\sigma} C_{i+1}$ for infinitely many i. This means that σ is taken infinitely many times in EX.

The fairness condition can be local or global. The fairness condition is **local** if, it asserts that each action that can be taken infinitely often is actually taken infinitely often. However, it is **global** if it asserts that each step that transitions the system from a configuration C to another configuration C', and that can be taken infinitely often, will actually be taken infinitely often.

The authors stated that global fairness implies local fairness [36].

In addition to strong fairness, there are also weak forms of fairness:

Definition 4.6. (**Weak Global Fairness**) For every C, α and C' such that $(C, \alpha) \to C'$, if (C, α) occurs infinitely often in EX, then C' occurs infinitely often in EX.

Definition 4.7. (**Weak Local Fairness**) For every action σ, if σ is enabled infinitely often in EX, then there exist C, α, C' such that $(C, \alpha) \xrightarrow{\sigma} C'$, (C, α) occurs infinitely often in EX, and C' infinitely often in EX.

We can notice that, unlike the **strong** fairness that requires that particular steps should occur infinitely often in the execution, the **weak** fairness only insists that the resulting configurations from these steps occur infinitely often.

According to these definitions, the fairness condition introduced by Angluin et al. can be classified as being a weak global fairness. Indeed, Angluin et al. assume that local fair scheduling rarely exists in real distributed systems, and that global fairness traduces better realistic scheduling for such systems [11].

4.2.3 Fairness Condition According to Chatzigiannakis et al.

Unlike the two previous works, Chatzigiannakis et al. focused, not on the fairness of an execution of a population protocol, but on the properties that a probabilistic scheduler for population protocols should satisfy to be fair with probability 1 [25].

Therefore, they started by introducing a formal definition of a probabilistic scheduler with respect to (w.r.t) a transition graph. We should mention that a transition graph can be described as follows:

Definition 4.8. (**Transition Graph**) The transition graph $G(\mathcal{A}, \mathcal{P})$ of a protocol $\mathcal{A}$ running in a population $\mathcal{P}$ is the directed graph whose nodes are all possible population configurations and whose edges are all possible transitions between these configurations.

A transition graph will be denoted, in what follows, by $T(\mathcal{A}, G) = (V(T), E(T))$ where G is the communication graph over which the protocol $\mathcal{A}$ runs.

Hence, the formal definition of a probabilistic scheduler with respect to a transition graph is:

Definition 4.9. (**Probabilistic Scheduler**) A probabilistic scheduler, w.r.t a transition graph $T(\mathcal{A}, G)$, defines for each configuration $C \in V(T)$ an infinite sequence of probability distributions of the form $(d_1^C, d_2^C, \ldots)$, over the set $\Gamma^+(C) = \{C' \mid C \to C'\}$, where d_t^C: $\Gamma^+(C) \to [0, 1]$ and such that $\Sigma_{C' \in \Gamma^+(C)}\, d_t^{C'} = 1$ holds for all t and C'.

In other words, for each configuration C from the transition graph, for any $t \geq 1$, if C is encountered for the t^{th} time, then there is a probability distribution d_t^C over $\Gamma^+(C)$ that specifies with which probability C will go to any configuration C' such that C' is reachable from C in one step.

Then, Chatzigiannakis et al. [25] introduced the consistency property of a probabilistic scheduler w.r.t. a transition graph and that can be defined this way:

Definition 4.10. (**Consistent Scheduler**) A probabilistic scheduler is consistent, w.r.t. a transition $T(\mathcal{A}, G)$, if for all configurations $C \in V(T)$, it holds that $d^C = d_1^C = d_2^C = \ldots$. This means that any time the configuration C is encountered, the scheduler chooses the next configuration with the same probability distribution d^C over Γ^+. This holds for all C, each with its own distribution.

Given this definition, we can consider that a consistent probabilistic scheduler w.r.t $T(\mathcal{A}, G)$ assigns a label to each directed edge of T. Each edge label is the probability to go

from the configuration representing the source of this directed edge to the configuration corresponding to the target of this edge.

More formally, this labeling can be a mapping P from $E(T)$ to $[0, 1]$, such that for any configuration $C_i \in V(T)$, $\sum_{C_j \in \Gamma^+(C_i)} P_{(C_i,C_j)} = 1$, where $P_{(C_i,C_j)}$ is the probability to go from the configuration C_i to C_j. These probabilities do not depend neither on time nor on the number of times a configuration is encountered.

Introducing these definitions allowed the authors to present the following theorem. This theorem states the conditions that should be satisfied by a probabilistic consistent scheduler to be fair with probability 1 [25].

Theorem 4.1. *(**Fair Scheduler with Probability 1**) Any consistent scheduler, for which it holds that $P_{(C_i,C_j)} > 0$, for any protocol $\mathcal{A}$, any communication graph G, and all configurations C_i, $C_j \in V(T(\mathcal{A}, G))$ where $C_i \to C_j$ and $C_i \neq C_j$, is fair with probability* 1.

In this work, we will rather focus on this last definition of fairness that concerns probabilistic schedulers.

4.3 Existing Schedulers for Population Protocols

The population protocol model is based on pairwise interactions. Scheduling will therefore consist on deciding which pairs of agents will communicate, and in what order. In this context, the enabled processes are not any more single processes, as it is usually presumed on distributed systems, but ordered pairs of processes able to communicate. Choosing an ordered pair of agents from a population, corresponds to choosing an ordered pair of neighbor nodes from the interaction graph (or in other words enabled edges from the interaction graph). Thus, an enabled process that could be picked by the scheduler in population protocols can also be described as being a directed edge from the interaction graph.

Different schedulers were proposed for population protocols (and that are also valid for any variant or extension of this model as long as it preserves the pairwise interactions concept). There are schedulers that act without having any knowledge of the protocol. Whereas, there are those that do know the protocol and take this knowledge into account.

Definition 4.11. (**Protocol Oblivious vs Protocol Aware Scheduler**) A scheduler is called protocol oblivious (or agnostic) if it constructs the interaction pattern without any knowledge on the executed protocol, and protocol-aware if it takes into account the information concerning the underlying protocol.

According to this classification, we list below the existing schedulers designed for population protocols.

4.3.1 Protocol Oblivious Schedulers

In [4], Angluin et al. presented the probabilistic population protocol model. This model corresponds to the population protocol model where interactions are scheduled by a probabilistic scheduler called the Random Scheduler.

The Random Scheduler

The Random Scheduler is protocol oblivious. Whatever the protocol run by the population is, this scheduler always acts similarly. All the pairs of agents have the same probability to interact.

At each step, the scheduler chooses independently, randomly and uniformly only one edge from the interaction graph, which is one ordered pair of agents. Once two agents are picked by the scheduler, they are allowed to interact. They communicate their corresponding states to each other, and then update them according to the transition function of the protocol.

Given this random, uniform and independent choice, Angluin et al. stated that the Random Scheduler is fair with probability 1 [4]. Also, Spirakis et al. proved that this scheduler is fair with probability 1 with respect to their fairness condition [25].

4.3.2 Protocol Aware Schedulers

Chatzigiannakis et al. proposed, in [25], three probabilistic schedulers that are, unlike the Random Scheduler, protocol aware. Yet, each has its own way of taking into account the population protocol that will be run over the population.

We suppose, in what follows, that $G = (V, E)$ represents the interaction graph over which the population protocol runs.

The State Scheduler

The State Scheduler proceeds as follows at each step of the computation of the protocol. It first chooses independently and uniformly at random an ordered pair of states (q_1, q_2) from all the interaction candidates of the current configuration. An ordered pair of states (q_1, q_2) is an interaction candidate under a configuration C if: $\exists\,(u, v) \in E$ such that $C(u) = q_1$ and $C(v) = q_2$. Then, the scheduler chooses, independently and uniformly at random, only one ordered pair of nodes related by an edge in E, and whose states are (q_1, q_2). These chosen nodes can thus communicate and update their states according to the transition function.

Chatzigiannakis et al. proved that this scheduler is fair with probability 1 [25].

Now, let consider the execution of the population protocol 3-States Approximate Majority (see Section 2.2) under the scheduling of the State Scheduler. We recall that the set of states of this protocol is $Q_{AM} = \{x, y, b\}$. Then, for any configuration C, the set of states that can be interaction candidates under C should be a subset of the following set $\{(x, x), (x, y), (x, b), (y, y), (y, x), (y, b), (b, b), (b, x), (b, y)\}$. The State Scheduler chooses independently and uniformly at random an ordered pair which is an interaction candidate under the current configuration. Let suppose that the chosen one is the pair (x, y). Now, from all the edges in the interaction graph whose extremities are nodes representing agents with states (x, y), the scheduler should pick uniformly at random only one of them. Then, the pairwise interaction can take place, and the states can be updated.

According to [25], the processing of this scheduler permits, more often, the choice of meaningful transitions that lead to the protocol's progress.

The Transition Function Scheduler

The protocol awareness of this scheduler is shown through the choices it makes concerning the communicating pairs. These choices are based on the transition function rules of the protocol. The scheduler chooses pairs that, when they communicate, will lead to the protocol progress: that is at least the state one of the initiator or the responder will modify its state, and it ignores all the transitions where no state changes.

To reach a configuration C' from a configuration C, the scheduler proceeds as follows. First, it picks independently and uniformly at random a pair $((q_1, q_2), (q'_1, q'_2))$ where (q_1, q_2) is an interaction candidate under C and $\delta(q_1, q_2) = (q'_1, q'_2)$. Then, it chooses independently and uniformly at random an ordered pair of nodes $(u, v) \in E$, whose states correspond to (q_1, q_2) under C. Then, to obtain C', the transition function is applied to (q_1, q_2) to update their states.

In case the Transition Function Scheduler can not find any pair able to lead to the protocol progress, it works like the Random Scheduler.

Let us take the example of the One Way Epidemic population protocol consisting of:

- $X_{OW} = Y_{OW} = \{1, 0\}$ with 1: the input (respectively output) of an infected agent and 0: the input (respectively output) of a non infected agent,
- $Q_{OW} = \{q_0, q_1\}$,
- $I_{OW} : 0 \to q_0$, $1 \to q_1$,
- $O_{OW} : q_0 \to 0$, $q_1 \to 1$, and
- $\delta_{OW}(q_1, q_0) = (q_1, q_1)$.

Usually non described transitions in a population protocol are those that are non effective (also called identity transitions). They are those that leave both the state of the initiator and that of the responder unchanged. For this protocol, they consist on:

$$\delta_{OW}(q_0; q_0) = (q_0; q_0),\ \delta_{OW}(q_0; q_1) = (q_0; q_1) \text{ and } \delta_{OW}(q_1; q_1) = (q_1; q_1).$$

We consider now the execution of this protocol under the Transition Function Scheduler. The only effective transition rule in this protocol concerns the pair of states (q_1, q_0). Thus, at each step of the computation, the Transition Function Scheduler checks if, among the pairs of states that are interaction candidates under the current configuration, there exist pairs (q_1, q_0). If it is the case, the scheduler picks independently and uniformly at random an ordered pair of nodes from all the pairs $(u, v) \in E$ such that, the state of u is q_1 and the state of v is q_0. Hence, the effective transition rule can be applied, and consequently the protocol progresses. Otherwise, the Transition Function Scheduler acts as the Random Scheduler by choosing independently and uniformly at random a pair of ordered neighbor nodes from G.

Chatzigiannakis et al. proved in [25] that this scheduler is fair with probability 1.

The Modified Scheduler

The Modified Scheduler is a slightly modified version of the State Scheduler. It distinguishes two classes of transition rules: a class of non effective transition rules and a class composed of the remaining rules. It selects, with a probability $1-\varepsilon$, rules from the class of identity rules and with probability $0 \leq \varepsilon \leq 1$ rules from the other class. The probability of each class is evenly divided between its members.

The knowledge that both the State Scheduler and the Transition Function Scheduler have concerning the transition function of the protocol is used so that the protocol progresses easier and faster. The strategy is to try to avoid spending steps in non effective transitions. However, in the case of the Modified scheduler, the strategy can be totally the opposite. In fact, this scheduler can lead the protocol to its worst case scenario by choosing a very small ε, and consequently by choosing with a high probability the identity transition rules.

In spite of this possible behavior, Chatzigiannakis et al. [25] proved that the Modified scheduler is fair with probability 1.

4.4 Equivalence between Schedulers

As shown in the previous section, there are different fair probabilistic schedulers that are designed for population protocols. The authors of [25] proposed to introduce two metrics to compare these fair schedulers with each other. These two metrics are: time equivalence and computational equivalence. They allow to check if two schedulers affect similarly the performance, and respectively the correctness, of a protocol.

4.4.1 Time Equivalence

Two probabilistic schedulers are time equivalent if they satisfy the following characterization [25]:

Definition 4.12. (**Time Equivalence**) Two fair probabilistic schedulers S_1 and S_2 are called time equivalent with respect to a protocol $\mathcal{A}$ iff all computations of $\mathcal{A}$ under S_1 and under S_2, beginning from the same initial configuration, take asymptotically the same expected time (number of steps) to converge.

Chatzigiannakis et al. proved, in [25], that the State Scheduler and the Transition Function Scheduler are time equivalent with respect to the One way Epidemic protocol (see Section 4.3.2). In fact, the computation of this protocol under the scheduling of the State Scheduler or the Transition Function Scheduler requires in both cases $O(n)$ interactions to reach the protocol's stabilization. However, with respect to the same protocol, the Random Scheduler is not time equivalent to them, as any computation under its scheduling requires $\Theta(n \log n)$ interactions [7].

4.4.2 Computational Equivalence

The second metric proposed by Chatzigiannakis et al. to establish a comparison between probabilistic schedulers is the computational equivalence described as follows [25]:

Definition 4.13. (**Computational Equivalence**) Two fair probabilistic schedulers S_1 and S_2 are called computationally equivalent with respect to a protocol $\mathcal{A}$ iff for all computations of $\mathcal{A}$ under S_1 and under S_2, beginning from the same initial configuration, $\mathcal{A}$ stabilizes to the same output assignment with high probability.

Lemma 1, from [25], states that the 3-States Approximate Majority (see Section 2.2) errs under the Transition Function Scheduler with a constant probability, when the initial number of agents with state x, N_x, is a $\Theta(N_y)$ (with N_y the initial number of agents with state y). This implies that a computation of this protocol under the Transition Function Scheduler, starting with a majority of x and with N_x is $\Theta(N_y)$, can stabilize to a configuration where the minority y wins. However, this can never happen if this protocol runs under the scheduling of the Random Scheduler. We can therefore establish that the Transition Function Scheduler is not computationally equivalent to the Random Scheduler w.r.t the 3-States Approximate Majority protocol.

We can consequently conclude that probabilistic schedulers, even if they satisfy the same fairness condition, are not necessarily equivalent.

4.5 The Random Scheduler Procedures

When authors introduced schedulers for the population protocol model, they just described how these schedulers proceed to choose an ordered pair of agents to interact.

In this section, we consider the possible implementations of such theoretical entities, and more specifically the implementation of the Random Scheduler. We present a procedure proposed by Becchetti et al. [18]. Then, we propose our solution that, unlike the previous one, preserves the anonymity of the agents [60]. It is also designed according to the mapping model between population protocols and anonymous asynchronous message passing (see Section 3.2.4), and stands for the procedure $TrySynchronization()$ in this context.

4.5.1 Random Scheduler Procedure: Becchetti et al.

Experiments of population protocols in a sensor network are presented in [18]. Becchetti et al. considered a population of agents, each equipped with a sensor. These sensors are able to communicate via a wireless media. The agents are placed in the same room, therefore the interaction graph is supposed to be complete. Also, each agent u has a unique identifier Id_u.

Becchetti et al. supposed that the population protocols running in this population are scheduled by the Random Scheduler. They therefore proposed an algorithm that simulates the processing of this scheduler, described by Algorithm 3.

This algorithm consists on two phases: a setup phase, and a neighborhood discovery and leader election phase. The basic idea is to elect a leader in this population, that is going to play the role of the initiator. Then, this leader chooses one of its neighbors to be a responder.

First, each agent u generates a random number Rd_u which is a combination of its identifier and its local time.

Algorithm 3 The Random Scheduler Procedure of Becchetti et al.

```
1: loop
2:    InitiatorFlag ← true
3:    NeighborsList ← ∅
4:    Randomly generate Rd;
5:    Broadcast m(Id, Rd);
6:    while ((ReceivedMessage ≠ ∅) and (InitiatorFlag = true)) do
7:       m(Id_v, Rd_v) ← the message received from a neighbor v;
8:       if (Id_v ∉ NeighborsList) then
9:          Add Id_v to NeighborsList;
10:      end if
11:      if (Rd_v > Rd) or ((Rd_v = Rd) and (Id_v > Id)) then
12:         InitiatorFlag ← false;
13:      end if
14:   end while
15:   if (InitiatorFlag = true) then
16:      Randomly select a responder v from NeighborsList;
17:   end if
18: end loop
```

Then, it starts discovering its neighborhood. Concurrently, a leader election process is initiated to nominate the initiator of the interaction in the network. Each agent initially has an initiator flag set to true and an empty list of neighbors. It starts by periodically broadcasting a message containing (Id_u, RD_u). Respectively, it receives messages from its neighborhood.

When an agent i receives a message from its neighbor j, the agent i checks its neighbors list. If Id_j does not appear there, i adds it. Then, i compares its Rd_i to Rd_j. If $Rd_i > Rd_j$, agent i maintains its initiator flag to true and considers that it still has the chance to be chosen as initiator. If $Rd_i < Rd_j$, i sets its initiator flag to false. Otherwise, if $Rd_i = Rd_j$, then the identifiers are compared, and if i has the highest Id it maintains its initiator flag to true.

As the interaction graph is complete, each agent is going to discover all the other agents of the population as neighbors, and eventually only one agent will maintain an initiator flag set to true. This initiator chooses randomly one neighbor (that will be the responder) to interact with.

When the initiator and the responder are both designated, the communication can take place. Once the interaction ends, the agents restarts all the procedure so that a new pair will be chosen to communicate.

This procedure permits in fact a central scheduling where only one ordered pair of agents is chosen and enabled to communicate at each step of the computation. Using the message passing as a communication model for such networks is a suitable choice. Yet, in this scheduling procedure, the agents are not anonymous, which does not respect the anonymity hypothesis of the population protocols.

4.5.2 Random Scheduler Procedure: Our Proposition

We propose an algorithm describing the Random Scheduler running in complete interaction graphs, and which is an enhancement of a result that we presented in [60].

We consider the mapping of populations protocols to distributed algorithms running over anonymous asynchronous systems based on message passing (see Section 3.2.4). Accordingly, the procedure we propose, described by Algorithm 4, is based on message passing. It preserves the anonymity of the agents as no identities are assigned to the agents, but we suppose that each agent is able to differentiate its different ports thanks to port numbering.

This procedure is executed by each agent in the population. Initially, each agent u starts by initiating its attribute $GlobalMin$ to true. Then, it generates, for each of its incident links, a random number $nbre$. The minimum number that was generated is attributed to $LocalMin$, and the edge label of the link over which this minimum was generated is associated to $LocalMinPort$.

Then, agent u broadcasts its $LocalMin$, and receives those of its neighbors. If the agent u concludes that it has generated the minimum number in the whole population, it becomes the initiator of the interaction that is going to take place. Consequently, it informs its neighbor, which is linked to it via $MinPort$, that it has to be the responder. Otherwise, if the agent is aware that it has not generated the minimum number, it checks if it is requested to be a responder. If it is not the case, it becomes aware that it does not belong to the chosen pair in this computation step.

This implementation guarantees that, at each step of the computation of the protocol, there is only one ordered pair of agents that are chosen to communicate. Hence, it satisfies the description of the Random Scheduler.

The procedure described by Algorithm 4 can also stand for the $TrySynchronization(u)$ procedure, with u the process representing the agent u in the population. It picks, at each computation step, only one ordered pair of agents to communicate. For each agent of this chosen pair, this procedure assigns non zero values to the parameters $role$ and $port$. For the remaining agents, these parameters maintain their initial values, that equal 0. Hence, by introducing this procedure to the local algorithm $Algo$ (see Algorithm 1), we obtain a description of the pairwise interaction process in a population protocol: the scheduler synchronizing a pair of agents, then the bidirectional and simultaneous communication. This latter was not explored in [18].

Remark 4.1. We would mention that the implementation described by Algorithm 4, as well as the one described by Algorithm 3, totally corresponds to the processing of the Random Scheduler. They are both randomized procedures whose results are the random choice of only one ordered pair of agents to communicate. Each ordered pair is chosen with the same probability, which is $\frac{1}{n(n-1)}$. Therefore, we do not have to investigate their fairness as the Random Scheduler was already proved to be a probabilistic fair scheduler with probability 1.

Algorithm 4 The Random Scheduler Procedure

```
1: loop
2:   GlobalMin ← true
3:   for i from 1 to d(u) do
4:     Generate a random number nbre;
5:     if (i = 1) then
6:       LocalMin ← nbre;
7:       LocalMinPort ← i;
8:     else
9:       if (nbre < LocalMin) then
10:        LocalMin ← nbre;
11:        LocalMinPort ← i;
12:      end if
13:    end if
14:  end for
15:  for i from 1 to d(u) do
16:    Send(LocalMin, i);
17:  end for
18:  for i from 1 to d(u) do
19:    Receive(NeighborLocalMin, i);
20:    if (LocalMin ≤ NeighborLocalMin) then
21:      GlobalMin ← false;
22:      break;
23:    end if
24:  end for
25:  if (GlobalMin = true) then
26:    role ← i;
27:    port ← LocalMinPort;
28:    Send(BeResponder, port);
29:  else
30:    for i from 1 to d(u) do
31:      Receive(m, i);
32:      if (m = BeResponder) then
33:        role ← r;
34:        port ← i;
35:        break;
36:      end if
37:    end for
38:  end if
39: end loop
```

4.6 The Handshake Scheduler

According to the definition of the $k-$centrality (Definition 4.2), all the schedulers presented above are $diam(G)-$central, also called sequential schedulers, where $diam(G)$ is the diameter of the interaction graph G over which a protocol runs. They allow only one enabled pair of agents to communicate at each step of the computation of a protocol.

In this section, we introduce the Handshake Scheduler (also denoted the *HS Scheduler*) which is a new probabilistic fair scheduler based on randomized handshakes. Unlike the existing schedulers, the Handshake Scheduler is $1-$central. It allows more than only one interaction to take place concurrently, as long as the picked pairs of nodes are disjoint. This scheduler can also be described as a distributed algorithm based on message passing that can be used in simulations and experiences of population protocols in real-world networks of sensors (or any tiny objects) without any need of identifiers. Compared to the implementations presented in the previous section of the Random Scheduler, the implementation of our proposed scheduler needs only one phase and does not require the completeness of the interaction graph.

4.6.1 Description of the Handshake Scheduler

Angluin et al. designed the population protocols as a computational model for networks of tiny artifacts where interactions take place between pairs of agents. By assuming that these protocols run under the Random Scheduler, they supposed that, at each step of a protocol's computation, only one ordered pair of agents is allowed to communicate. The same assumption was considered for the schedulers proposed by Chatzigiannakis et al.

Yet, these networks represent distributed systems where there is no centralized coordination. Consequently, more than one pairwise interaction can take place simultaneously. A distributed scheduler can capture these concurrent interactions as it is able to select, at each step of the computation, a set of enabled pairs of processes. However, to be quite suitable for the context of population protocols, this distributed scheduler has to satisfy the two following properties:

- **1-central scheduler** The scheduler should take into account that an agent in a population protocol can not participate in two communications at the same time. And, once it is involved in an interaction, it should finish communicating and updating its state according to the ongoing interaction before starting a new one. Otherwise, this can lead to erroneous states.

  To illustrate this, we take the example of the following scenario. Let a_1, a_2 and a_3 be three agents with respective states: q_1, q_2 and q_3. These agents belong to a given population $\mathcal{P}$. Let $\mathcal{A}$ be any population protocol running over the population $\mathcal{P}$. We suppose that among the transition rules of the transition function δ of the protocol $\mathcal{A}$, there are these two following rules:

  $$\delta\left(q_1, q_2\right) = \left(q_1', q_2'\right) \text{ and } \delta\left(q_3, q_1\right) = \left(q_3', q_1''\right).$$

  Suppose that a_1 initiates an interaction with a_2. They both exchange their corresponding states. Based on the state q_1 of the initiator, the receiver a_2 changes its state to q_2'. Likewise, a_1 has to change its state to q_1'. However, suppose that in

meanwhile, a_1 is involved as a receiver in an interaction with a_3 that should normally update its state to $q''1$. Now, which of the two states q'_1 and q''_1, the agent a_1 will adopt? Whatever a_1's chosen state is, one of the agents a_2 or a_3 would have changed its state based on an invalid state of a_1. Thus, concurrent interactions between joint pairs of agents cause inconsistent configurations of the whole network.

Therefore, if a distributed scheduler is designed for population protocols, it should choose, at each step of the computation, a set of disjoint ordered pairs of enabled agents to ensure that each agent is involved in only one interaction. According to the definition of the k-centrality, this scheduler has to be 1-central: among the enabled pairs of agents, those chosen by the scheduler should be separated by at least one edge.

- **Probabilistic scheduler** To better reflect the randomness (non determinism) of the order in which the agents interact during a protocol's computation, caused by the asynchrony of the population and the passive mobility of the agents, the distributed scheduler should be probabilistic. It attributes to each enabled pair a probability to be picked. If this scheduler is protocol oblivious, the probability distribution as well as the choice of the communicating edges, will not be based on any knowledge of the protocol run by the population. Otherwise, these probabilities will depend on the protocol.

Given these constraints, we propose a scheduler based on the randomized **handshake** model, also called the randomized **rendezvous** model. A handshake is an agreement between two nodes to communicate together in an exclusive mode [71]. Hence, the pairs of agents that will be chosen by the scheduler at each step will be those that are going to succeed on reaching an agreement about a handshake. Simultaneous random choices of different communicating pairs from the interaction graph are possible and the handshake guarantees that these pairs are disjoint. The probability for each pair of agents (or an edge) to be chosen by the scheduler is the probability that these two agents agree about a handshake.

We call this scheduler the Handshake Scheduler and denote it the *HS Scheduler*. We can represent this scheduler as a labeling HS over the edges of the interaction graph. A more formal definition can be as follows:

Definition 4.14. (**The Handshake Scheduler**) For any population $\mathcal{P}$ with an interaction graph $G = (V, E)$, for any population protocol $\mathcal{A}$ running in $\mathcal{P}$, a scheduling of the protocol $\mathcal{A}$ under the Handshake Scheduler corresponds to the following mapping:

$$\begin{array}{rccc} HS: & E & \rightarrow & [0,1] \\ & e & \mapsto & \mathbb{P}r\left(\mathcal{HS}(e)\right) \end{array}$$

with $\mathbb{P}r\left(\mathcal{HS}(e)\right)$ the probability that a handshake takes place over the edge e.

Now, if we consider the computation of a population protocol under the *HS Scheduler*, the main basic concepts that were already presented in Chapter 2, remain the same excepting the definition of a transition.

In fact, this definition should now take into account the possible choice of, not only one, but multiple edges to go from one configuration to another. With respect to this description, we give the new definition of a transition.

Definition 4.15. (**Transition**) Let C and C' be two population configurations. Let $Enc = \{e_1, e_2, \ldots, e_k\}$ be a set of disjoint encounters from E such that:

- $\forall\, i \in \{1, 2, \ldots, k\}$, $e_i \in E$,
- $\forall\, l,\, l' \in \{1, 2, \ldots, k\}$, such that $l \neq l'$, if $e_l = (u, v)$ and $e_{l'} = (u', v')$ then: $u \neq u'$, $u \neq v'$, $v \neq u'$ and $v \neq v'$.

We say that C goes to C' via the set of encounters Enc, denoted $C \xrightarrow{Enc} C'$, if:

- $\forall\, e_i = (u_i, v_i) \in Enc$, $C'(u_i) = \delta_1(C(u_i), C(v_i))$ and $C'(v_i) = \delta_2\,(C(u_i), C(v_i))$,
- $C'(w) = C(w)$ for any node w which is not an extremity of an edge from Enc.

We say that C can go in one step to C', denoted $C \to C'$, if $C \xrightarrow{Enc} C'$ for some set of encounters $Enc \subset E$ (Enc should fulfill the conditions we specified above, that is it should be a set of disjoint edges from E). $C \to C'$ is called a transition.

A configuration can also be reachable from another one with more than one computation step, and thus via a sequence of configurations.

Definition 4.16. (**Reachable Configuration**) Let C and C' be two population configurations. $C \xrightarrow{*} C'$ denotes that C' is reachable from C via a sequence of configurations $C_0, C_1, \ldots, C_k$ where $C = C_0$, $C_k = C'$ and $\forall\, i \in \{0, 1, ..., k-1\}$, $C_i \to C_{i+1}$ (with respect to Definition 4.15).

As we already mentioned above, the proposed scheduler is based on the handshake model. In what follows, we give more details about this model.

4.6.2 The Handshake Model

In some communication models, both the sender and the receiver need to be ready to exchange information. They need to have an agreement to communicate: it is called the handshake model. This model is useful in radio networks, where if a node is called simultaneously by more than one neighbor, the received messages can be in collision. The rendezvous model is also useful in case a physical meeting is needed between the entities to communicate such as in robots networks. Moreover, in anonymous asynchronous networks, based on point-to-point communication via synchronous message passing, such an agreement is needed so that the sender and the receiver can exchange messages [32].

Randomized Handshake Algorithms

Different works proposed handshakes algorithms that depend on the assumptions made on the system. We focus on the case of anonymous asynchronous networks that are based on point-to-point communication via synchronous message passing.

Angluin stated in [2] that there is no deterministic algorithm to implement synchronous message passing in an anonymous network that passes messages asynchronously. Therefore, if such algorithm exists, it should be a randomized algorithm.

A Synchronous Randomized Handshake Algorithm Métivier et al. proposed a randomized distributed handshake algorithm that uses a simple bit to encode messages [52, 53]. In this algorithm, each node v in the network runs Algorithm 5. We would mention that we present this algorithm as it was introduced by the authors. Consequently we should outline that choosing a neighbor by a process v, in this context of anonymous processes, corresponds more precisely to choosing a port, and the neighbor $c(v)$ is the neighbor with which v can communicate via the chosen port.

Algorithm 5 The Randomized Handshake Algorithm

```
1: loop
2:    Choose independently and uniformly at random a neighbor c(v);
3:    Send 1 to c(v);
4:    Send 0 to each neighbor u ≠ c(v);
5:    Receive messages from all the neighbors;
6:    r_{c(v)} ← the number received from c(v);
7:    if (r_{c(v)} = 0) then
8:       There is no rendezvous;
9:    else
10:      There is a rendezvous with c(v);
11:   end if
12: end loop
```

Each node v chooses independently and uniformly at random a node $c(v)$ (or more exactly a port number) from its neighborhood. The node v sends 1 to $c(v)$, and 0 to the rest of its neighbors. This implies that v would like to synchronize with node $c(v)$. There is a rendezvous between v and $c(v)$ if they mutually choose each other by sending 1 to each other. Hence, a computation step related to the asynchronous algorithm that uses this rendezvous algorithm for synchronization can take place.

Locally, when a node v has more than one neighbor, Algorithm 5 defines how v chooses only one of them to synchronize with. This guarantees that each node in the network will be able to participate to at most one rendezvous at a given time.

This handshake algorithm is hence a local mechanism that automatically and randomly performs a temporary and exclusive synchronization between two neighbors in an asynchronous environment and whose global result of is a set of disjoint pairs of synchronized nodes (matchings).

We mentioned in Chapter 3 (see Section 3.1) that, in graph relabeling systems, different relabeling rules can be applied simultaneously as long as the set of interacting nodes are disjoint. Accordingly, to encode the synchronization needed for $LC0$ interactions in asynchronous message passing systems, Bauderon et al. [14] opted for this handshake algorithm.

An Asynchronous Randomized Handshake Algorithm The randomized rendezvous algorithm proposed by Métivier et al. is a synchronous algorithm that uses an important number of messages. Indeed, each process sends a message via each of its ports.

Fontaine et al. proposed, in [37], an asynchronous version of this algorithm that also reduces the number of messages sent over the network. In this algorithm, each process in the network proceeds as described in Algorithm 6.

Algorithm 6 The Asynchronous Randomized Handshake Algorithm

```
1: loop
2:    Choose independently and uniformly at random a neighbor c(v);
3:    Send 1 to c(v);
4:    Receive a message m from a neighbor w;
5:    if ((w = c(v)) and (m = 1)) then
6:       There is a rendezvous between v and c(v);
7:       Return to step 2;
8:    else
9:       if ((w = c(v)) and (m = 0)) then
10:         Return to step 2; // There is no rendezvous with c(v)
11:      else
12:         Send 0 to w;
13:         Return to step 4;
14:      end if
15:   end if
16: end loop
```

Each process v chooses uniformly at random a neighbor $c(v)$, and sends it 1. Then, it remains henceforth waiting for a message from this neighbor. Consequently, there are three possible scenarios:

1. If the process v receives a message 1 from its chosen neighbor $c(v)$, then there is a rendezvous. The processes v and $c(v)$ are hence synchronized and a computation step (related to the asynchronous algorithm using the rendezvous algorithm for pairwise synchronization) can take place. Then, process v executes once again the algorithm.

2. If v receives a message from $c(v)$, but whose value is 0, it concludes that $c(v)$ has chosen another neighbor to synchronize with. There is hence no rendezvous. Consequently, process v moves to step 2 to try a new synchronization.

3. If process v still did not received a message from $c(v)$, and in meanwhile, it receives a message 1 from a neighbor different from $c(v)$, it sends 0 to it. The process v returns afterwards to step 4 to continue waiting for a response from $c(v)$.

Fontaine et al. stated that this randomized asynchronous handshake algorithm is also fault tolerant, that is even if there are some faulty nodes, the algorithm is able to continue running correctly [37].

4.6.3 The Randomized Algorithm of the Handshake Scheduler

We provided in a previous section a description of the Handshake Scheduler we introduced, that shows its specificities compared to the already existing schedulers. We aim now to propose a suitable algorithm to describe the processing of this scheduler.

This Handshake Scheduler is based on randomized handshakes. Thus, we think that its corresponding algorithm can be inspired from the randomized rendezvous algorithms we presented above. As the synchronous handshake algorithm (Algorithm 5) was used for synchronization in $LC0$ computations, and as we already established some correspondence between the pairwise interactions in populations protocols and $LC0$ interactions, we opt for adopting this algorithm as a Handshake Scheduler algorithm. This proposition is applied in the context of a scheduling of the OR population protocol whose transition function is symmetric.

We then deduce that, both the synchronous and asynchronous versions of the handshake algorithm can be used to describe the Handshake Scheduler provided that symmetry breaking is added to them.

The Randomized Handshake Algorithm as a Scheduler for Population Protocols

We studied, in [60], the broadcast in anonymous asynchronous mobile wireless sensor networks. Starting from only one informed node, the information should reach every node in the network at the end. This broadcasting process can be simulated by the OR predicate which corresponds to a logical OR over the inputs of the sensors. This predicate is stably computable by the population protocols, and more specifically by the following one that we call the OR protocol.

The OR protocol can be described by the 6-tuple $(X_{OR}, Y_{OR}, Q_{OR}, I_{OR}, O_{OR}, \delta_{OR})$ with:

- $X_{OR} = Y_{OR} = \{1, 0\}$ with 1: the input (respectively output) of an informed agent and 0: the input (respectively output) of a non informed agent,
- $Q_{OR} = \{q_0, q_1\}$,
- $I_{OR}(0) = q_0$, $I_{OR}(1) = q_1$,
- $O_{OR}(q_0) = 0$, $O_{OR}(q_1) = 1$, and
- δ_{OR}:

$$\delta_{OR}(*, q_1) = \delta_{OR}(q_1, *) = (q_1, q_1),\ \forall * \in Q_{OR}.$$

Thus, we proposed a randomized distributed algorithm based on the OR population protocol to avoid collision and information duplication problems. We represented the sensor network as an anonymous asynchronous system based on message passing, and with port numbering. Consequently, we needed to proceed according to the mapping of the computation of a population protocol to a distributed anonymous asynchronous algorithm with message passing, that we defined in Chapter 3 (see Section 3.2.4).

We proposed to use Algorithm 5 to stand for the *TrySynchronization* procedure and to obtain pairwise synchronizations. The synchronized pairs of agents are those that succeeded in obtaining a rendezvous.

Once two agents are synchronized, they communicate their respective states to each other. This bidirectional communication is as described in the *Algo* algorithm (Algorithm 1): each agent v of the synchronized pair sends its state via the port $c(v)$, and

receives the state of the synchronized neighbor also via this port. If one of them is informed, and thus in state q_1, and the second is not, the informed node forwards the information to the non informed one. The non informed node then becomes informed and updates its state from q_0 to q_1. Otherwise, the information is not sent.

Using the randomized rendezvous algorithm ensures that, once a node is involved in a communication, it will communicate in an exclusive mode. Thus, if it receives the information, it will be from only one neighbor at a given time, which avoids collision problems. Also, communicating the respective states between two synchronized nodes, before the informed node sends the information, avoids the nodes receiving more than one copy of the information. Furthermore, as the OR predicate is stably computable by population protocols, it guarantees that the broadcast algorithm eventually stabilizes to a configuration where all the nodes receive the information.

Although the randomized rendezvous provides only the port number via which an agent v is synchronized, and not the role of v, it can stand for the $TrySynchronization$ procedure needed by the $Algo$ algorithm describing the OR protocol. In fact, the transition function of this protocol is symmetric. It consists of only one effective transition rule which is symmetric:

$$\delta(q_1, q_0) = \delta(q_0, q_1) = (q_1, q_1) \, .$$

Hence, the application of a transition rule in not conditioned by the role of the agent in this case as the transition function affects similarly the initiator and a responder in an interaction.

Accordingly, the randomized rendezvous can be considered, in this case, as an implementation of a distributed scheduler based on handshakes.

The Handshake Scheduler Algorithm: an Enhancement of the Randomized Handshake Algorithm

The synchronous randomized handshake algorithm, as it was used in the previous section, allows the synchronization of multiple disjoint pairs of agents in some iterations of the broadcast algorithm. This randomized handshake algorithm decides which pairs of agents will interact and in what order. However, these synchronized pairs are not ordered, which means that the initiator and the responder are not distinguished. By analogy, we can also say that using the asynchronous version of this algorithm will lead to the same result.

Not breaking the symmetry between the two synchronized agents has no consequence in the context of the protocol OR protocol or any population protocol whose transition function is symmetric. However, not all the population protocols have symmetric transition rules. Also, breaking the symmetry is a fundamental assumption in the population protocol model. Therefore, to be a suitable distributed scheduler for population protocols, each of these two randomized rendezvous should be adjusted to consider symmetry breaking.

In order to fit this requirement, we propose that instead of sending 1 when inviting the chosen neighbor $c(v)$, each agent v generates independently and uniformly at random a value $r_v \in \{1, 2, \ldots, N\}$ where N is a constant such that $N \geq 2$, and sends it to $c(v)$. A handshake takes place, if two nodes mutually choose each other and generate two different

non zero values. The role of the initiator is be attributed to the node that generated the higher value.

We also recall that population protocols were designed for asynchronous networks of agents that have constraints on memory, computational power and energy. Thus, we opt for an adjustment of the asynchronous randomized handshake algorithm (Algorithm 6) to describe the distributed handshake scheduler. In fact, on the one hand, this version of the handshake algorithm takes into account the asynchrony of the communications in the population. And, on the other hand, as it uses less messages (the 0 messages) compared to the synchronous version, it respects the constraint related to the energy of the agents.

The result of these modifications is the randomized algorithm described in Algorithm 7.

Algorithm 7 The Randomized Handshake Scheduler Algorithm

1: **loop**
2: Choose independently and uniformly at random a port $port_{c(v)}$;
3: Choose independently and uniformly at random a number $r_v \in \{1, 2, \ldots, N\}$;
4: Send $(r_v, port_{c(v)})$;
5: Receive $(r_w, port_w)$;
6: **if** $\big((port_w = port_{c(v)})$ and $(r_v \neq r_w)\big)$ **then**
7: **if** $(r_v > r_w)$ **then**
8: $role \leftarrow i$;
9: **else**
10: $role \leftarrow r$;
11: **end if**
12: Return to step 2;
13: **else**
14: **if** $\big((port_w = port_{c(v)})$ and $((r_w = 0)$ or $(r_v = r_w))\big)\big)$ **then**
15: Return to step 2; // There is no rendezvous with $c(v)$
16: **else**
17: Send $(0, port_w)$;
18: Return to step 5;
19: **end if**
20: **end if**
21: **end loop**

The randomized distributed algorithm, consisting of a collection of Algorithm 7 run by the agents of the population, plays the role of the Handshake Scheduler for a population protocol. Also, Algorithm 7 stands for the *Trysynchronization* procedure in the *Algo* algorithm according to the mapping of a population protocol to a distributed algorithm based on message passing in an anonymous asynchronous system.

Once a synchronization (or a handshake) is established between two nodes, they exchange their states. If there is a transition rule involving these states, they will apply it with respect to their roles.

4.6.4 Analysis of the Randomized Handshake Scheduler Algorithm

For the analysis of the distributed algorithm corresponding to the Handshake Scheduler, we consider a synchronous execution of Algorithm 7 (see [44]).

Probability of at least a Handshake

Let $\mathcal{A}$ be a population protocol running over an interaction graph $G = (V, E)$. We suppose that the pairwise interactions are handled by the Handshake Scheduler. We thus recall that, at each computation, this scheduler picks for interaction the ordered pairs of agents that succeeded in a handshake.

For any pair $e = (u, v)$, such that the edge e exists in the interaction graph G, there is a handshake between agents u and v, denoted $\mathcal{HS}(e)$ if, and only if, $c(u) = v$, $c(v) = u$ and $r_u \neq r_v$.

Lemma 4.2. *Let $\mathcal{A}$ be a population protocol running under the HS Scheduler in a population whose interaction graph is $G = (V, E)$, then:*

$$\forall e = (u, v) \in E, \mathbb{P}r\left(\mathcal{HS}(e)\right) \;=\; \frac{\beta}{d(u)d(v)}\,, \text{ with } \beta = 1 - \frac{1}{N}\,.$$

Proof. We first note that the probability that $c(u) = v$ is:

$$\mathbb{P}r(c(u) = v) = \frac{1}{d(u)}\,, \text{where } d(u) \text{ is the degree of the node } u. \tag{4.1}$$

We also note that the probability that u and v generate two different non zero values is:

$$\mathbb{P}r(r_u \neq r_v) = \sum_{i=1}^{N} \frac{1}{N}\frac{N-1}{N} = 1 - \frac{1}{N} = \beta\,. \tag{4.2}$$

Then, we obtain:

$$\begin{aligned} \mathbb{P}r\left(\mathcal{HS}(e)\right) &= \mathbb{P}r(c(u) = v) \times \mathbb{P}r(c(v) = u) \times \mathbb{P}r(r_u \neq r_v) \\ &= \frac{\beta}{d(u)d(v)}\,. \end{aligned} \tag{4.3}$$

□

In the sequel, we denote $\{e_1, \ldots, e_m\}$ the set of edges in the interaction graph. We also denote by $\mathcal{HS}_G$ the event: *There is at least a handshake in the graph G*. The $\overline{\mathcal{HS}}_G$ and $\overline{\mathcal{HS}}(e)$ are respectively the complement events of $\mathcal{HS}_G$ and $\mathcal{HS}(e)$. Then, as described in [53], we have:

$$\mathbb{P}r\left(\overline{\mathcal{HS}}_G\right) \;=\; \mathbb{P}r\left(\wedge_{i=1}^{m}\overline{\mathcal{HS}}(e_i)\right)$$

Applying Proposition 1.6 gives:

$$\begin{aligned} \mathbb{P}r\left(\overline{\mathcal{HS}}_G\right) &= \prod_{i=1}^{m} \mathbb{P}r\left(\overline{\mathcal{HS}}(e_i) \mid \wedge_{j=1}^{i-1}\overline{\mathcal{HS}}(e_j)\right) \\ &= \prod_{i=1}^{m} \left(1 - \mathbb{P}r\left(\mathcal{HS}(e_i) \mid \wedge_{j=1}^{i-1}\overline{\mathcal{HS}}(e_j)\right)\right)\,. \end{aligned} \tag{4.4}$$

Let g be any subset of E. Let ξ_g (respectively $\overline{\xi_g}$) denote the event of obtaining a handshake for at least one edge (respectively no edge) in g. Then, according to a Lemma from [32] (we will give more details about this Lemma in the following chapter), we can state that:

$$\forall\, e \in E,\ \mathbb{P}r\left(\mathcal{HS}(e)|\overline{\xi_g}\right) \geq \mathbb{P}r(\mathcal{HS}(e)).$$

This yields:

$$\forall\, 1 \leq i \leq m,\ \mathbb{P}r\left(\mathcal{HS}(e_i) \mid \wedge_{j=1}^{i-1}\overline{\mathcal{HS}}(e_j)\right) \geq \mathbb{P}r\left(\mathcal{HS}(e_i)\right)\ .$$

Consequently, (4.4) becomes:

$$\mathbb{P}r\left(\overline{\mathcal{HS}}_G\right) \ \leq\ \prod_{i=1}^{m}\left(1-\mathbb{P}r\left(\mathcal{HS}(e_i)\right)\right)\ . \tag{4.5}$$

Based on Proposition 1.15, we can write:

$$\forall\,(x_i)_{1\leq i\leq m} \text{ such that: } \forall\, i, 0 \leq x_i \leq 1,\ \text{we have } \prod_{i=1}^{m}(1-x_i) \leq \left(1-\frac{\sum_{i=1}^{m} x_i}{m}\right)^m .$$

Then, the inequality (4.5) becomes:

$$\mathbb{P}r\left(\overline{\mathcal{HS}}_G\right) \ \leq\ \left(1-\frac{\sum_{i=1}^{m}\mathbb{P}r\left(\mathcal{HS}(e_i)\right)}{m}\right)^m . \tag{4.6}$$

On the other hand, we have:

$$\begin{aligned}
\sum_{i=1}^{m}\mathbb{P}r\left(\mathcal{HS}(e_i)\right) &= \sum_{\{u,v\}\in E}\frac{\beta}{d(u)d(v)}\\
&\geq m\left(\prod_{\{u,v\}\in E}\frac{\beta}{d(u)d(v)}\right)^{\frac{1}{m}}\\
&\geq m\left(\frac{\beta}{\frac{1}{m}\sum_{\{u,v\}\in E}d(u)d(v)}\right)\\
&\geq m\left(\frac{\beta}{\frac{1}{m}\sum_{\{u,v\}\in E}(n-1)^2}\right)\\
&\geq \frac{\beta\, n\,(n-1)}{2(n-1)^2}\\
&\geq \frac{\beta}{2}\,.
\end{aligned} \tag{4.7}$$

Thus, (4.6) becomes:

$$\mathbb{P}r\left(\overline{\mathcal{HS}}_G\right) \ \leq\ \left(1-\frac{\beta}{2m}\right)^m \sim e^{-\frac{\beta}{2}}\,. \tag{4.8}$$

This yields:

$$\mathbb{P}r\left(\mathcal{HS}_G\right) \geq 1-e^{\frac{-\beta}{2}}\ . \tag{4.9}$$

Consequently, given the inequality 4.9, we can state the following Lemma:

Lemma 4.3. *Let $\mathcal{A}$ be a population protocol running on the communication graph $G = (V, E)$ under the HS Scheduler. Then, the probability that the HS Scheduler picks at least one ordered pair of nodes at the end of Algorithm 7 is lower bounded by $1 - e^{\frac{-\beta}{2}}$.*

The Number of Simultaneous Handshakes

Let $G = (V, E)$ be the communication graph of a population protocol $\mathcal{A}$. Let X be the random variable $(r.v)$ which counts the number of simultaneous handshakes that can take place at the same step, and that corresponds to the number of concurrent pairwise interactions that can take place. Using the linearity of the expectation, we can obtain the expected number of X:

$$\mathbb{E}(X) = \sum_{(u,v)\in E} \left(\frac{\beta}{d(u)d(v)} \right) . \tag{4.10}$$

Analysis of the Handshake Scheduler Algorithm in particular cases of graph structures

We consider the following particular cases of graph structures that can represent the interaction graph of a population and compute the corresponding probabilities of a handshake over an edge as well as the number of possible simultaneous handshakes.

Case of a Complete Graph Let the interaction graph $G = (V, E)$ be a complete graph of size $n \geq 2$. Then, for any $e = (u, v) \in E$, the probability of a handshake over this edge is

$$\mathbb{P}r(\mathcal{HS}(e)) = \frac{\beta}{(n-1)^2} . \tag{4.11}$$

Accordingly, the expected number of simultaneous handshakes becomes:

$$\begin{aligned} \mathbb{E}(X) &= \sum_{(u,v)\in E} \frac{\beta}{(n-1)^2} \\ &= \frac{n(n-1)}{2} \frac{\beta}{(n-1)^2} \\ &= \frac{\beta\, n}{2\,(n-1)} . \end{aligned} \tag{4.12}$$

Case of a Random Graph $\mathbf{G_{n,p}}$ In this section, we consider that the communication graph $G = (V, E)$ is a random graph $G_{n,p}$ with $p > 0$. It is straightforward that we have some edges.

The probability of a handshake over an edge $e = (u, v)$, with $e \in E$ is as described by the following Lemma.

Lemma 4.4. *Let $G_{n,p} = (V, E)$ be a random graph, and let $e = (u, v) \in E$. Then,*

$$\mathbb{P}r(\mathcal{HS}(e)|e \in E) = \beta \frac{(1 - q^{n-1})^2}{(n-1)^2 p^2} . \tag{4.13}$$

Proof. We assume that $G_{n,p} = (V, E)$ is a random graph and $e = (u, v) \in E$. We study the probability of having a handshake on e conditioned on the event $e \in E$.

$$\begin{aligned}\mathbb{P}r(\mathcal{HS}(e)|\, e \in E) &= \sum_{k_1=1}^{n-1}\sum_{k_2=1}^{n-1} \mathbb{P}r(\mathcal{HS}(e)|\, e \in E \wedge d(u) = k_1 \wedge d(v) = k_2) \\ &\times \mathbb{P}r(d(u) = k_1 \wedge d(v) = k_2|\, e \in E).\end{aligned}$$

We already have,

$$\mathbb{P}r(\mathcal{HS}(e)|\, e \in E \wedge d(u) = k_1 \wedge d(v) = k_2) = \frac{\beta}{k_1\, k_2}.$$

Also,

$$\mathbb{P}r(d(u) = k_1 \wedge d(v) = k_2|\, e \in E) = \mathbb{P}r(d(u) = k_1|\, e \in E) \times \mathbb{P}r(d(v) = k_2|\, e \in E) \quad (4.14)$$

On the other hand, the degree distribution of any vertex x in a random graph $G_{n,p}$ is binomial such as:

$$\mathbb{P}r(d(x) = w) = \binom{n-1}{w} p^w\, (1-p)^{n-1-w}.$$

Thus,

$$\mathbb{P}r(d(u) = k|\, e \in E) = \binom{n-2}{k-1} p^{k-1}\, (1-p)^{n-1-k}.$$

Then, from 4.14, we obtain:

$$\mathbb{P}r(d(u) = k_1 \wedge d(v) = k_2|\, e \in E) = \binom{n-2}{k_1-1}\binom{n-2}{k_2-1} p^{k_1+k_2-2}\, (1-p)^{2n-2-k_1-k_2}.$$

This yields

$$\mathbb{P}r(\mathcal{HS}(e)|\, e \in E) = \beta\left(\sum_{k=1}^{n-1}\frac{1}{k}\binom{n-2}{k-1} p^{k-1}\, (1-p)^{n-1-k}\right)^2. \quad (4.15)$$

Now, we focus on:

$$\begin{aligned}\sum_{k=1}^{n-1}\frac{1}{k}\binom{n-2}{k-1} p^{k-1}\, (1-p)^{n-1-k} &= \sum_{k=1}^{n-1}\frac{1}{k}\frac{(n-2)!}{(k-1)!(n-1-k)!} p^{k-1}\, (1-p)^{n-1-k} \\ &= \frac{1}{n-1}\sum_{k=1}^{n-1}\frac{(n-2)!}{k!(n-1-k)!} p^{k-1}\, (1-p)^{n-1-k} \\ &= \frac{1}{p\,(n-1)}\left(\sum_{k=0}^{n-1}\binom{n-1}{k} p^k\, (1-p)^{n-1-k} - q^{n-1}\right) \\ &= \frac{(1-q^{n-1})}{p\,(n-1)}. \quad (4.16)\end{aligned}$$

Consequently, integrating (4.16) in (4.15) gives:

$$\mathbb{P}r(\mathcal{HS}(e)|\, e \in E) = \beta\,\frac{(1-q^{n-1})^2}{p^2\,(n-1)^2}.$$

Note that this proof is an adaptation of the analysis already presented in [71] to our handshake procedure, however we gave here further details of the computation.

□

Corollary 4.5. *Let $G_{n,p} = (V, E)$ be a random graph, and let u and v be two vertices from V. Then,*

$$\mathbb{P}r(\mathcal{HS}(e)) = \beta \frac{(1-q^{n-1})^2}{(n-1)^2 p}\,. \tag{4.17}$$

Proof. Let $G_{n,p} = (V, E)$ be a random graph. Let u and v be two vertices from V such that $u \neq v$. The probability of a handshake between u and v can be expressed this way:

$$\begin{aligned} \mathbb{P}r(\mathcal{HS}(e)) &= \mathbb{P}r(\mathcal{HS}(e)|e \in E) \times \mathbb{P}r(e \in E) \\ &+ \mathbb{P}r(\mathcal{HS}(e)|e \notin E) \times \mathbb{P}r(e \notin E). \end{aligned} \tag{4.18}$$

There is no handshake between u and v if they are not linked by an edge from E. This yields:

$$\mathbb{P}r(\mathcal{HS}(e)|e \notin E) = 0\,.$$

Then, (4.18) becomes:

$$\begin{aligned} \mathbb{P}r(\mathcal{HS}(e)) &= \mathbb{P}r(\mathcal{HS}(e)|e \in E) \times \mathbb{P}r(e \in E) \\ &= \beta \frac{(1-q^{n-1})^2}{p^2\,(n-1)^2} \times p \\ &= \beta \frac{(1-q^{n-1})^2}{p\,(n-1)^2}\,. \end{aligned} \tag{4.19}$$

□

As a consequence of Corollary (4.5), if $X_{n,p}$ is the number of simultaneous handshakes in a $G_{n,p}$ graph and $\mathbb{E}\,(X_{n,p})$ is its expected value, then:

$$\mathbb{E}\,(X_{n,p}) = \frac{\beta\, n}{2\,(n-1)\,p} \left(1-q^{n-1}\right)^2\,. \tag{4.20}$$

We note that the expression (4.20) can be simplified for some particular values of p:

- If p is a constant, then $(1-q^{n-1})^2 \to 1$ as $n \to \infty$. Hence,

$$\mathbb{E}\,(X_{n,p}) \sim \frac{\beta}{2p} \text{ as } n \to \infty.$$

- If $p = \frac{\alpha \log n}{n}$, with $\alpha > 1$ then:

$$\mathbb{E}(X_{n,p}) \sim \beta \frac{n}{2\alpha \log n}, \text{ as } n \to \infty. \tag{4.21}$$

The most interesting case is the last one where $p = \frac{\alpha \log n}{n}$ for $\alpha > 1$. Indeed, this value is the connectivity threshold for a $G_{n,p}$ [33, 34, 13].

4.6.5 The Fairness of the Handshake Scheduler

We introduced the *HS Scheduler* and proposed a randomized procedure to implement it. Now, we aim to prove that this scheduler satisfy a basic property which is fairness.

Based on the analysis results presented in the previous section, we first prove that the *HS Scheduler* is a probabilistic consistent scheduler. Then, we conclude that it is fair with probability 1.

The Handshake Scheduler: a Probabilistic Consistent Scheduler

We recall that a transition graph $T = (V(T), E(T))$ of a protocol $\mathcal{A}$ running on G, also denoted $T(\mathcal{A}, G)$, is the directed graph whose nodes are all possible population configurations and edges are all possible one-step transitions, and that may contain self-loops.

Theorem 4.6. *The HS Scheduler is a probabilistic consistent scheduler.*

Proof. Let $\mathcal{A}$ be any population protocol running over an interaction graph $G = (V, E)$ under the *HS Scheduler*. Let $T(\mathcal{A}, G) = (V(T), E(T))$ be its transition graph, and C_i be any configuration in $V(T)$. Let C_j be any configuration in $V(T)$ reachable in one step from C_i.

We define $\mathcal{E}nc_{C_iC_j} = \{Enc \mid Enc \subset E(G) \text{ and } C_i \xrightarrow{Enc} C_j\}$. Then, for each element Enc from $\mathcal{E}nc_{C_iC_j}$, we define the set E of the edges of the interaction graph G, as the union of three disjoint subsets: Enc, F_1 and F_2. F_1 represents the set of edges that are joint to Enc. F_2 represents the set of edges that are disjoint to Enc.

More formally, $\forall\, Enc \subset \mathcal{E}nc_{C_iC_j}$, we rewrite the set E as following:

$$E = Enc \uplus F_1 \uplus F_2 \,.$$

with:

- $F_1 = \{f \in E \mid \text{if } f = (u, v) \text{ then } \exists\, e \in Enc \text{ such that } e = (u, v') \text{ or } e = (u', v)\}$,
- $F_2 = \{f \in E \mid \text{if } f = (u, v) \text{ then } \forall\, e \in Enc, \text{ if } e = (u', v') \text{ then } u \neq u',\ u \neq v',\ v \neq u' \text{ and } v \neq v'\}$.

Any time C_i is encountered, C_j is selected with the following probability

$$\mathbb{P}r_{C_iC_j} = \sum_{Enc \subset \mathcal{E}nc_{C_iC_j}} \mathbb{P}r\left(\mathcal{HS}(Enc)\right) \mathbb{P}r\left(\overline{\mathcal{HS}}(F_2)\right) \,.$$

Consequently, with respect to the transition graph T and according to Definition 4.9, we can state that the *HS Scheduler* is probabilistic.

The probability of handshakes on any set of edges depends on the probability of a handshake on each of its edges. We already proved in the previous section (see Lemma 4.2) that for any edge $e = (u, v) \in E$, we have:

$$\mathbb{P}r\left(\mathcal{HS}(e)\right) = \frac{\beta}{d(u)d(v)}.$$

This probability does not depend on time. Therefore, the value of the probability $\mathbb{P}r_{C_iC_j}$ will be independent of the number of times C_i has been encountered. This leads us to conclude that the *HS Scheduler* is also consistent.

□

The Handshake Scheduler: a Fair Scheduler with Probability 1

Theorem 4.7. *The HS Scheduler is fair with probability 1.*

Proof. Let $\mathcal{A}$ be any population protocol whose interaction graph is $G = (V, E)$ and that runs under the scheduling of the *HS Scheduler*. Let $T(\mathcal{A}, G)$ be its transition graph and C_i, C_j be any configurations in $V(T)$ such that $C_i \rightarrow C_j$ and $C_i \neq C_j$.

In Theorem 4.6, we already proved that the *HS Scheduler* is a probabilistic consistent scheduler. Now, to prove that this scheduler is fair with probability 1, according to Theorem 4.1, we still have to prove that $\mathbb{P}r_{C_iC_j} > 0$.

Based on Definition 4.15 of a transition in a computation of a protocol under the *HS Scheduler*, we have $C_i \rightarrow C_j$ implies that: $\exists\ Enc \subset E$ such that $C_i \xrightarrow{Enc} C_j$. This means that $\mathbb{P}r_{C_iC_j} > 0$.

Hence, by applying Theorem 4.1, we conclude that the *HS Scheduler* is fair with probability 1.

□

4.7 Conclusion

We were interested, in this chapter, in scheduling in population protocols. We gave an overview of the existing schedulers in literature, and that are all central. We focused on the Random scheduler and proposed its implementation as a distributed algorithm based on message passing. We then introduced the *HS Scheduler*, a new probabilistic scheduler for population protocols which is, unlike the existing ones, distributed. We also proposed an algorithm describing this scheduler and which based on the randomized handshake algorithms this scheduler. We presented an analysis of this Handshake Scheduler algorithm that enabled us to prove that this scheduler is a probabilistic consistent fair scheduler with probability 1.

The Random scheduler and the *HS Scheduler* are both protocol oblivious schedulers that are fair with probability 1. We would investigate now if they are also time equivalent with respect to some protocols.

Chapter 5

Time Complexity Analyses of some Protocols under the Random Scheduler and the HS Scheduler

We introduced in the previous chapter the *HS Scheduler* as a new probabilistic scheduler for populations protocols. This scheduler is protocol oblivious and fair with probability 1 like the Random Scheduler introduced by Angluin et al. However, satisfying the fairness property does not imply the equivalence of the schedulers [25].

We propose to investigate, in this chapter, the time equivalence of these two schedulers with respect to three protocols: the OR population protocol (see Section 4.6.3), the Leader Election population protocol (see Section 2.2.3) and the Maximal Matching mediated population protocol. We therefore study the complexity of the stabilization time of these protocols by considering two scheduling scenario: either under the Random Scheduler or under the *HS Scheduler*. We also consider different structures of interaction graphs.

We prove that the two schedulers are time equivalent with respect to these three protocols when running in complete interaction graphs. Yet, the *HS Scheduler* performs better in case of random interaction graphs.

It is important to note that the stabilization time corresponds to the number of steps or rounds needed for the protocol to stabilize. In the case of protocols running under the Random Scheduler, the time complexity can also be considered as being the number of interactions needed until the protocol stabilizes as there is only one interaction at each step of the computation. However, it is not the case when the assumed scheduler is the *HS Scheduler* as more than one interaction can take place in a round.

5.1 Basic Notations and Statements

We introduce in this section some basic notations and statements that are useful for the study we are going to establish in this chapter.

Let $\mathcal{P}$ be any population of n agents with $n \geq 2$ running a (mediated) population protocol. Let $G = (V, E)$ be the interaction graph of this population, and let $e = (u, v)$ be any edge from E.

We recall that $\mathcal{HS}(e)$ denotes the event of having a handshake over the edge e at a given round, and $\mathbb{P}r(\mathcal{HS}(e))$ is the probability of this event.

For any set of edges $g \subset E$, we define: ξ_g (respectively $\overline{\xi_g}$) the event of obtaining a handshake for at least one edge (respectively no edge) in g, $\mathbb{P}r(\xi_g)$ the probability of this event, and $\pi(g) = \sum_{e \in g} \mathbb{P}r(\mathcal{HS}(e))$ (note that $\pi(g)$ is not a necessarily a probability since it may be > 1).

We also recall the following lemmas and corollary stated in [32]:

Lemma 5.1. *Let g be any subset of E. For any $e \in E$, we have:*

$$\mathbb{P}r\left(\mathcal{HS}(e)|\overline{\xi_g}\right) \geq \mathbb{P}r(\mathcal{HS}(e)).$$

Lemma 5.2. *For any $g \subset E$, we have:*

$$\mathbb{P}r(\xi_g) \geq \lambda \, min(1, \pi(g)), \text{ with } \lambda = 1 - e^{-1} \text{ where } e = exp(1).$$

Corollary 5.3. *For any $g \subset E$, we have:*

$$\frac{1}{\mathbb{P}r(\xi_g)} \leq \frac{e}{e-1}\max(1, \frac{1}{\pi(\xi_g)}) \leq \frac{e}{e-1}(1+\frac{1}{\pi(\xi_g)}).$$

In [32], the supposed handshake model is the one introduced in [53] where the probability of a handshake is different from our model. However, in the respective proofs of these lemmas and corollary, the value of this probability was not involved. Consequently, they remain valid and suitable for our case too.

5.2 The OR Population Protocol

In this section, we study the time equivalence of the Random Scheduler and the *HS Scheduler* with respect to the *OR* population protocol (see Section 4.6.3). This protocol computes a logical *OR* over the inputs of the agents in a population. Initially, there is only one agent in state q_1 and the rest of the population is in state q_0. Stabilization happens when all the agents become with state q_1.

As presented in the previous chapter, this protocol can describe an information broadcast that starts from one agent to reach all the members of the population. It can also describe the propagation of an epidemic in a population: starting from only one infected node, the protocol stabilizes when the whole population becomes infected.

5.2.1 The OR Population Protocol over a Complete Graph

We now study the complexity of the stabilization time of the *OR* population protocol running in a population whose interaction graph is complete. The first scenario supposes that the computation of this protocol is under the Random Scheduler. The second one supposes that this computation is under the *HS Scheduler*.

The OR Population Protocol over a Complete Graph with the Random Scheduler

Let $\mathcal{P}$ be a population with a complete interaction graph $K_n = (V, E)$. We suppose that an epidemic protocol, represented by the *OR* population protocol, is running in this population under the Random Scheduler. Let T_{K_n} be the expected number of interactions before this epidemic protocol, starting from one infected agent, infects all the other ones. We established, in [60], a characterization of the time T_{K_n} described by the following theorem.

Theorem 5.4. *Let T_{K_n} be the time needed by the OR population protocol, running in a complete interaction graph and scheduled by the Random Scheduler, to stabilize. Let $E(T_{K_n})$ be the expected value of T_{K_n}. Then:*

$$E(T_{K_n}) \sim O(n\log(n)), \text{ as } n \to \infty\,.$$

Proof. The computation process of the *OR* protocol, where the interactions are scheduled by the probabilistic Random Scheduler, can be modeled by a Markov chain.

For any $t \geq 0$, we denote by X_t the set of agents that are infected. We also define the random variable (r.v for short) x_t as the size of the set X_t, for any $t \geq 0$. The behavior of this r.v. can be modeled by an homogeneous increasing Markov chain whose set of vertices is $\{1, 2, \cdots, n\}$ and whose transitions are as depicted in Figure 5.1.

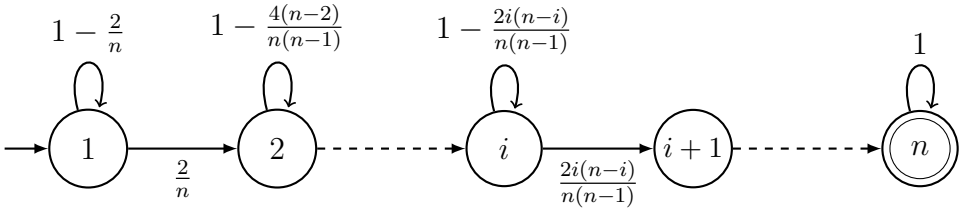

Figure 5.1: The Markov chain corresponding to the computation process of the population protocol OR under the Random Scheduler in a complete interaction graph.

The Random Scheduler is sequential, it chooses only one ordered pair of agents to interact at each step. Thus, the number of infected agents can increase by at most 1 at each computation step. In fact, the size of the set X_t increases only when the Random scheduler picks from the interaction graph an edge that has one endpoint with state q_1, and thus is in X_t, and the other endpoint with state q_0, and consequently belongs to $V \setminus X_t$.

Indeed, as the interaction graph is complete and consequently $|E| = \frac{n(n-1)}{2}$, for any $t \geq 0$, the transition probabilities are given by:

$$\mathbb{P}r\left(x_{t+1} = i \mid x_t = j\right) = \begin{cases} \frac{2(i-1)(n-i+1)}{n(n-1)}, & \text{if } j = i-1 \geq 1 \\ 1 - \frac{2(i-1)(n-i+1)}{n(n-1)}, & \text{if } j = i \geq 1 \\ 0 & \text{otherwise.} \end{cases}$$

For any $i \geq 1$, let T_i denote the time complexity of the Markov process starting at state $x_t = i$ to reach the absorbing state $x_{t'} = n$, and let $E_i = \mathbb{E}(T_i)$ denote its expected value. Then, the time complexity of the epidemic process starting from only one infected agent corresponds to T_1, and to compute its expected value E_1, we can solve the following system:

$$E_1 = 1 + \left(1 - \frac{2}{n}\right) E_1 + \frac{2}{n} E_2. \tag{5.1}$$

$\forall\, 1 \leq k \leq n-1$, we have:

$$E_k = 1 + \left(1 - \frac{2k(n-k)}{n(n-1)}\right) E_k + \frac{2k(n-k)}{n(n-1)} E_{k+1}. \tag{5.2}$$

From (5.2), we obtain:

$$\frac{2k(n-k)}{n(n-1)} E_k = 1 + \frac{2k(n-k)}{n(n-1)} E_{k+1}.$$

Yielding:

$$E_k = \frac{n(n-1)}{2k(n-k)} + E_{k+1}.$$

Hence:

$$E_1 = \sum_{i=1}^{n-2} \frac{n(n-1)}{2i(n-i)} + E_{n-1}.$$

We already have:

$$E_{n-1} = \frac{1}{\mathbb{P}r(x_{t+1} = n | x_t = n-1)} = \frac{n}{2}.$$

Thus:

$$\begin{aligned} E_1 &= \sum_{i=1}^{n-2} \frac{n(n-1)}{2i(n-i)} + \frac{n}{2} \\ &= \frac{n(n-1)}{2} \sum_{i=1}^{n-2} \frac{1}{i(n-i)} + \frac{n}{2}. \end{aligned} \tag{5.3}$$

On the other hand, we have:

$$\frac{1}{i(n-i)} = \frac{\frac{1}{n}}{i} + \frac{\frac{1}{n}}{n-i}. \tag{5.4}$$

We finally obtain:

$$\begin{aligned} E_1 &= \frac{n(n-1)}{2} \left[\frac{1}{n} \sum_{i=1}^{n-2} \frac{1}{i} + \frac{1}{n} \sum_{i=1}^{n-2} \frac{1}{n-i} \right] + \frac{n}{2} \\ &= \frac{n(n-1)}{2} \left[\frac{1}{n} H_{n-2} + \frac{1}{n} \sum_{i=n-1}^{2} \frac{1}{i} \right] + \frac{n}{2}. \end{aligned} \tag{5.5}$$

Thus, with $i' = n - i$, we get:

$$\begin{aligned} E_1 &= \frac{(n-1)}{2} \left[H_{n-2} + H_{n-1} - 1 \right] + \frac{n}{2} \\ &= \frac{(n-1)}{2} \left[2H_{n-1} - \frac{1}{n-1} - 1 \right] + \frac{n}{2} \\ &= \left[(n-1)H_{n-1} - \frac{1}{2} \right] - \frac{n-1}{2} + \frac{n}{2} \\ &= (n-1)H_{n-1}. \end{aligned} \tag{5.6}$$

Knowing that $H_n = \sum_{i=1}^{n} \frac{1}{i} \sim \log(n)$, and as T_{K_n} corresponds to T_1, we obtain:

$$E(T_{K_n}) \sim n \log(n), \text{ as } n \to \infty\,. \tag{5.7}$$

□

The OR Population Protocol over a Complete Graph with the HS Scheduler

We investigate the time needed for an epidemic protocol to stabilize if it runs in a population whose interaction graph is the complete graph $K_n = (V, E)$, and under the distributed *HS Scheduler*. The result is represented by the theorem that follows.

Theorem 5.5. *Let T_{K_n} be the time needed by the OR protocol to stabilize when running in a complete interaction graph K_n under the HS Scheduler. Let $E(T_{K_n})$ be the expected value of T_{K_n}. Then:*

$$E(T_{K_n}) \sim O(n \log(n)), \text{ as } n \to \infty .$$

Proof. This proof is based on the computation of the upper bound of the stabilization time T_{K_n}. We consider a pessimistic epidemic process where at most one new agent is infected at each computation step. The stabilization time of the epidemic protocol will be bounded by the sum of the expected time to increase the number of infected agents by one:

$$E(T_{K_n}) \leq \sum_{k=1}^{n-1} \frac{1}{\mathbb{P}r(\xi_{G_k})},$$

with G_k the set of edges linking the set of k agents with state q_1 to the $(n-k)$ agents with state q_0.

We have:

$$\pi(G_k) = \sum_{a=(u,v)\,\in\, G_k} \mathbb{P}r(\mathcal{HS}(a)).$$

As the interaction graph is a complete graph, and according to (4.11), we can write:

$$\begin{aligned} \pi(G_k) &= \sum_{a=(u,v)\,\in\, G_k} \frac{\beta}{(n-1)^2} \\ &= \frac{\beta}{(n-1)^2}\, k\,(n-k) \\ &\leq 1. \end{aligned} \tag{5.8}$$

Given (5.8) and Lemma 5.2, we get:

$$\begin{aligned} E(T_{K_n}) &\leq \sum_{k=1}^{n-1} \frac{1}{\lambda\, \pi(G_k)} \\ &\leq \sum_{k=1}^{n-1} \frac{(n-1)^2}{\lambda\, \beta\, k\,(n-k)} \\ &\leq \frac{(n-1)^2}{\lambda\, \beta} \sum_{k=1}^{n-1} \frac{1}{k\,(n-k)}. \end{aligned}$$

Applying (5.4) gives:

$$E(T_{K_n}) \leq \frac{(n-1)^2}{\lambda\, \beta} \frac{2}{n} H_{n-1}. \tag{5.9}$$

This yields:

$$E(T_{K_n}) \sim O(n \log n), \text{ as } n \to \infty . \tag{5.10}$$

□

Comparison

Thanks to the analyses we presented in the two previous sections, we can state the following result:

Corollary 5.6. *The Random Scheduler and the HS Scheduler are time equivalent with respect to the OR population protocol running in a population $\mathcal{P}$ whose interaction graph is complete.*

Proof. According to Theorem 5.4 and Theorem 5.5, the *OR* population protocol needs $O(n\log(n))$ computation steps when it is run over a complete graph under the Random Scheduler, and likewise under the Handshake Scheduler. Consequently, according to Definition 4.12, these two schedulers are time equivalent with respect to this population protocol running in a complete interaction graph.

□

5.2.2 The OR Population Protocol over a Random Graph

Let $\mathcal{P}$ be a population whose interaction graph $G = (V, E)$ is a random graph $G_{n,p}$. To guarantee that this graph is connected with high probability, we suppose that $p = \alpha\frac{\log n}{n}$ with $\alpha > 1$ [13]. We study then the stabilization time of the *OR* population protocol running in the population $\mathcal{P}$, first when being scheduled by the Random Scheduler, and then when scheduled by the *HS Scheduler*.

The OR Population Protocol over a Random Graph with the Random Scheduler

The stabilization time of the *OR* population protocol running in a $G_{n,p}$ graph, when the interactions are scheduled by the Random scheduler, is characterized as follows [60]:

Theorem 5.7. *Let $T_{G_{n,p}}$ denote the time needed by the OR protocol to stabilize when running over the random graph $G_{n,p}$ under the Random Scheduler and let $E(T_{G_{n,p}})$ denote its expected value. Then:*

$$E(T_{G_{n,p}}) \sim O(n^2), \text{ as } n \to \infty\,.$$

Proof. Let $G_{n,p} = (V, E)$ be a random graph. Using the same notation as for the case $G = K_n$, we can see that the size of the set X_t will increase if, and only if, the peer of vertices which is chosen to communicate has one end-point in X_t and the other one in $V \setminus X_t$. We also need to ensure that this peer exists.

Thus, $\forall\, 1 \leq k \leq n-1$, we have:

$$\mathbb{P}r\left(x_{t+1} = k+1 \mid x_t = k\right) = \frac{2k(n-k)}{n(n-1)} \times p. \tag{5.11}$$

Hence, as for the complete graph, if we denote by T_k the time complexity of the Markov process starting at $x_t = k$ to reach the absorbing state $x_{t'} = n$, and with $E_k = \mathbb{E}\left(T_k\right)$ then:

$$\forall 1 \leq k \leq n-1,\;\; E_k = 1 + \left(1 - \frac{2k(n-k)}{n(n-1)}p\right) E_k + \frac{2k(n-k)}{n(n-1)} p\, E_{k+1} \tag{5.12}$$

A straightforward computation yields:

$$\begin{aligned} E_1 &= \sum_{k=1}^{n-1} \frac{n(n-1)}{2k(n-k)p} \\ &= \frac{n(n-1)}{2p} \sum_{k=1}^{n-1} \frac{1}{k(n-k)} \\ &= \frac{n(n-1)}{2p} \frac{2}{n} H_{n-1} \\ &= \frac{n-1}{p} H_{n-1}. \end{aligned} \tag{5.13}$$

As $p = \alpha \frac{\log n}{n}$, we get:

$$\begin{aligned} E(T_{G_{n,p}}) &= \frac{n(n-1)}{\alpha \log n} H_{n-1} \\ &\sim \frac{n(n-1)}{\alpha} \end{aligned} \tag{5.14}$$

Hence,

$$E(T_{G_{n,p}}) \sim O(n^2), \text{ as } n \to \infty\,. \tag{5.15}$$

□

The OR Population Protocol over a Random Graph with the Handshake Scheduler

We consider now the computation of the *OR* population protocol by a population whose interaction graph is a $G_{n,p} = (V, E)$, and where interactions are orchestrated by the *HS Scheduler*. The expected value of the time needed for this computation to stabilize is provided by the following theorem.

Theorem 5.8. *Let $T_{G_{n,p}}$ denote the time needed by the OR protocol to stabilize when running over the random graph $G_{n,p}$ under the HS Scheduler and let $E(T_{G_{n,p}})$ denote its expected value. Then:*

$$E(T_{G_{n,p}}) \sim O(n \log^2(n)), \text{ as } n \to \infty\,.$$

Proof. This proof consists on characterizing the upper bound of $T_{G_{n,p}}$. Thus, we consider the same pessimistic epidemic process as in the proof of Theorem 5.5, where at most one new agent is infected. The stabilization time of the epidemic protocol will be bounded by the sum of the expected time to increase the number of infected agents by one:

$$E(T_{G_{n,p}}) \leq \sum_{k=1}^{n-1} \frac{1}{\mathbb{P}r(\xi_{G_k})},$$

where G_k denotes the set of edges existing between the k agents with state q_1 and the $(n-k)$ agents with state q_0 ($G_k \subset E$).

We define:

$$\pi(G_k) = \sum_{a \in G_k} \mathbb{P}r(\mathcal{HS}(a))$$

We are assuming that $a \in G_k$. As $G_k \subset E$, we guarantee that: $a \in E$. Thus, applying Lemma 4.4 gives:

$$\mathbb{P}r(\mathcal{HS}(a)) = \beta \frac{(1-q^{n-1})^2}{(n-1)^2\, p^2}.$$

This yields

$$\begin{aligned} \pi(G_k) &= \sum_{(u,v) \in G_k} \beta \frac{(1-q^{n-1})^2}{(n-1)^2\, p^2} \\ &= \beta \frac{(1-q^{n-1})^2}{(n-1)^2\, p^2} \, |G_k|, \end{aligned}$$

with $|G_k|$ denoting the size of the set of edges G_k.

According to Corollary 5.3, we have:

$$\frac{1}{\mathbb{P}r(\xi_{G_k})} \leq \frac{e}{e-1}\left(1 + \frac{1}{\pi(G_k)}\right).$$

This yields,

$$\begin{aligned} E(T_{G_{n,p}}) &\leq \sum_{k=1}^{n-1} \frac{e}{e-1}\left(1 + \frac{1}{\pi(G_k)}\right) \\ &\leq \frac{e}{e-1}\left(\sum_{k=1}^{n-1} 1 + \sum_{k=1}^{n-1} \frac{1}{\pi(G_k)}\right) \\ &\leq \frac{e}{e-1}\left((n-1) + \sum_{k=1}^{n-1} \frac{p^2\,(n-1)^2}{\beta\,(1-q^{n-1})^2\,|G_k|}\right) \\ &\leq \frac{e}{e-1}\left((n-1) + \frac{p^2\,(n-1)^2}{\beta\,(1-q^{n-1})^2} \sum_{k=1}^{n-1} \frac{1}{|G_k|}\right) \end{aligned}$$

Given that, $\forall\ 1 \leq k \leq n-1$, we have:

$$1 \leq |G_k| \leq k(n-k)$$

We can write:

$$E(T_{G_{n,p}}) \leq \frac{e}{e-1}\left((n-1) + \frac{p^2\,(n-1)^3}{\beta\,(1-q^{n-1})^2}\right).$$

Knowing that: $\forall\, m \geq 0$, $\forall\, 0 \leq x \leq 1$, we have:

$$1 - mx \leq (1-x)^m \leq e^{-mx}. \tag{5.16}$$

We get:

$$1 - 2q^{n-1} \leq (1 - q^{n-1})^2 \leq e^{-2q^{n-1}} \tag{5.17}$$

Which gives:

$$\frac{1}{(1-q^{n-1})^2} \leq \frac{1}{1-2q^{n-1}} \leq \frac{1}{1-2(1-p)^{n-1}} \tag{5.18}$$

Using (5.16) again, we obtain:

$$1 - (n-1)p \leq (1-p)^{n-1} \leq e^{-(n-1)p}$$

This yields:

$$\frac{1}{1-2(1-p)^{n-1}} \leq \frac{1}{1-2\,e^{-(n-1)p}} \tag{5.19}$$

Inequalities (5.18) and (5.19) give:

$$\frac{1}{(1-q^{n-1})^2} \leq \frac{1}{1-2\,e^{-(n-1)p}}. \tag{5.20}$$

Given that $p = \alpha\frac{\log n}{n}$ with $\alpha > 1$, (5.20) becomes:

$$\begin{aligned}
\frac{1}{(1-q^{n-1})^2} &\leq \frac{1}{1-2\,e^{-(n-1)\,\alpha\,\frac{\log n}{n}}} \simeq \frac{1}{1-2\,e^{-\alpha\log n}} \\
&\leq \frac{1}{1-2\,n^{-\alpha}} \\
\frac{p^2}{(1-q^{n-1})^2} &\leq \frac{\alpha^2\,\log^2 n}{n^2(1-2\,n^{-\alpha})} \\
&\leq \frac{\alpha^2\,n^\alpha\,\log^2 n}{n^2(n^\alpha-2)} \simeq \frac{\alpha^2\,\log^2 n}{n^2}.
\end{aligned}$$

This gives:

$$E(T_{G_{n,p}}) \leq \frac{e}{e-1}\left((n-1) + \frac{\alpha^2\,\log^2 n}{\beta\,n^2}(n-1)^3\right)$$

Accordingly, we have:

$$E(T_{G_{n,p}}) \sim O(n\log^2 n), \text{ as } n \to \infty\,. \tag{5.21}$$

□

Comparison

We presented above results concerning the stabilization time of the OR population protocol running in a population whose interaction graph is the random graph $G_{n,p}$. The first result concerned a scheduling under the Random Scheduler. The second one concerned a scheduling under the *HS Scheduler*.

We deduced from these results the following corollary:

Corollary 5.9. *The Random Scheduler and the Handshake Scheduler are not time equivalent with respect to the OR population protocol running over the random communication graph $G_{n,p}$.*

Proof. According to Theorem 5.7 and Theorem 5.8, the population protocol Leader Election that runs over the random communication graph $G_{n,p}$ stabilizes faster when it is under the Handshake scheduler, compared to the case where it is under the Random Scheduler. □

5.3 The Leader Election Population Protocol

We study, in this section, the stabilization time of the population protocol Leader Election that we already introduced in a previous chapter (see Section 2.2.3). Initially, all the agents start with the state L. Hence, what is the time needed by the protocol to reach a configuration where only one L remains?

5.3.1 The Leader Election over a Complete Graph

Let $\mathcal{P}$ be a population with a complete interaction graph $K_n = (V, E)$ and whose agents are executing the Leader Election protocol. Let T_{K_n} be the stabilization time to get a single leader in the population $\mathcal{P}$ and let $E(T_{K_n})$ denote the expected value of T_{K_n}.

The Leader Election over a Complete Graph under the Random Scheduler

We have from [4] the following characterization of the stabilization time of the Leader Election protocol running in a complete graph under the Random Scheduler.

Theorem 5.10. *Let T_{K_n} be the time needed by the Leader Election protocol to stabilize when running over a complete interaction graph K_n under the Random Scheduler. Let $E(T_{K_n})$ be the expected value of T_{K_n}. Then:*

$$E(T_{K_n}) \sim O(n^2), \text{ as } n \to \infty .$$

Proof. In [4], the authors considered the time T_{K_n} as being equal to the sum of all the times until two leaders among the n meet, then two leaders among the $(n-1)$ agents with state L meet ,... until finally the two last leaders L meet. We here give further details for this proof.

Let X_t denote the set of agents in state L at time t, for any $t \geq 0$, and x_t be the random variable that denotes the size of X_t. Initially $x_0 = n$, then it decreases until reaching, at some t, the absorbing state $x_t = 1$.

The size of the set X_t decreases by at most 1 at each computation step. This happens only if the Random Scheduler picks, for the interaction, an ordered pair of agents that are both with state L. The probability of this event corresponds to:

$$\begin{aligned} \forall\, 2 \leq k \leq n,\ \mathbb{P}r(x_{t+1} = k-1 | x_t = k) &= \frac{\binom{k}{2}}{\binom{n}{2}} \\ &= \frac{k(k-1)}{n(n-1)}. \end{aligned}$$

Then, $E(T_{K_n})$ can be described as:

$$\begin{aligned} E(T_{K_n}) &= \sum_{k=2}^{n} \frac{1}{P(x_{t+1} = k-1 | x_t = k)} \\ &= \sum_{k=2}^{n} \frac{n(n-1)}{k(k-1)} \\ &= n(n-1) \sum_{k=2}^{n} \frac{1}{k(k-1)}. \end{aligned} \tag{5.22}$$

On the other hand, we can write:

$$\begin{aligned} \sum_{k=2}^{n} \frac{1}{k(k-1)} &= \sum_{k=2}^{n} (\frac{1}{k-1} - \frac{1}{k}) \\ &= \sum_{k'=1}^{n-1} \frac{1}{k'} - \sum_{k=2}^{n} \frac{1}{k} \\ &= 1 - \frac{1}{n}. \end{aligned} \tag{5.23}$$

Hence, given (5.23), (5.22) becomes:

$$\begin{aligned} E(T_{K_n}) &= n(n-1)(1 - \frac{1}{n}) \\ &= (n-1)^2. \end{aligned} \tag{5.24}$$

Thus:

$$E(T_{K_n}) \sim O(n^2),\ \text{as } n \to \infty\,. \tag{5.25}$$

□

The Leader Election over a Complete Graph under the Handshake Scheduler

By analyzing the stabilizing time of the Leader Election protocol running in the population $\mathcal{P}$ that has a complete interaction graph, and where interactions are orchestrated by the *HS Scheduler*, we obtained the following theorem.

Theorem 5.11. *Let T_{K_n} be the stabilization time of the Leader Election that runs over a complete interaction graph K_n under the HS Scheduler and let $E(T_{K_n})$ be its expected value. Then:*

$$E(T_{K_n}) \sim O(n^2),\ \textit{as } n \to \infty\,.$$

Proof. We bound the expected time to have only one leader in the whole population, by allowing at most one new defeated agent per computation step. In this pessimistic process, we sum up the expected time to decrease the number of leaders agents by one. We consequently obtain:

$$E(T_{K_n}) \leq \sum_{k=2}^{n} \frac{1}{\mathbb{P}r(\xi_{G_k})}$$

where G_k denotes the graph formed by the k agents with state L (it is a complete graph), and ξ_{G_k} is the event of obtaining at least one handshake in a round for at least one edge in G_k.

On one hand, according to Lemma 5.2, we have:

$$\mathbb{P}r(\xi_{G_k}) \geq \lambda \, min(1, \pi(G_k)). \tag{5.26}$$

On the other hand, we have:

$$\begin{aligned} \pi(G_k) &= \sum_{a \in G_k} \mathbb{P}r(\mathcal{HS}(a)) \\ &= \sum_{(u,v) \in G_k} \frac{\beta}{d(u)d(v)} \\ &= \frac{\beta}{(n-1)^2} \frac{k(k-1)}{2} \\ &\leq 1. \end{aligned} \tag{5.27}$$

Inequalities (5.26) and (5.27) yield:

$$\begin{aligned} E(T_{K_n}) &\leq \sum_{k=2}^{n} \frac{1}{\lambda \, \pi(G_k)} \\ &\leq \sum_{k=2}^{n} \frac{2(n-1)^2}{\lambda \, \beta \, k(k-1)} \\ &\leq \frac{2(n-1)^2}{\lambda \, \beta} \sum_{k=2}^{n} \frac{1}{k(k-1)} \\ &\leq \frac{2(n-1)^2}{\lambda \, \beta} (1 - \frac{1}{n}). \end{aligned} \tag{5.28}$$

From the expression (5.28), we conclude that:

$$E(T_{K_n}) \sim O(n^2), \text{ as } n \to \infty \, . \tag{5.29}$$

□

Comparison

According to Theorem 5.10, the expected value of the time that the Leader Election protocol needs to stabilize when running in the population $\mathcal{P}$ under the Random Scheduler is $O(n^2)$. This is the same expected value of the time that this protocol needs when running in the population $\mathcal{P}$ under the *HS Scheduler*, according to Theorem 5.11. We consequently obtain the following corollary:

Corollary 5.12. *The Random Scheduler and the HS Scheduler are time equivalent with respect the Leader Election population protocol running over a complete interaction graph.*

5.3.2 The Leader Election over a Random Graph

In this section, we suppose that the interaction graph of the population $\mathcal{P}$ where the Leader Election protocol runs, is the random graph $G_{n,p}$. To guarantee that this random graph is connected with high probability, we suppose that $p = \alpha\frac{\log n}{n}$ with $\alpha > 1$. We study then the complexity of the stabilization time of the Leader Election protocol in this case.

The Leader Election over a Random Graph Scheduled by the Random Scheduler

We focus, in this section, on the scenario where the interactions in the population $\mathcal{P}$ are handled by the Random Scheduler. The expected value of the stabilization time of the Leader Election protocol in this case is defined by the following theorem.

Theorem 5.13. *Let $T_{G_{n,p}}$ denote the time needed by the Leader Election population protocol to stabilize when executed over the random graph $G_{n,p}$ under the Random Scheduler. Let $E(T_{G_{n,p}})$ denote the expected value of $T_{G_{n,p}}$. Then:*

$$E(T_{G_{n,p}}) \sim O(\frac{n^3}{\log n}),\ \textit{as } n \to \infty\,.$$

Proof. Let X_t denote the set of agents in state L at time t for any $t \geq 0$, and x_t be the random variable that denotes the size of X_t. Initially $x_0 = n$, then it can decrease until reaching, at some t, the absorbing state $x_t = 1$.

The size of the set X_t decreases if the interaction happens between a pair of agents that are both in state L and that are necessarily linked by an edge from the interaction graph. The probability of such event corresponds to:

$$\begin{aligned}\mathbb{P}r(x_{t+1} = k-1 | x_t = k) &= \frac{\binom{k}{2}}{\binom{n}{2}} \times p \\ &= \frac{k(k-1)}{n(n-1)} \times p.\end{aligned}$$

$$\begin{aligned}E(T_{G_{n,p}}) &= \sum_{k=2}^{k=n} \frac{1}{\mathbb{P}r(x_{t+1} = k-1 | x_t = k)} \\ &= \sum_{k=2}^{k=n} \frac{n(n-1)}{p\,k(k-1)} \\ &= \frac{n(n-1)}{p} \sum_{k=2}^{k=n} \frac{1}{k(k-1)} \\ &= \frac{n(n-1)}{p}(1 - \frac{1}{n}) \\ &= \frac{(n-1)^2}{p} \qquad (5.30)\end{aligned}$$

As $p = \alpha\frac{\log n}{n}$, (5.30) becomes:

$$E(T_{G_{n,p}}) \quad = \quad \frac{n(n-1)^2}{\alpha \log n} \sim O(\frac{n^3}{\log n}), \text{ as } n \to \infty . \tag{5.31}$$

□

The Leader Election over a Random Graph Scheduled by the Handshake Scheduler

We focus on the computation of the Leader Election protocol, under the *HS Scheduler*, by the population $\mathcal{P}$ whose interaction graph is a $G_{n,p}$.

We consider a pessimistic scenario where at each computation step there is at most one defeated leader. If we define X_t the set of agents in state L at time t, for any $t \geq 0$, and x_t the random variable that denotes the size of X_t. Then, we have initially $x_0 = n$. The size of the set X_t decreases until reaching, at some t, the absorbing state $x_t = 1$.

At a computation step, the size of X_t can decrease by at most 1. This happens when there is a rendezvous over an edge of the random interaction graph whose both endpoints are with state L. We define the probability of this event as:

$$\mathbb{P}r(x_{t+1} = k-1 | x_t = k) \quad = \quad \beta \frac{(1-q^{n-1})^2}{p^2\,(n-1)^2}.$$

We can consequently write:

$$\begin{aligned} E(T_{G_{n,p}}) \quad &\leq \quad \sum_{k=2}^{k=n} \frac{1}{\mathbb{P}r(x_{t+1} = k-1 | x_t = k)} \\ &\leq \quad \sum_{k=2}^{k=n} \frac{p^2\,(n-1)^2}{\beta\,(1-q^{n-1})^2} \\ &\leq \quad \frac{p^2\,(n-1)^3}{\beta\,(1-q^{n-1})^2}. \end{aligned} \tag{5.32}$$

As $p = \alpha\frac{\log n}{n}$, (5.32) becomes:

$$E(T_{G_{n,p}}) \quad \leq \quad \frac{(n-1)\,\alpha^2\,\log^2(n)}{\beta\,(1-q^{n-1})^2} \sim O(n\log^2(n)), \text{ as } n \to \infty . \tag{5.33}$$

Given (5.33), we can state the following theorem:

Theorem 5.14. *Let $T_{G_{n,p}}$ denote the time needed by the Leader Election population protocol to stabilize when executed over the random graph $G_{n,p}$ under the Handshake Scheduler. Let $E(T_{G_{n,p}})$ denote the expected value of $T_{G_{n,p}}$. Then:*

$$E(T_{G_{n,p}}) \sim O(n\log^2(n)), \text{ as } n \to \infty .$$

Comparison

Given Theorem 5.13 and Theorem 5.14, we can conclude that the computation of the Leader Election protocol by the agents of the population $\mathcal{P}$ stabilizes faster when it is scheduled by the *HS Scheduler*, compared to a scheduling under the Random Scheduler. We consequently obtain this corollary:

Corollary 5.15. *The Random Scheduler and the HS Scheduler are not time equivalent with respect to the Leader Election protocol running in a population whose interaction graph is a random graph $G_{n,p}$.*

5.4 The Maximal Matching Mediated Population Protocol

We recall that, according to Definition 1.25, a matching M of a graph $G = (V, E)$ is a set of edges from E such that each two edges from M do not share any vertex. This matching is maximal if by adding any edge to it, it will not be a matching any more (see Definition 1.26).

The mediated population protocol that computes a Maximal Matching in a graph G was introduced in [29] and can be described by $(X_M, Y_M, Q_M, S_M, I_M, O_M, \delta_M, \iota_M, \omega_M, r_M)$ with:

- $X_M = \{0\}$, $Y_M = \{0, 1\}$,
- $Q_M = \{q_0, q_1\}$, $S_M = \{0, 1\}$,
- $I_M(0) = q_0$, $O_M(q_0) = 0$ and $O_M(q_1) = 1$,
- $\iota_M(0) = 0$, $\omega_M(0) = 0$ and $\omega_M(1) = 1$,
- $\delta_M(q_0, q_0, 0) = (q_1, q_1, 1)$.
- r_M: "Get each edge $e \in E$ such that $\omega_M(s_e) = 1$ where s_e is the state of the edge e."

Initially, all the agents start with state q_0 and all the edges with state 0. If an interaction happens between two agents q_0 linked by an edge with state 0, both agents update their states to q_1 and the edge linking them becomes with state 1. The protocol stabilizes when the matching formed by the edges with state 1 is maximal, that is there no more pairs of nodes with state q_0 linked by an edge with state 0.

5.4.1 The Maximal Matching over a Complete Graph

Let $\mathcal{P}$ be a population whose interaction graph $G = (V, E)$ is thereafter supposed to be the complete graph of n nodes K_n. A computation of the time needed for the Maximal Matching to stabilize, when running in this population, is established through the following analyses for two scenarios: the scheduler is a Random Scheduler, or the scheduler is the *HS Scheduler*.

The Maximal Matching over a Complete Graph with the Random Scheduler

By analyzing the computation process of the Maximal Matching protocol in the population $\mathcal{P}$ where the interactions are under the control of the sequential Random Scheduler, we obtained the following theorem:

Theorem 5.16. *Let T_{K_n} be the stabilization time of the Maximal Matching mediated population protocol that runs over a complete interaction graph K_n under the Random Scheduler. Let $E(T_{K_n})$ be its expected value. Then:*

$$E(T_{K_n}) \sim O(n^2), \text{ as } n \to \infty .$$

Proof. The process of selecting the edges of the Maximal Matching can be modeled by a Markov chain. For any $t \geq 0$, we denote by X_t, the set of nodes with state q_0. We also define the random variable x_t as the size of the set X_t, for any $t \geq 0$. Initially, we have $x_0 = n$. Then, the size of the set X_t decreases by 2 only if the picked edge by the Random Scheduler has both extremities with state q_0, that is both extremities belong to X_t. We would mention that, $\forall t \geq 0$, the graph formed by the set of nodes of X_t and the edges existing between them, is a complete graph.

Accordingly, the behavior of the random variable x_t can be modeled by an homogeneous Markov chain whose set of vertices is $\{n, n-2, n-4, ..., n-2\lfloor \frac{n}{2} \rfloor\}$ and whose transitions are given by:

$$\mathbb{P}r(x_{t+1} = j | x_t = i) = \begin{cases} \frac{i(i-1)}{n(n-1)} & if\ j = i-2 \\ 1 - \frac{i(i-1)}{n(n-1)} & if\ j = i \geq 0 \\ 0 & otherwise. \end{cases}$$

For any $i \geq 0$, let T_i denote the time complexity of the Markov process $x_{t \geq 0}$ starting at state $x_t = i$ to reach the absorbing state $x_{t'} \leq 1$, and let $E_i = E(T_i)$ denote its expected value. Thus, the time complexity of the computation of the mediated population protocol Maximal Matching is T_n, and E_n its expected value.

For any $0 \leq i \leq \lfloor \frac{n}{2} \rfloor - 1$, we have:

$$E_{n-2i} = 1 + (1 - \frac{(n-2i)(n-2i-1)}{n(n-1)})E_{n-2i} + \frac{(n-2i)(n-2i-1)}{n(n-1)} E_{n-2(i+1)}.$$

This gives,

$$E_{n-2i} = \frac{n(n-1)}{(n-2i)(n-2i-1)} + E_{n-2(i+1)}.$$

This yields,

$$\begin{aligned} E_n &= \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor - 1} \frac{n(n-1)}{(n-2i)(n-2i-1)} \\ &= n(n-1) \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor - 1} \frac{1}{(n-2i)(n-2i-1)} \\ &= n(n-1)\Big(\sum_{i=0}^{\lfloor \frac{n}{2} \rfloor - 1} \frac{1}{n-2i-1} - \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor - 1} \frac{1}{n-2i} \Big) \\ &= n(n-1) \sum_{i=0}^{2\lfloor \frac{n}{2} \rfloor - 1} \frac{(-1)^{i+1}}{n-i} \\ &\leq n(n-1) \ln 2. \end{aligned} \tag{5.34}$$

Given (5.34), we can deduce that:

$$E(T_{K_n}) \sim O(n^2), \text{ as } n \to \infty . \tag{5.35}$$

□

The Maximal Matching over a Complete Graph with the *HS Scheduler*

We suppose now that the interactions in the population $\mathcal{P}$ are handled by the distributed *HS Scheduler*. We investigate the time needed for the mediated protocol Maximal Matching to stabilize when running in $\mathcal{P}$. The result is described by the following theorem:

Theorem 5.17. *Let T_{K_n} be the stabilization time of the Maximal Matching mediated population protocol that runs over a complete interaction graph K_n under the HS Scheduler. Let $E(T_{K_n})$ be the expected value of T_{K_n}. Then:*

$$E(T_{K_n}) \sim O(n^2), \text{ as } n \to \infty .$$

Proof. We suppose that the computation of the Maximal Matching protocol under the *HS Scheduler* proceeds as follows. At each computation step, the *HS Scheduler* picks at most one ordered pair of agents that are both with state q_0. Supposing that there is at most one handshake in a round happening between two agents in state q_0, then at each round, the number of agents with state q_0 decreases by 2.

$$E(T_{K_n}) \leq \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \frac{1}{\mathbb{P}r(\xi_{C_{n-2k}})}. \tag{5.36}$$

where C_{n-2k} the set of edges existing between the remaining $(n-2k)$ agents with state q_0 (it is a complete graph of $(n-2k)$ nodes).

$$\begin{aligned} \pi(C_{n-2k}) &= \sum_{a \in C_{n-2k}} \mathbb{P}r(\mathcal{HS}(a)) \\ &= \frac{\beta}{(n-1)^2} \frac{(n-2k)(n-2k-1)}{2} \\ &\leq 1. \end{aligned} \tag{5.37}$$

Given (5.37) and (5.36), and based on Lemma 5.2, we obtain:

$$\begin{aligned} E(T_{K_n}) &\leq \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor - 1} \frac{1}{\lambda\, \pi(C_{n-2k})} \\ &\leq \frac{2(n-1)^2}{\lambda\, \beta} \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor - 1} \frac{1}{(n-2k)(n-2k-1)} \\ &\leq \frac{2(n-1)^2}{\lambda\, \beta} \Big(\sum_{k=0}^{\lfloor \frac{n}{2} \rfloor - 1} \frac{1}{n-2k-1} - \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor - 1} \frac{1}{n-2k} \Big) \\ &\leq \frac{2(n-1)^2}{\lambda\, \beta} \sum_{k=0}^{2\lfloor \frac{n}{2} \rfloor - 1} \frac{(-1)^{k+1}}{n-k} \\ &\leq \frac{2(n-1)^2}{\lambda\, \beta}\, \ln 2. \end{aligned} \tag{5.38}$$

According to (5.38),

$$E(T_{K_n}) \sim O(n^2), \text{ as } n \to \infty\,. \tag{5.39}$$

□

Comparison

According to Theorem 5.16 and to Theorem 5.17, the Maximal Matching protocol needs a stabilization time which is a $O(n^2)$ when running in the population $\mathcal{P}$, either under the Random Scheduler, or under the *HS Scheduler*. We consequently get:

Corollary 5.18. *The HS Scheduler is time equivalent to the Random Scheduler with respect to the Maximal Matching mediated population protocol running over a complete interaction graph.*

5.4.2 The Maximal Matching over a Random Graph

We suppose now that $\mathcal{P}$ is a population that has a random interaction graph, which is $G_{n,p} = (V, E)$ with $p = \alpha \frac{\log n}{n}$ and $\alpha \geq 1$. We investigate on the time needed by the Maximal Matching protocol running in this population to stabilize. As in the previous sections, we consider two scheduling scenarios: under the Random Scheduler, and under the *HS Scheduler*.

The Maximal Matching over a Random Graph under the Random Scheduler

We describe the expected value of the stabilization time of the Maximal Matching protocol running in the population $\mathcal{P}$ under the Random Scheduler through the following theorem:

Theorem 5.19. *Let $T_{G_{n,p}}$ denote the time needed by the Maximal Matching mediated population protocol to stabilize while running over the random graph $G_{n,p}$ under the Random Scheduler. Let $E(T_{G_{n,p}})$ denote the expected value of $T_{G_{n,p}}$. Then:*

$$E(T_{G_{n,p}}) \sim O(\frac{n^3}{\log n}), \text{ as } n \to \infty\,.$$

Proof. Using the same notation as for the case $G = K_n$, we have initially the set X_0 consists of all the agents of the population as they all start with state q_0. Then, $\forall t \geq 0$, the size of the set X_t decreases by 2 if, and only if, the two communicating nodes are picked from X_t and obviously an edge from E is linking them. The probability of this event can be described by:

$$\mathbb{P}r\left(x_{t+1} = i-2 \mid x_t = i\right) = \frac{i(i-1)}{n(n-1)} \times p. \tag{5.40}$$

Hence, as for the complete graph, we denote by T_i the time complexity of the Markov process starting at $x_t = i$ to reach the absorbing state $x_{t'} \leq 1$ (we are aware that in case of random graph, the absorbing state could be greater than 1, but we consider the minimum value that the set of agents with state q_0 can reach). We also define $E_i = \mathbb{E}\left(T_i\right)$.

Then, for any $0 \leq i \leq \lfloor \frac{n}{2} \rfloor$, we have:

$$E_{n-2i} = 1 + (1 - \frac{(n-2i)(n-2i-1)}{n(n-1)}\, p) E_{n-2i} + \frac{(n-2i)(n-2i-1)}{n(n-1)}\, p\, E_{n-2(i+1)}.$$

This gives:

$$E_{n-2i} = \frac{n(n-1)}{(n-2i)(n-2i-1)\, p} + E_{n-2(i+1)}.$$

A same reasoning as for the previous case yields:

$$\begin{aligned} E_n &= \frac{n(n-1)}{p} \sum_{i=0}^{2\lfloor \frac{n}{2} \rfloor + 1} \frac{(-1)^{i+1}}{n-i} \\ &\leq \frac{n(n-1)}{p} \ln 2 \\ &\leq \frac{n^2(n-1)}{\alpha \log n} \ln 2. \end{aligned} \tag{5.41}$$

Given (5.41), we can state that:

$$E(T_{G_{n,p}}) \sim O(\frac{n^3}{\log n}), \text{ as } n \to \infty\,. \tag{5.42}$$

□

The Maximal Matching over a Random Graph under the HS Scheduler

We suppose now that the pairwise interactions in the population $\mathcal{P}$ are under the control of the *HS Scheduler*. Then, a characterization of the expected value of the time needed by the Maximal Matching, running in the population $\mathcal{P}$, to stabilize is given by the theorem that follows.

Theorem 5.20. *Let $T_{G_{n,p}}$ denote the time needed by the Maximal Matching protocol to stabilize when executed over the random graph $G_{n,p}$ under the HS Scheduler. Let $E(T_{G_{n,p}})$ denote the expected value of $T_{G_{n,p}}$. Then:*

$$E(T_{G_{n,p}}) \sim O(n \log^2(n)), \text{ as } n \to \infty\,.$$

Proof. Let X_t be the set of agents with state q_0 and x_t the size of this set, $\forall t \geq 0$. Initially, as all the agents of the population $\mathcal{P}$ start with state q_0, we have $x_0 = n$. Then, at each computation step, we consider the pessimistic scenario where the size of the set X_t can either decrease by 2 or still unaltered. x_t decreases by 2, only if a handshake happens between two agents with state q_0 and that are necessarily linked by an edge from the random interaction graph. The probability of this event is as follows:

$$\forall\, 0 \leq i \leq \lfloor \frac{n}{2} \rfloor - 1,\ \mathbb{P}r\left(x_{t+1} = n - 2(i+1) | x_t = n - 2i\right) \quad = \quad \beta \frac{(1-q^{n-1})^2}{p^2\,(n-1)^2}.$$

Hence, if we bound the expected time to obtain a Maximal Matching of the interaction graph, by allowing at most one new edge in the matching per computation step, we can sum up the expected time to decrease the number nodes with state q_0 by 2.

$$\begin{aligned} E(T_{G_{n,p}}) &\leq \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor - 1} \frac{1}{\mathbb{P}r(x_{t+1} = n - 2(i+1) | x_t = n - 2i)} \\ &\leq \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor - 1} \frac{p^2\,(n-1)^2}{\beta\ (1-q^{n-1})^2} \\ &\leq \frac{p^2\,(n-1)^2}{\beta\ (1-q^{n-1})^2} (\lfloor \frac{n}{2} \rfloor - 1). \end{aligned} \tag{5.43}$$

Given the fact we supposed $p = \alpha \frac{\log(n)}{n}$, we obtain from inequality (5.43):

$$E(T_{G_{n,p}}) \sim O(n \log^2(n)),\ \text{as } n \to \infty. \tag{5.44}$$

□

Comparison

Based on the analyses we established in the two previous section, we can state the following corollary:

Corollary 5.21. *The Random Scheduler and the HS Scheduler are not time equivalent with respect to the Maximal Matching protocol running in a population whose interaction graph is a random graph $G_{n,p}$.*

Proof. According to Theorem 5.19, the Maximal Matching protocol running in the population $\mathcal{P}$ under the Random Scheduler has a stabilization time whose expected value is $O\left(\frac{n^3}{\log(n)}\right)$. However, when the computation in $\mathcal{P}$ is under the *HS Scheduler*, the expected value of the stabilization time of this protocol is $O(n \log^2(n))$. We can thus deduce that the computation of the Maximal Matching in a random interaction graph stabilizes faster when it is under the scheduling of the *HS Scheduler*. Consequently, these two schedulers are not time equivalent with respect to the Maximal Matching protocol running in the population $\mathcal{P}$ (whose interaction graph is $G_{n,p}$). □

5.5 Time Complexity with High Probability

In this section, we continue the study of the three protocols we studied above: the *OR* population protocol, the Leader Election and the Maximal Matching. However, we will focus now on a characterization with high probability of the stabilization time of these protocols when running under the *HS Scheduler*. We recall that with high probability means with probability $(1 - o(\frac{1}{n}))$.

5.5.1 The Leader Election Population Protocol

We consider a population $\mathcal{P}$ whose interaction graph is complete and where interactions are scheduled by the *HS Scheduler*. Supposing that the Leader Election protocol is running in $\mathcal{P}$, a description of its stabilization time, with high probability, is given by this theorem:

Theorem 5.22. *The time needed for the Leader Election population protocol to stabilize when it runs in a complete interaction graph and under the HS Scheduler is, with high probability,* $O(n)$.

Proof. Let V_L (respectively V_F) denote the set of agents with state L (respectively F) and $|V_L|$ (respectively $|V_F|$) the size of this set. Initially, $|V_L| = n$ as all the agents start with state L. When the Leader Election population protocol stabilizes, only one leader remains in the whole population which leads to: $|V_L| = 1$ and $|V_F| = n - 1$.

Let $G_L = (V_L, E_L)$ be the graph where the set of vertices consists of the set V_L, and the set of edges E_L is the set of links existing between the agents with state L. For the following analysis, the graph G_L should satisfy the following property: "At each step of the protocol's computation, G_L is a connected subgraph of the communication graph G". The hypothesis that the interaction graph $G = K_n$ guarantees the satisfaction of this condition.

Let $\mathbb{P}r\left(\mathcal{HS}_{G_L}\right)$ be the probability of having at least one handshake over G_L. According to Lemma 4.3 stated in the previous chapter, we have the following result:

$$\mathbb{P}r\left(\mathcal{HS}_{G_L}\right) \geq 1 - e^{\frac{-\beta}{2}}.$$

Let f be any round (phase) of the protocol's computation. f is called a *good (successful) round* if there have been at least one handshake between two agents with state L that took place during f. Otherwise, f is called a *bad (unsuccessful) round*.

$\forall\, t > 0$, let f_t denote the t^{th} round and let $X(t)$ denote the size of the set of successful rounds among the t rounds: $f_1, f_2, ..., f_t$. Then, we have:

$$E(X(t)) \geq (1 - e^{(-\frac{\beta}{2})})t.$$

According to the Chernoff bound (see Definition 1.68), we have:

$$\forall a > 0,\ \mathbb{P}r\left(X(t) \leq E(X(t)) - a\right) \leq 2\, e^{(-\frac{a^2}{3E(X(t))})}.$$

This yields,

$$\mathbb{P}r\left(X(t) \leq (1 - e^{(-\frac{\beta}{2})})t - a\right) \leq 2\, e^{\left(-\frac{a^2}{3(1-e^{(-\frac{\beta}{2})})t}\right)} \leq 2\, e^{\left(-\frac{a^2}{3t}\right)}.$$

For $t = \frac{2(n-1)}{1-e^{(-\frac{\beta}{2})}}$ and $a = n$, we get:

$$\mathbb{P}r(X(t) \leq n-2) \leq 2\,e^{(-\frac{n}{6})} = o(\frac{1}{n}).$$

Hence, we can state that with high probability, at time $t = \frac{2(n-1)}{1-e^{(-\frac{\beta}{2})}}$, we have $(n-1)$ successful rounds that already took place.

On the other hand, to reach $|V_L| = 1$, at most $(n-1)$ good rounds are needed, because at each good round there is at least one leader from V_L that is defeated.

Thus, the time needed for the Leader Election protocol to stabilize when $G = K_n$ and the imposed scheduler is the *HS Scheduler* is, with high probability, $O(n)$. □

5.5.2 The OR Population Protocol

Let $\mathcal{P}$ be a population whose interaction graph is complete and where pairwise interactions take place according to the *HS Scheduler*. We suppose that the OR population protocol is running in $\mathcal{P}$. A description of its stabilization time, with high probability, is presented in the following theorem:

Theorem 5.23. *The time needed for the OR population protocol to stabilize when the interaction graph $G = K_n$ and under the HS Scheduler is, with high probability, $O(n)$.*

Proof. Let $G = (V, E)$ be the communication graph with $V = V_1 \cup V_0$. V_1 (respectively V_0) denotes the set of agents with state q_1 (respectively q_0) and $|V_1|$ (respectively $|V_0|$) the size of this set. Initially, only one agent is supposed to be aware of the information, then $|V_1| = 1$ and $|V_0| = n-1$. The protocol stabilizes when $|V_1| = n$ and $|V_0| = 0$.

Let $G' = (V, E')$ be the graph formed by the set of initial vertices V, and the set of edges $E' \subset E$, a restriction of E that contains only the edges existing between agents with different states. More specifically, any edge from E' should have one extremity from V_0 and the second from V_1.

For the following analysis, the graph G' should satisfy the following property: "At each step of the protocol's computation, G' is a connected subgraph of the communication graph G''. If we initially suppose that G is a complete graph, we can guarantee that, at each step of the computation of the protocol, G' is a connected subgraph of G.

For any round f of the protocol's computation, f is called a *good round* if there have been at least one handshake over G'. Otherwise, f is called a *bad (unsuccessful) round*.

Let $\mathbb{P}r\left(\mathcal{HS}_{G'}\right)$ be the probability of having at least one handshake over G', which corresponds to the probability of a good round. According to Lemma 4.3, we have:

$$\mathbb{P}r\left(\mathcal{HS}_{G'}\right) \geq 1 - e^{\frac{-\beta}{2}}\ .$$

$\forall\, t > 0$, let f_t denote the t^{th} round and let $X(t)$ denote the size of the set of successful rounds among the t rounds: $f_1, f_2, ..., f_t$. Thus, we can have:

$$E(X(t)) \geq (1 - e^{(-\frac{\beta}{2})})t.$$

The same reasoning as the previous section, using the Chernoff bound, yields:

$$\forall a > 0,\ \mathbb{P}r\left(X(t) \leq (1 - e^{(-\frac{\beta}{2})})t - a\right) \leq 2\,e^{(-\frac{a^2}{3t})}.$$

Now, by assuming that $t = \frac{2(n-1)}{1-e^{(-\frac{\beta}{2})}}$ and $a = n$, we obtain:

$$\mathbb{P}r\left(X(t) \leq n-2\right) \quad \leq \quad 2\,e^{(-\frac{n}{6})} = o(\frac{1}{n}). \tag{5.45}$$

If we consider the pessimistic process, where at each successful round, only one agent goes from state q_0 to state q_1, we can notice that $(n-1)$ successful rounds are needed to reach a whole population with q_1. Consequently, the OR protocol needs at most $(n-1)$ successful rounds to stabilize.

According to (5.45), at $t = \frac{2(n-1)}{1-e^{(-\frac{\beta}{2})}}$ and with high probability, wa have $(n-1)$ successful rounds that already took place over the communication graph.

This implies that the stabilization time of the OR protocol running over a complete graph, and scheduled by the *HS Scheduler* is, with high probability, $O(n)$. □

5.5.3 The Maximal Matching Mediated Population Protocol

Theorem 5.24. *The Maximal Matching mediated population protocol running under the HS Scheduler and over a complete interaction graph needs, with high probability, $O(n)$ computation steps to stabilize.*

Proof. Let V_1 (respectively V_0) denote the set of agents with state q_1 (respectively q_0) and $|V_1|$ (respectively $|V_0|$) the size of this set. Initially, all the agents start with state q_0, hence: $|V_1| = 0$ and $|V_0| = n$.

As the interaction graph is supposed to be complete, the protocol stabilizes when:

- $|V_1| = n$ and $|V_0| = 0$, if n is even,
- $|V_1| = n-1$ and $|V_0| = 1$, if n is odd.

Let $G_{V_0} = (V_0, E_0)$ be the graph formed by the set of vertices V_0, and the set of edges E_0 which is the set of links existing between the agents of V_0. For the following analysis, the graph G_{V_0} should satisfy the following property: "At each step of the protocol's computation, G_{V_0} is a connected subgraph of the communication graph G". This property is satisfied in this case, as the communication graph is supposed to be complete.

Let $\mathbb{P}r\left(\mathcal{HS}_{V_0}\right)$ be the probability of having at least one handshake over G_{V_0}. A reasoning similar to those already presented above gives:

$$\mathbb{P}r\left(\mathcal{HS}_{V_0}\right) \geq 1 - e^{\frac{-\beta}{2}}\ .$$

Let f be any round (phase) of the protocol's computation. f is called a *good (successful) round* if there have been at least one handshake between two agents from V_0 during f. Otherwise, f is called a *bad (unsuccessful) round.*

As in the previous proofs, $\forall\, t > 0$, f_t denotes the t^{th} round and $X(t)$ denotes the size of the set of successful rounds among the t rounds: $f_1, f_2, ..., f_t$. Then, we have:

$$E(X(t)) \geq (1 - e^{(-\frac{\beta}{2})})t.$$

Using the same Chernoff bound as above, we get:

$$\forall a > 0,\ \mathbb{P}r\left(X(t) \leq (1 - e^{(-\frac{\beta}{2})})t - a\right) \leq 2\,e^{(-\frac{a^2}{3t})}.$$

For $t = \frac{n-1}{1-e^{(-\frac{\beta}{2})}}$ and $a = \frac{n}{2}$, we get:

$$\mathbb{P}r\left(X(t) \leq \frac{n}{2} - 1\right) \leq 2\,e^{(-\frac{n}{12})} = o(\frac{1}{n})\,. \tag{5.46}$$

Now, if we consider the pessimistic process where, at each successful round, only one new edge is added to the maximal matching of the communication graph. In this process, $\frac{n}{2}$ good rounds are needed to reach stabilization. This implies that the maximal matching mediated population protocol needs at most $\frac{n}{2}$ successful rounds to stabilize.

Yet, according to (5.46), we have: with high probability, at time $t = \frac{n-1}{1-e^{(-\frac{\beta}{2})}}$, $(\frac{n}{2} - 1)$ successful rounds have already took place.

Consequently, we conclude that, the Maximal Matching protocol needs, with high probability, $O(n)$ time to stabilize.

□

5.6 Conclusion

We summarize in Table 5.1 the results we obtained in this chapter. In fact, we considered three protocols: the *OR* protocol, the Leader Election protocol and the Maximal Matching protocol. We analyzed the complexity of the time each needed to stabilize for two scheduling scenarios: running under the Random Scheduler, or running under the *HS Scheduler*.

We found that the Random scheduler and the *HS Scheduler* are time equivalent with respect to these three protocols in case the interaction graph is complete. However, they are not when the interaction graph is a random graph. Indeed, in this case of interaction graph, the protocols stabilize faster when they are under the scheduling of the *HS Scheduler*.

	Graph's Structure		**Scheduler**		**Time Complexity**
	K_n	$G_{n,p}$	Random	*HS*	
OR	×		×		$O(n\log(n))$
	×			×	$O(n\log(n))$
		×	×		$O(n^2)$
		×		×	$O(n\log^2(n))$
Leader Election	×		×		$O(n^2)$
	×			×	$O(n^2)$
		×	×		$O(\frac{n^3}{\log n})$
		×		×	$O(n\log^2(n))$
Maximal Matching	×		×		$O(n^2)$
	×			×	$O(n^2)$
		×	×		$O(\frac{n^3}{\log n})$
		×		×	$O(n\log^2(n)$

Table 5.1: Summary of time analyses of some protocols under the Random Scheduler and the *HS Scheduler*

We also proved in this chapter, that with high probability, these three protocols need $O(n)$ computation step to stabilize when running in a complete interaction graph under the *HS Scheduler*.

Chapter 6

The Fair Protocol Aware HS Scheduler

Population protocols are defined as being protocols that do not halt, but only stabilize [4]. However, introducing termination to this model would be interesting. The termination property is basic for distributed systems. Also, given the constraining characteristics of the systems for which the population protocols are designed, this property could help the agents preserve their energies. In [55, 56], the authors suppose that the agents can have access to some global knowledge via an oracle called an Absence Detector. Thanks to this knowledge, they design halting protocols where all the agents eventually reach some special states from which they stop participating in effective interactions.

In this chapter, we focus on the Leader Election protocol with absence detector. We study its stabilization time when its computation takes place under the Random Scheduler, and then under the *HS Scheduler*.

We then focus on the question: could an agent halt while having only a local knowledge based on its current state? For this purpose, we extend a subclass of population protocols (and the models extending it), with a local detection termination that enables each agent to detect if it reached a final state. A final state is an irreversible state that does not appear in effective transition rules. We also introduce, the Protocol Aware HS Scheduler. This is slight modified version of the *HS Scheduler* that allows agents with final states to stop participating in the protocol's computation and terminate. We prove that this scheduler is fair with probability 1. We then present some time complexity analyses of the Leader Election and the Maximal Matching protocols under this scheduler.

6.1 Population Protocols with Absence Detector

Michail et al. extend the basic population protocols with a cover-time service which is capable of detecting if a walking state has covered the whole network (in term of states and not agents). Thanks to this service, introducing termination to the population protocols becomes possible. The authors then reduce this service to an oracle model: all the agents of the population will be connected to a powerful oracle-node able to detect if any state is present or not in the population. This special oracle-node is called absence detector and this new extension of the population protocols is called population protocols with absence detector.

6.1.1 The Model of Population Protocols with Absence Detector

The model of population protocols with absence detector is described as a basic population protocol for which we add a detection transition function representing the interaction rules of the agents with the absence detector. Formally, it can be specified as follows:

Definition 6.1. (**Population Protocol with Absence Detector**) A population protocol with absence detector is a 7−tuple $(X, Y, Q, I, \omega, \delta, \gamma)$ where X, Y and Q are finite sets and:

- X and Y are respectively the input and output alphabets,
- Q is the set of states,
- I: $X \to Q$ is the input function,

- δ: $Q \times Q \to Q \times Q$ is the transition function,
- γ: $Q \times \{0,1\}^{|Q|} \to Q$ is the detection transition function.

This model supposes a complete interaction graph. Let $G = (V, E)$ denote the interaction graph over which a protocol with absence detector runs. V is the set of nodes representing the n agents of the population in addition to the absence detector. The size of the set V is: $|V| = n + 1$. And E is the set of edges representing all the permissible communication links that are in this case linking each node from the set V to all the other nodes of this set.

The absence detector has a special state which is the absence vector that indicates the presence or not of each state from the population in the current configuration. The absence vector consists of a vector $a \in \{0,1\}^{|Q|}$ such as, $\forall q \in Q$, $a[q] = 1$ iff q is absent in the current configuration of the population.

Based on the absence detector, Michail et al. introduce some termination to these protocols. A population protocol with absence detector is halting if, Q includes two special subsets Q_{h_accept} and Q_{h_reject} such that: every agent eventually reaches some state $q \in Q_{h\_accept}$ (respectively $q \in Q_{h\_reject}$). Once this happens, the agent halts, that is: it stops participating in effective interactions. If $q \in Q_{h\_accept}$ (respectively $q \in Q_{h\_reject}$), then its output is 1 (respectively 0). Note that, halting in this context does not mean that the agent stops interacting with the others, but only that these interactions will not lead to any states update.

6.1.2 The Halting Leader Election Population Protocol with Absence Detector

In [55, 56], the authors present a halting population protocol that elects a leader in a population. We denote this protocol $LE_{AD} = (X, Y, Q, I, \omega, \delta, \gamma)$ with:

- $X = \{1\}$,
- $Q = \{L, F, L_{halt}, F_{halt}\}$,
- $I(1) = F$,
- δ defining only one effective transition rule which is: $\delta(L, F) = (L, F_{halt})$, and
- $\gamma : \begin{cases} (F, a) \mapsto L, & \text{if } a[L] = 1, \\ (L, a) \mapsto L_{halt}, & \text{if } a[F] = 1. \end{cases}$

The authors do not specify the output alphabet and the output function as they consider them meaningless for this setting.

All the agents of the population have as initial input 1, which is mapped to the follower state F. The leader state L is assigned to the first agent that interacts with the absence detector according to the application of the detection transition function rule $\gamma(F, a) = L$ as $a[L] = 1$ (no leader state already exists in the population). Henceforth, no more state L appears anymore as $a[L] = 0$, and consequently $\gamma(F, a) = L$ is not applicable

anymore. This unique agent with leader state will reduce every agent with state F, it interacts with, to an agent with state F_{halt} until $a[F]$ becomes 1. Thus, the only effective remaining interaction is $\gamma(L, a) = L_{halt}$ that changes the state of the agent with state L to L_{halt}. As a result, the agent with state L_{halt} is the unique leader of the population and all the remaining agents (except the absence detector) are followers.

According to this description, the agent that reaches the state L_{halt} can detect the global termination of this protocol which corresponds to the fact that all the agents have halted and a unique leader is designated. This was not possible with the basic Leader Election population protocol, but to achieve this the global knowledge was needed.

The Halting Leader Election Population Protocol with Absence Detector $\mathbf{LE_{AD}}$ under the Random Scheduler

We investigate now if including this absence detector in the population has any impact on the stabilization (or halting in this case) time of this protocol while running under the Random Scheduler. We have as a result the following theorem.

Theorem 6.1. *Let T_{LE_AD} be the time needed by the LE_{AD}, that runs over a complete graph under the Random Scheduler, to stabilize. Let $\mathbb{E}(T_{LE\_AD})$ be its expected value. Then, we have:*

$$\mathbb{E}(T_{LE\_AD}) \sim O(n^2 \log(n)), \text{ as } n \to \infty.$$

Proof. Initially, the population if formed by $(n + 1)$ agents: n agents with state F, and the absence detector.

Let X_t denote the set of agents with state L at time t, for any $t \geq 0$. Let x_t be the random variable that represents the size of X_t. Initially, $x_0 = 0$ and the absorbing state is $x'_t = 1$, as according to the transition function δ of this LE_{AD} protocol, only one agent from the initial agents with state F, is allowed to become L.

Then, the first interaction that takes place between an agent with state F and the absence detector happens according to the following probability:

$$\begin{aligned} \mathbb{P}r(x_{t+1} = 1 | x_t = 0) &= \frac{\binom{n}{1}}{\binom{\frac{n(n+1)}{2}}{1}} \\ &= \frac{2}{n+1}. \end{aligned} \tag{6.1}$$

Let T_{AF} denote the time needed for the first meeting between an agent F and the absence detector to happen. Let $\mathbb{E}(T_{AF})$ be its expected value. Then, with respect to (6.1), we get:

$$\begin{aligned} \mathbb{E}(T_{AF}) &= \frac{1}{\mathbb{P}r(x_1 = 1 | x_0 = 0)} \\ &= \frac{n+1}{2}. \end{aligned} \tag{6.2}$$

From the moment when an agent with state L appears, it starts reducing every agent from the remaining $(n - 1)$ agents with state F to F_{halt}.

Thus, let Y_t designate the set of agents with state F_{halt} at time t, for any $t \geq 0$ and y_t be the random variable denoting its size. At $t = 0$, $y_0 = 0$. Then, y_t is incremented each time the agent with state L interacts with an agent with state F until reaching the absorbing state $(n-1)$.

Consequently, $\forall i$, such that $0 \leq i \leq n-2$, we have:

$$\begin{aligned} \mathbb{P}r(y_{t+1} = i+1 | y_t = i) &= \frac{\binom{n-i-1}{1}}{\binom{\frac{n(n+1)}{2}}{1}} \\ &= \frac{2(n-i-1)}{n(n+1)}. \end{aligned} \tag{6.3}$$

If T_{LF} is the time needed to reduce the $(n-1)$ agents with state F to state F_{halt}, and $\mathbb{E}(T_{LF})$ its expected value, then according to (6.3) we get:

$$\begin{aligned} \mathbb{E}(T_{LF}) &= \sum_{i=0}^{n-2} \frac{1}{\mathbb{P}r(y_{t+1} = i+1 | y_t = i)} \\ &= \sum_{i=0}^{n-2} \frac{n(n+1)}{2(n-i-1)} \\ &= \frac{n(n+1)}{2} \sum_{i=0}^{n-2} \frac{1}{n-i-1} \\ &= \frac{n(n+1)}{2} \sum_{i=0}^{n-2} \frac{1}{n-i-1} \\ &= \frac{n(n+1)}{2} \sum_{i'=1}^{n-1} \frac{1}{i'} \\ &\simeq \frac{n(n+1)}{2} \log(n-1), \text{ as } n \to \infty. \end{aligned} \tag{6.4}$$

Finally, the leader needs to interact again with the absence detector to update its state from L to L_{halt}. Let l_t be the r.v denoting the size of the set of vertices with agent L at time t. In this case, $l_0 = 1$ and its absorbing state is 0. Let T_{LA} be the time needed for this event to take place, and let $\mathbb{E}(T_{LA})$ be its expected value. Then,

$$\begin{aligned} \mathbb{E}(T_{LA}) &= \frac{1}{\mathbb{P}r(l_{t+1} = 0 | l_t = 1)} \\ &= \frac{n(n+1)}{2}. \end{aligned} \tag{6.5}$$

Now, let T_{LE_AD} be the time needed for the protocol LE_{AD} to stabilize. Then, its expected value $\mathbb{E}(T_{LE\_AD})$ is such that:

$$\begin{aligned} \mathbb{E}(T_{LE\_AD}) &= \mathbb{E}(T_{AF}) + \mathbb{E}(T_{LF}) + \mathbb{E}(T_{LA}) \\ &= O(n^2 \log(n)), \text{ as } n \to \infty. \end{aligned} \tag{6.6}$$

□

The LE_{AD} protocol, running under the Random Scheduler over the complete graph, needs $O(n^2 \log(n))$ computation steps to stabilize, while the basic Leader Election population protocol needs $O(n^2)$.

The Halting Leader Election Population Protocol with Absence Detector $\mathbf{LE_{AD}}$ under the HS Scheduler

We suppose now that the protocol LE_{AD} is running under the *HS Scheduler* in a population whose interaction graph is complete. A description of the expected value of its stabilization time is given in the following theorem:

Theorem 6.2. *Let T_{LE_AD} be the time needed by the LE_{AD}, that runs over a complete graph under the HS Scheduler, to stabilize. Let $\mathbb{E}(T_{LE\_AD})$ be its expected value. Then:*

$$\mathbb{E}(T_{LE\_AD}) \sim O(n^2 \log(n)), \text{ as } n \to \infty.$$

Proof. As already specified above, we suppose that the protocol LE_{AD} runs over a population formed by $(n+1)$ elements: n agents that are initially with state F, and the absence detector.

We start by focusing on the first effective step of this protocol which consists on the first meeting of an agent with state F with the absence detector and that will update the state of this agent to L. Let T_{AF} denote the time needed for this first meeting to happen and let $\mathbb{E}(T_{AF})$ be its expected value.

Let G_{AF} be the set of edges linking each agent of the population to the absence detector. G_{AF} forms a star where the center is the absence detector and we have:

$$\pi(G_{AF}) = \sum_{a=(u,v)\in G_{AF}} \mathbb{P}r(\mathcal{HS}(a)).$$

As the interaction graph is the complete graph K_{n+1}, and according to (4.11), we can write:

$$\begin{aligned} \pi(G_{AF}) &= \sum_{a=(u,v)\in G_{AF}} \frac{\beta}{n^2} \\ &= \frac{\beta}{n^2} \times n \\ &= \frac{\beta}{n} \leq 1\,. \end{aligned} \tag{6.7}$$

Let $\xi_{G_{AF}}$ be the event of obtaining at least (and also at most in this case as we can not have more than one handshake over a star) one handshake in G_{AF}. According to Lemma 5.2, we can write: $\mathbb{P}r(\xi_{G_{AF}}) \geq \lambda\, min(1, \pi(G_{AF}))$ that, given (6.7), becomes:

$$\mathbb{P}r(\xi_{G_{AF}}) \geq \lambda\, \pi(G_{AF})\,. \tag{6.8}$$

On the other hand we have,

$$\mathbb{E}(T_{AF}) = \frac{1}{\mathbb{P}r(\xi_{G_{AF}})}\,. \tag{6.9}$$

This yields:

$$\begin{aligned} \mathbb{E}(T_{AF}) &\leq \frac{1}{\lambda\, \pi(G_{AF})} \\ &\leq \frac{n}{\lambda\, \beta}\,. \end{aligned} \tag{6.10}$$

Now, once the state L appeared in the population, the agent having this state will try to change each agent with state F to F_{halt}. Let G_{LF} be the set of edges linking the agent L to each of the k remaining agents with state F. Initially $k = n - 1$ and should finally reach 0. G_{LF} forms the star of size $k + 1$ where the center is the agent with state L.

$$\begin{aligned}
\pi(G_{LF}) &= \sum_{a=(u,v)\in G_{LF}} \mathbb{P}r(\mathcal{HS}(a)) \\
&= \sum_{a=(u,v)\in G_{LF}} \frac{\beta}{n^2} \\
&= \frac{\beta}{n^2} \times |G_{LF}| \\
&= \frac{\beta\, k}{n^2} \leq 1, \quad \forall\, 0 \leq k \leq n-1\,. \qquad (6.11)
\end{aligned}$$

Let $\xi_{G_{LF}}$ be the event of obtaining at least one handshake in G_{LF}. According to Lemma 5.2, and given (6.11), we have:

$$\mathbb{P}r(\xi_{G_{LF}}) \geq \lambda\, \pi(G_{LF})\,. \qquad (6.12)$$

Let T_{LF} be the time needed to reach a configuration where $k = 0$, and thus all the agents that were F have become F_{halt}. Let $\mathbb{E}(T_{LF})$ be the expected value of T_{LF}. Then, we have:

$$\mathbb{E}(T_{LF}) = \sum_{k=1}^{n-1} \frac{1}{\mathbb{P}r(\xi_{G_{LF}})}\,.$$

Given (6.12), this becomes:

$$\begin{aligned}
\mathbb{E}(T_{LF}) &\leq \sum_{k=1}^{n-1} \frac{1}{\lambda\, \pi(G_{LF})} \\
&\leq \sum_{k=1}^{n-1} \frac{n^2}{\lambda\, \beta\, k} \\
&\leq \frac{n^2}{\lambda\, \beta} \sum_{k=1}^{n-1} \frac{1}{k} \\
&\leq \frac{n^2}{\lambda\, \beta}\, H_{n-1}\,. \qquad (6.13)
\end{aligned}$$

Now, let T_{LA} be the time needed for the leader L to interact with the absence detector. Let $\mathbb{E}(T_{LA})$ be the expected value of T_{LA}. Let a be the edge linking the leader with the absence detector, then:

$$\begin{aligned}
\mathbb{E}(T_{LA}) &= \frac{1}{\mathbb{P}r(\mathcal{HS}(a))} \\
&= \frac{n^2}{\beta}\,. \qquad (6.14)
\end{aligned}$$

The expected value of the time needed for the LE_{AD} to halt is the sum of the three expected values that we just estimated. We can thus conclude that: the expected value of the stabilization time of this protocol is $O(n^2 \log(n))$. □

According to Theorem 6.1 and Theorem 6.2, the protocol LE_{AD} has the same time complexity when running over a complete graph under the Random Scheduler or under the HS Scheduler. Consequently, we can deduce the following corollary:

Corollary 6.3. *The Random Scheduler and the Handshake Scheduler are time equivalent with respect to the LE_{AD} protocol running over a population $\mathcal{P}$ with a complete interaction graph.*

6.2 Population Protocols with Some Local Termination Detection

According to Michail et al., introducing termination to population protocols has needed an oracle-node that has a global knowledge about the states of the agents forming the population, and which is connected to each of them. However, population protocols are distributed protocols designed to run over distributed systems and supposing that the agents have access to some global knowledge is a bit too strong as hypothesis. Thus, a question arises of whether it is possible to introduce termination to population protocols without any need to a global knowledge.

In the sequel, we propose the population protocols model with some local termination detection that extends a specific subclass of protocols from the original model to allow the detection of the local termination of some agents according to their states.

We would mention that attributing some local termination detection is applicable for any model extending the population protocols.

6.2.1 The Model of Population Protocols with Some Local Termination Detection

When reviewing the already proposed population protocols in literature (and also those extending them), we can notice that there is a family of protocols that have some states which once reached by an agent are never left. Furthermore, these states never appear as a member of an effective interaction. In other words, an agent that reaches such a state stops contributing to the protocol's progress. We denote these states **final** states, which correspond to halting states in the context of population protocols with absence detector.

Definition 6.2. (**Final State**) A final state is a state that never appears in the left hand side of an effective transition rule of a population protocol. It is also an irreversible state: once it is reached by an agent, this latter will never be updated to a new one.

A final state can be described more formally as follows:

Definition 6.3. (**Final States of a Protocol**) Let $\mathcal{A}$=(X, Y, Q, I, O, δ) be a population protocol. Q_{final} is the finite set of final states of the protocol $\mathcal{A}$ such that:

- $Q_{final} \subset Q$, and
- $\forall\, q \in Q, \forall\, q_{final} \in Q_{final}$, $\delta(q_{final}, q) = (q_{final}, q)$ and $\delta(q, q_{final}) = (q, q_{final})$.

Hence, regardless of the state of the agent with which an agent with a final state may interact, this interaction can not lead to any progress in the computation of the protocol. We therefore propose to introduce **local termination** of such agents. When an agent reaches a final state, it implies that it has locally terminated and halts without stopping interacting with the rest of the population.

From Population Protocols to Population Protocols with some Local Termination Detection

We are going to focus now on the family of protocols where $Q_{final} \neq \emptyset$ and rethink them in such a way that they can deal with the local termination of agents with final states. We thus extend this family of basic population protocols with a **local termination detection function** that enables an agent to check if its state is final.

We call this new extension of this subclass of population protocols the **population protocols with some local termination detection**. We should mention that all the concepts and definitions that are going to be presented hold for the population protocols model as well as any model extending it (among those that we presented in Chapter 2).

Let then $\mathcal{A} = (X, Y, Q, I, O, \delta)$ be a population protocol with Q including a non empty set of final states Q_{final}. Then, expressing $\mathcal{A}$ as a population protocol with some local termination detection gives $\mathcal{A}'=(X, Y, Q_{final}, Q, I, O, \delta, \zeta)$ where X, Y, Q_{final}, Q are finite sets and:

- X, Y, Q, I, O, and δ are the same elements than those designed for $\mathcal{A}$,
- Q_{final} is a subset of the set of states Q that consists of the set of the final states of the protocol $\mathcal{A}$, and
- $\zeta : Q \to \{0, 1\}$ is a local termination detection function described as follows:
  - $\zeta(q) = 1$, if $q \in Q_{final}$, and
  - $\zeta(q) = 0$ otherwise.

  Thanks to this function, an agent can detect that it has locally terminated.

Accordingly, unlike the population protocols with absence detector, an agent in a population executing a population protocol with some local termination detection can halt based, not on some global knowledge, but only on its local state. However, even if an agent detects that it has locally terminated, it continues interacting with the rest of the population.

6.2.2 Examples of Population Protocols with Some Local Termination Detection

In this section, we propose three examples of protocols that we converted to (mediated) population protocols with some local termination detection: the Leader Election population protocol (see Section 2.2.3), the VarProduct mediated population protocol (see Section 2.5) and the Maximal Matching mediated population protocol (see Section 5.4).

The Leader Election Population Protocol with Some Local Termination Detection

Let LE denote the basic Leader Election population protocol designed to run over complete graphs and already introduced in Chapter 2. It is a restriction of the protocol proposed by Angluin et al. [3] to only one effective transition rule which is:

$$(L, L) \rightarrow (L, F).$$

We can notice that, in this protocol, once an agent reaches the state F, it stops contributing in effective transitions and can not also update its state to a new one anymore. Consequently, F is a final state according to Definition 6.2 and Definition 6.3. Now, introducing local termination and the detection of this termination to this population protocol gives rise to a leader election population protocol with some local termination detection that we denote LE_T. This protocol consists of the following 8−tuple (X, Y, Q_{final}, Q, I, O, δ, ζ) with:

- $X = \{L\}$, $Y = \{L, F\}$,
- $Q_{final} = \{F\}$, $Q = \{L, F\}$,
- $\forall\ x \in X$, $I(x) = x$,
- $\forall\ y \in Y$, $O(y) = y$,
- $\delta : (L, L) \rightarrow (L, F)$, and
- $\forall q \in Q$, $\zeta(q) = 1$ if $q = F$, and $\zeta(q) = 0$ otherwise.

The VarProduct Mediated Population Protocol with Some Local Termination Detection

According to the description of the VarProduct mediated population protocol presented in Section 2.5, and to our definitions of finite states, we can conclude that the state $\bar{c}$ is a final state. Thus, converting VarProduct to a mediated population protocol with some local termination detection gives $VarProduct_T$ = (X, Y, Q_{final}, Q, S, I, O, ι, ω, δ, ζ) with:

- X, Y, Q, S, I, O, ι, ω, and δ are the same elements as those described for the VarProduct protocol,
- $Q_{final} = \{\bar{c}\}$, and
- $\forall q \in Q$, $\zeta(q) = 1$ if $q = \bar{c}$, and $\zeta(q) = 0$ otherwise.

The Maximal Matching Mediated Population Protocol with Some Local Termination Detection

The Maximal Matching, as it is described in Section 5.4, consists of only one effective transition rule which is:

$$(q_0, q_0, 0) \rightarrow (q_1, q_1, 1).$$

Thus, an agent reaching q_1 stops participating in effective interactions. Consequently, q_1 is a final state. Hence, a Maximal Matching mediated population protocol for which we introduce some local termination detection can be described by the protocol $MaxMatch_T = (X, Y, Q_{final}, Q, S, I, O, \iota, \omega, \delta, \zeta)$ with:

- X, Y, Q, S, I, O, ι, ω, and δ are the same elements as those described for the Maximal Matching protocol,
- $Q_{final} = \{q_1\}$, and
- $\forall q \in Q$, we have: $\zeta(q) = 1$ if $q = q_1$, and $\zeta(q) = 0$ if $q = q_0$.

Unlike the previous protocols, the $MaxMatch_T$ protocol (also the Maximal Matching protocol) can reach a configuration where all the agents are with the final state q_1. We say that this protocol is a locally terminating protocol.

Definition 6.4. (**Locally Terminating Population Protocol**) A locally terminating population protocol is a population protocol with some local detection of termination where all the agents eventually reach a final state.

Remark 6.1. Introducing local termination detection to any population protocol has no effect on the computation process of this latter (there are no added or omitted computation steps). Thus, the time needed for the protocol with some local termination detection to stabilize is the same as the one needed for the original protocol.

6.3 The Protocol Aware HS Scheduler

In both models, the population protocols with absence detector and the population protocols with some local detection of termination, a node halts when reaching a state from which is does not contribute to any effective interaction anymore, but continue interacting with the rest of the population. Any interaction that will involve the halting agent can not lead to any states update. Thus, if an agent with a final state stops interacting with the others, this has no consequence on the computation of the protocol. Therefore, we think about introducing effective termination of agents with final states. Stopping the interactions that involve agents with final states could be interesting as this could help an agent save its limited resources, and also may even lead to a faster stabilization of the protocol as non effective interactions will be avoided.

6.3.1 The Description of the Protocol Aware HS Scheduler

We propose to introduce this effective termination of the agents with final states via a protocol aware scheduler able to take into account the states of the agents. This protocol aware scheduler should consider that an agent with a final state is not an enabled agent anymore, which implies that this agent will never be picked among an ordered pair of agents chosen to communicate by this scheduler.

Therefore, we design a slightly modified version of the *HS Scheduler* that we call **Protocol Aware HS Scheduler**. This scheduler is also based on randomized handshakes while taking into account the states of the agents. Hence, no handshake will be possible over an edge where one extremity is an agent in a final state.

The Protocol Aware HS Scheduler can also be considered as a labeling of the edges of the interaction graph over which the protocol runs. The label of each edge is the probability that a handshake happens over it. However, unlike the case of the *HS Scheduler*, this labeling depends on the current states of the nodes, and thus on the current configuration of the population.

Accordingly, a formal definition that describes a scheduling of a population protocol with some local termination detection under the Protocol Aware HS Scheduler corresponds to:

Definition 6.5. (**Protocol Aware HS Scheduler**) Let $\mathcal{P}$ be a population with an interaction graph $G = (V, E)$. Let $\mathcal{A}$ be a population protocol with some local termination detection running over $\mathcal{P}$. A scheduling of $\mathcal{A}$ with the Protocol Aware HS Scheduler corresponds to the following mapping:

$$\begin{array}{rccc} HS\,Aware & : & (V(T), E) & \rightarrow & [0,1] \\ & & (C, e) & \mapsto & \mathbb{P}r\left(\mathcal{HS}_C(e)\right) \end{array}$$

with $V(T)$ is the set of vertices of $T(\mathcal{A}, G)$ which is the transition graph of the protocol $\mathcal{A}$ over the graph G, and $\mathbb{P}r\left(\mathcal{HS}_{\mathcal{C}}(e)\right)$ is the probability that, given the configuration C of the population $\mathcal{P}$, a handshake takes place over the edge e.

Compared to the protocol aware schedulers that are already proposed in the context of population protocols, this scheduler can simulate the Modified Scheduler (see Section 4.3.2) where $\varepsilon = 1$ but without being $diam(G)-$central. In fact, as it is based on handshakes, the Protocol Aware HS Scheduler is $1-$central.

The Protocol Aware HS Scheduler is suitable not only for the population protocols with some local termination detection but also for all the models extending them as long as they preserve the pairwise interactions.

6.3.2 The Protocol Aware HS Scheduler Procedure

As mentioned above, the Protocol Aware HS Scheduler is based on the randomized handshake model that should take into account the states of the agents. Some slight modifications are brought to the local randomized handshake procedure described in Algorithm 7 so that its global result can suit the description of the Protocol Aware HS Scheduler and simulate its functioning. The result is called the Protocol Aware HS Scheduler procedure defined by Algorithm 8.

The Protocol Aware HS Scheduler procedure is the Algorithm 7 where we insert, once the loop is initiated, a test step. If the agent's state is a final state, and consequently ζ returns 1, then the agent halts. Otherwise, it executes the computation steps from 2 to 18 of the Algorithm 7 that remain unchanged.

Traduced in the context of message passing communication model in wireless sensor networks, this gives: the agent that halts stop sending any "hello" messages so that its neighbors can not detect its presence anymore. And from its neighbors side, and as the agents are anonymous, no activity will be detected over the port via which the halting agent was supposed to send messages. Consequently, the neighbors of the halting node will not detect its presence but without realizing that this specific agent has halted or does not belong to their neighborhood anymore. Consequently, once a node halts, it simply

stops being able to initiate any handshake (or any interaction). Likewise, its neighbors can not detect it anymore as a neighbor being able to interact with, and thus do not initiate any handshake with it.

Therefore, the global result of the Protocol Aware HS Scheduler procedure guarantees that no handshake is possible over an edge where one extremity is a node with a final state.

Algorithm 8 The Protocol Aware HS Scheduler Procedure

```
1: loop
2:   if (ζ(current_state_v) = 1) then
3:     Halt;
4:     break;
5:   else
6:     Execute Algorithm 7 from 2 to 18;
7:   end if
8: end loop
```

6.3.3 Analysis of the Protocol Aware HS Scheduler Procedure

In this section, we aim to analyze the Protocol Aware HS Scheduler procedure that we proposed to better describe the processing of Protocol Aware HS Scheduler. Therefore, we start by defining, for each agent belonging to a population that runs a population protocol, and for each possible configuration C of the population, an attribute $d_C(v)$.

Definition 6.6. (**Node's Degree in a Configuration**) Let $\mathcal{P}$ be a population with an interaction graph $G = (V, E)$. Let $\mathcal{A}$ be a population protocol running over $\mathcal{P}$. Let $T = (V(T), E(T))$ be the transition graph of the protocol $\mathcal{A}$ over the graph G. For any $v \in V$, any $C \in V(T)$, $d_C(v)$ denotes the degree of the node v according to the configuration C.

This attribute is initialized as follows:

Definition 6.7. (**Initialization of Node's Degree**) Let $\mathcal{P}$ be a population with an interaction graph $G = (V, E)$. Let $\mathcal{A}$ be a population protocol running in $\mathcal{P}$. Let $T = (V(T), E(T))$ be the transition graph of the protocol $\mathcal{A}$ over the graph G. Let $C_0 \in V(T)$ denote the initial configuration. For any $v \in V$, $d_{C_0}(v) = d(v)$ where $d(v)$ is the degree of the node v in G.

Then, this attribute can vary as described in the following definition:

Definition 6.8. (**Node's Degree Variation**) Let $\mathcal{P}$ be a population with an interaction graph $G = (V, E)$. Let $\mathcal{A}$ be a population protocol running over $\mathcal{P}$. Let $T = (V(T), E(T))$ be the transition graph of the protocol $\mathcal{A}$ over the graph G. Let C_{i-1} and C_i be any two configurations from $V(T)$ such that $C_{i-1} \to C_i$. Then, for any $v \in V$,

$$d_{C_i}(v) = \begin{cases} 0, & \text{if } C_i(v) \in Q_{final} \text{ (v reaches a final state in } C_i) \\ d_{C_{i-1}}(v) - x, & \text{if } x \text{ neighbors of v reach final states in } C_i \\ d_{C_{i-1}}(v), & \text{otherwise.} \end{cases}$$

In words, $d_C(v)$ denotes the number of the enabled neighbors of the node v: those that are not with final states in C and with which v can potentially interact. $d_C(v) = 0$ implies that: either no neighbor of v is enabled during the configuration C, or the state of v in C is final. This corresponds to the fact that v stops communicating with any of its neighbors.

For any configuration C, any node v from the population, the attribute $d_C(v)$ satisfies the following lemma:

Lemma 6.4. *Let $\mathcal{P}$ be a population with an interaction graph $G = (V, E)$. Let $\mathcal{A}$ be a population protocol running over $\mathcal{P}$. Let $T = (V(T), E(T))$ be the transition graph of the protocol $\mathcal{A}$ over the graph G. For any $t \geq 1$, for any $C \in V(T)$, let C^t denote the t^{th} time that the configuration C is encountered. Then, for any $v \in V$,*

$$d_{C^t}(v) = d_{C^{t-1}}(v) = ... = d_{C^1}(v).$$

Proof. Recall that a configuration C is a snapshot of all the nodes states in the population. By definition, for any $t \geq 1$, and for any $v \in V$:

$$C^t(v) = C^{t-1}(v) = ... = C^1(v)$$

Also, for any two configurations C and C' from $V(T)$ such that $C \neq C'$, there exists at least one node $u \in V$ for which $C(u) \neq C'(u)$.

Let v be any node from the set V. Then, according to the Definition 6.8, $d_C(v)$ is determined by the state of the node v and also the states of its neighborhood. Suppose that, for $t \geq 1$, there exists $t' > t$, such that, $d_{C^t}(v) \neq d_{C^{t'}}(v)$. This implies that v or one of its neighbors reached a final state. Even if this final state was not reached in the t^{th} time where C is encountered but in another configuration from which C is reachable, and given that a final state is irreversible, this final state will still appear in $C^{t'}$. That is: there exists at least one node u from V such that $C^t(u) \neq C^{t'}(u)$, which contradicts the definition. Thus, the condition that: $\forall\, t \geq 1$, $\forall\, v \in V$, $C^t(v) = C^{t-1}(v) = ... = C^1(v)$, guarantees that each agent has the same state each time the configuration C is encountered. This implies that: $d_{C^t}(v) = d_{C^{t-1}}(v) = ... = d_{C^1}(v)$.

□

The Probability of a Handshake over an Edge with the Protocol Aware HS Scheduler

Let $\mathcal{A}$ denote a population protocol that runs over a population $\mathcal{P}$ with a communication graph $G = (V, E)$, and let $T = (V(T), E(T))$ be its transition graph over G. Each agent of $\mathcal{P}$ executes Algorithm 8. As we already mentioned: $\forall\, e = (u, v) \in E$, $\forall\, C \in V(T)$, $\mathbb{P}r\left(\mathcal{HS}_C(e)\right)$ denotes the probability that, given the configuration C, a handshake happens over the edge e. $\mathbb{P}r\left(\mathcal{HS}_C(e)\right)$ corresponds to the probability that the Protocol Aware HS Scheduler picks the edge e and consequently allows the pair of agents forming e to communicate. This probability is described by the following lemma:

Lemma 6.5. *Let $\mathcal{A}$ be a population protocol running over an interaction graph $G = (V, E)$. Let $T = (V(T), E(T))$ be the transition graph of the protocol $\mathcal{A}$ over the graph G. Then, $\forall\, e = (u, v) \in E$, $\forall\, C \in V(T)$, we have:*

$$\mathbb{P}r\left(\mathcal{HS}_C(e)\right) \;=\; \frac{\beta}{d_C(u)\, d_C(v)}\,.$$

Proof. The proof is based on a same reasoning than the one of the proof of Lemma 4.2. We recall that a handshake between two agents means that they have mutually chosen each others to communicate, and they generated two different random values.

A node u can choose a node v iff u detects v as one of its neighbors (that is: v is still an enabled neighbor of u). Accordingly, given a configuration C from $V(T)$, the probability that u chooses v is:

$$\mathbb{P}r(c(u) = v) = \frac{1}{d_C(u)} \,. \tag{6.15}$$

Let $\mathbb{P}r(r_u \neq r_v)$ denote the probability that u and v generate two different random numbers. According to (4.2), we have: $\mathbb{P}r(r_u \neq r_v) = 1 - \frac{1}{N} = \beta$.

According to these two probabilities, if $e = (u, v)$, we obtain:

$$\mathbb{P}r\left(\mathcal{HS}_C(e)\right) \;=\; \frac{\beta}{d_C(u)\, d_C(v)} \,.$$

□

As a consequence of this Lemma, we have: a scheduling of a population protocol (or any of its extensions) that has an empty set of final states, with the Protocol Aware HS Scheduler, corresponds simply to a scheduling with the *HS Scheduler*. In fact, for any possible configuration C of the population, as there are no final states, nothing will cause the modification of the attribute d_C of each agent of the population. Thus, for any agent u, $d_C(u)$ will be equal to $d(u)$. Hence, for any possible configuration C of the population and $\forall e = (u, v) \in E$, we have: $\mathbb{P}r\left(\mathcal{HS}_C(e)\right) = \mathbb{P}r\left(\mathcal{HS}(e)\right)$.

Remark 6.2. We recall that in the proofs of Lemma 5.1, Lemma 5.2 and Corollary 5.3, proposed in [32], the value of the probability of a handshake was not involved. Thus, in spite of the difference between the handshake model used there (in [32]) and the one we just proposed, these two lemmas and this corollary remain valid for our model of handshake too.

The Probability of a Least one Handshake over a Graph with the Protocol Aware HS Scheduler

Now, we focus on the probability that the Protocol Aware HS Scheduler picks at least one pair of agents to communicate, which corresponds to the probability of having at least one handshake over the interaction graph. This probability is as specified in the following lemma:

Lemma 6.6. *Let $\mathcal{A}$ be a population protocol with some local termination detection running on a communication graph $G = (V, E)$ under the Protocol Aware HS Scheduler. Then, the probability that this Scheduler picks at least one ordered pair of nodes at the end of Algorithm 8 is lower bounded by $1 - e^{\frac{-\beta}{2}}$.*

Proof. In the sequel, we denote $\{e_1, \ldots, e_m\}$ the set of edges in the interaction graph over which the protocol $\mathcal{A}$ runs. Let C be any possible configuration of the population where $\mathcal{A}$ is executed. We also denote by $\mathcal{HS}_C(G)$ the event: *Given the configuration C, there is at least a handshake in the graph G.* The $\overline{\mathcal{HS}}_C(G)$ and $\overline{\mathcal{HS}_C}(e)$ are respectively the complement events of $\mathcal{HS}_C(G)$ and $\mathcal{HS}_C(e)$.

We have:

$$\mathbb{P}r\left(\overline{\mathcal{HS}}_C(G)\right) \;=\; \mathbb{P}r\left(\wedge_{i=1}^{m}\overline{\mathcal{HS}}_C(e_i)\right)\,.$$

A similar computation to the one established in the proof of Lemma 4.3 gives:

$$\mathbb{P}r\left(\overline{\mathcal{HS}}_C(G)\right) \;\leq\; \left(1-\frac{\sum_{i=1}^{m}\mathbb{P}r\left(\mathcal{HS}_C(e_i)\right)}{m}\right)^m\,. \tag{6.16}$$

On the other hand, we have:

$$\sum_{i=1}^{m}\mathbb{P}r\left(\mathcal{HS}_C(e_i)\right) \;=\; \sum_{\{u,v\}\in E}\frac{\beta}{d_C(u)d_C(v)}\,.$$

Now, with respect to Definition 6.7 and Definition 6.8, for any possible configuration C, we have:

$$\forall\, u\in V,\; d_C(u)\leq d(u)\,.$$

This yields:

$$\sum_{i=1}^{m}\mathbb{P}r\left(\mathcal{HS}_C(e_i)\right) \;\geq\; \sum_{\{u,v\}\in E}\frac{\beta}{d(u)d(v)}\,.$$

Yet, given (4.7) established in the proof of Lemma 4.3, we can write:

$$\sum_{i=1}^{m}\mathbb{P}r\left(\mathcal{HS}_C(e_i)\right) \;\geq\; \frac{\beta}{2}\,.$$

Thus, (6.16) becomes:

$$\mathbb{P}r\left(\overline{\mathcal{HS}}_C(G)\right) \;\leq\; \left(1-\frac{\beta}{2m}\right)^m \sim e^{-\frac{\beta}{2}}\,. \tag{6.17}$$

Consequently, we obtain:

$$\mathbb{P}r\left(\mathcal{HS}_C(G)\right)\geq 1-e^{\frac{-\beta}{2}}\,. \tag{6.18}$$

□

The Number of Simultaneous Handshakes with the Protocol Aware HS Scheduler

Let $G=(V,E)$ be the interaction graph of a protocol $\mathcal{A}$ running under the Protocol Aware HS Scheduler. Let $T(\mathcal{A},G)=(V(T),E(T))$ be the transition graph of $\mathcal{A}$ over G and C represent any configuration from $V(T)$. Given C, we denote X the random variable $(r.v)$ which counts the number of simultaneous handshakes that can take place at the same step. A similar computation to the one performed in 4.10, using the linearity of the expectation, we can obtain the expected number of X:

$$\mathbb{E}\left(X\right)=\sum_{(u,v)\in E}\left(\frac{\beta}{d_C(u)d_C(v)}\right)\,. \tag{6.19}$$

6.3.4 The Fairness of the Protocol Aware HS Scheduler

The Protocol Aware HS Scheduler, as it was introduced above, should satisfy the fairness condition proposed by Chatzigiannakis et al. in [25].

Theorem 6.7. *The Protocol Aware HS Scheduler is a probabilistic consistent fair scheduler with probability 1.*

Proof. First, we start by proving that the Protocol Aware HS Scheduler is probabilistic. Let $\mathcal{A}$ be any population protocol with an interaction graph $G = (V, E)$. Let $T(\mathcal{A}, G) = (V(T), E(T))$ be any transition graph and C_i be any configuration in $V(T)$. Let C_j be any configuration in $V(T)$ reachable in one step from C_i. We define $Enc_{C_iC_j} = \{enc \mid enc \subset E(G)$ and $C_i \overset{enc}{\rightarrow} C_j\}$.

Now, for each element enc from $Enc_{C_iC_j}$, we rewrite the set E of the edges of the communication graph G as the union of three disjoint subsets:

$$E = enc \uplus F_1 \uplus F_2 \,,$$

with:

- $F_1 = \{f \in E \mid$ if $f = (u, v)$ then $\exists e \in enc$ such that $e = (u, v')$ or $e = (u', v)\}$, and
- $F_2 = \{f \in E \mid$ if $f = (u, v)$ then $\forall e \in enc$, if $e = (u', v')$ then $u \neq u'$, $u \neq v'$, $v \neq u'$ and $v \neq v'\}$.

Any time C_i is encountered, C_j is selected with the following probability:

$$\mathbb{P}r_{C_iC_j} = \sum_{enc \subset Enc_{C_iC_j}} \mathbb{P}r\left(\mathcal{HS}_{C_i}(enc)\right) \mathbb{P}r\left(\overline{\mathcal{HS}}_{C_i}(F_2)\right) \,.$$

Hence, the scheduler can define a probability distribution to every configuration $C \in V(T)$. It is then a probabilistic scheduler.

The probability $\mathbb{P}r_{C_iC_j}$ depends on the probabilities of the handshakes, given C_i, over enc and F_2, which in turn depend on the probability $\mathbb{P}r\left(\mathcal{HS}_{C_i}(e)\right)$ where $e \in (enc \cup F_2)$.

According to Lemma 6.4 and Lemma 6.5, this probability does not depend on the number of times the configuration C_i is encountered. As a consequence, $\mathbb{P}r_{C_iC_j}$ also does not depend on the number of times the configuration C_i is encountered. Therefore, we can state that the Protocol Aware HS Scheduler is consistent.

Also, according to Definition 4.15 of a one step computation, $C_i \rightarrow C_j$ implies that there exists $enc \subset E$ such that $C_i \xrightarrow{enc} C_j$ which means that $\mathbb{P}r_{C_iC_j} > 0$.

Thus, with respect to Theorem 4.1, the Protocol Aware HS Scheduler is a probabilistic consistent fair scheduler with a probability 1.

□

6.4 The Time Complexity of Leader Election Protocols under the Protocol Aware HS Scheduler

6.4.1 The Time Complexity of the Halting Leader Election under the Protocol Aware HS Scheduler

Theorem 6.8. *The expected value of the stabilization time of the protocol LE_{AD} running over a complete graph under the Protocol Aware HS Scheduler is $O(n^2)$, as $n \to \infty$.*

Proof. Let $\mathcal{P}$ be a population of n agents and an absence detector. Let C_0 denote the initial configuration of $\mathcal{P}$ executing the LE_{AD} protocol over a complete interaction graph $G = (V, E)$, under the Protocol Aware HS Scheduler. Let T_{AF} denote the time needed for the first agent with state L to appear (which is the time needed for the first meeting between an agent with state F and the absence detector to take place). And, let $\mathbb{E}(T_{AF})$ be its expected value.

Let G_{AF} be the set of edges linking each agents of the population to the absence detector.

$$\begin{aligned} \pi(G_{AF}) &= \sum_{a=(u,v)\in G_{AF}} \mathbb{P}r(\mathcal{HS}_{C_0}(a)) \\ &= \sum_{a=(u,v)\in G_{AF}} \frac{\beta}{d_{C_0}(u)\, d_{C_0}(v)}\,. \end{aligned}$$

With respect to Definition 6.7, we have: $\forall u \in V$, $d_{C_0}(u)$ is initialized with the degree of u in G. Thus, we can rewrite (6.20) such as:

$$\begin{aligned} \pi(G_{AF}) &= \sum_{a=(u,v)\in G_{AF}} \frac{\beta}{d(u)\, d(v)} \\ &= \sum_{a=(u,v)\in G_{AF}} \frac{\beta}{n^2} \\ &= \frac{\beta}{n^2} \times n \\ &= \frac{\beta}{n} \leq 1\,. \end{aligned} \tag{6.20}$$

This takes us back to the same case already studied in the proof of Theorem 6.2. Thus, we can conclude that:

$$\mathbb{E}(T_{AF}) \leq \frac{n}{\lambda\,\beta}\,. \tag{6.21}$$

As an agent with state L appeared in the population, this latter will reduce every agent with state F it meets, to F_{halt}. Let G_{LF} be the set of edges linking the agent L to each of the k remaining agents with state F. Initially $k = n - 1$ and should finally reach 0. Now, $\forall\, 0 \leq k \leq n-1$, we denote C'_k any configuration where there are k agents with state F.

$$\begin{aligned} \pi(G_{LF}) &= \sum_{a=(u,v)\in G_{LF}} \mathbb{P}r(\mathcal{HS}_{C'_k}(a)) \\ &= \sum_{a=(u,v)\in G_{LF}} \frac{\beta}{d_{C'_k}(u)\, d_{C'_k}(v)}\,. \end{aligned} \tag{6.22}$$

In a configuration C'_k, there are k agents with state F, one agent with state L, the rest of the population, except the absence detector, are with the final state F_{halt}. Thus, in this configuration, and as the interaction graph is complete, the attribute $d_{C'_k}$ of each vertex from those forming the edges of G_{LF} is equal to $k+1$. This yields:

$$\begin{aligned} \pi(G_{LF}) &= \frac{\beta}{(k+1)^2} \times |G_{LF}| \\ &= \frac{\beta\, k}{(k+1)^2} \leq 1, \quad \forall\, 0 \leq k \leq n-1\,. \end{aligned} \tag{6.23}$$

Let $\xi_{G_{LF}}$ be the event of obtaining at least one handshake in G_{LF}. According to Lemma 5.2, and given (6.23), we have:

$$\mathbb{P}r(\xi_{G_{LF}}) \geq \lambda\, \pi(G_{LF})\,. \tag{6.24}$$

T_{LF} denotes the time needed for reducing all the agent with state F to F_{halt} by the agent L, and $\mathbb{E}(T_{LF})$ denotes its expected value. We have:

$$\mathbb{E}(T_{LF}) = \sum_{k=1}^{n-1} \frac{1}{\mathbb{P}r(\xi_{G_{LF}})}\,.$$

Given 6.12, this becomes:

$$\begin{aligned} \mathbb{E}(T_{LF}) &\leq \sum_{k=1}^{n-1} \frac{1}{\lambda\, \pi(G_{LF})} \\ &\leq \sum_{k=1}^{n-1} \frac{(k+1)^2}{\lambda\, \beta\, k} \\ &\leq \frac{1}{\lambda\, \beta} \sum_{k=1}^{n-1} (k+2+\frac{1}{k}) \\ &\leq \frac{1}{\lambda\, \beta} \left(\frac{n(n-1)}{2} + 2(n-1) + H_{n-1} \right)\,. \end{aligned}$$

The configuration C'_0, as described above, denotes the one where there are $(n-1)$ agents with state F_{halt} and one agent L (C'_0 is not the initial configuration C_0). Now, given this configuration, the last step needed for this protocol to stabilize (or halt) is an interaction between the agent with state L and the absence detector. Let T_{LA} the time needed for this step to take place and let $\mathbb{E}(T_{LA})$ be its expected value. Let a be the edge linking the leader with the absence detector.

$$\mathbb{E}(T_{LA}) = \frac{1}{\mathbb{P}r(\mathcal{HS}_{C'_0}(a))} \tag{6.25}$$

As the agent with state L and the absence detector are the only remaining agents with non final states, they both have attributes $d_{C'_0}$ equal to 1. Consequently, (6.25) becomes:

$$\mathbb{E}(T_{LA}) = \frac{1}{\beta}\,. \tag{6.26}$$

Accordingly, the complexity of the stabilization time of the protocol LE_{AD} is, as $n \to \infty$, $O(n^2)$. □

According to this theorem, and to Theorem 6.8 and Corollary 6.3, we can conclude that the protocol LE_{AD}, running over a complete graph, stabilizes faster when it is under the Protocol Aware HS Scheduler, compared to the Random Scheduler and HS Scheduler.

6.4.2 The Time Complexity of the Leader Election with Some Local Termination Detection under the Protocol Aware HS Scheduler

Theorem 6.9. *The complexity of the stabilization time of the Leader Election protocol with some local termination detection, running over a complete graph, under the Protocol Aware HS Scheduler is:* $O(n)$, *as* $n \to \infty$.

Proof. Let $\mathcal{P}$ be the population of n agents that executes the Leader Election with some local termination detection as described in Section 6.2.2 over a complete graph. We denote C_k any configuration of $\mathcal{P}$ where there are k agents with state L, with $1 \leq k \leq n$. The initial configuration will be C_n as initially, all the agents start with state L. Given C_k, let G_k denotes the set of edges existing between the k agents with state L. G_k is a complete graph of k nodes. ξ_{G_k} is the event of obtaining at least one handshake in a round for at least one edge in G_k.

Let T_{LE} be the time needed for this protocol to stabilize and let $E(T_{LE})$ be its expected value. Then, if we suppose that, at each round, at most one L goes to the final state F, then we obtain:

$$E(T_{K_n}) \leq \sum_{k=2}^{n} \frac{1}{P(\xi_{C_k})}$$

On the one hand and according to Lemma 5.2, we have:

$$\mathbb{P}r(\xi_{G_k}) \;\geq\; \lambda \, min(1, \pi(G_k)) \,. \tag{6.27}$$

On the other hand, $\pi(G_k)$ is such that:

$$\begin{aligned} \pi(G_k) &= \sum_{a=(u,v)\,\in\, G_k} \mathbb{P}r(\mathcal{HS}_{C_k}(a)) \\ &= \sum_{a=(u,v)\,\in\, G_k} \frac{\beta}{d_{C_k}(u)\, d_{C_k}(v)} \\ &= \frac{\beta}{(k-1)^2} |G_k| \\ &= \frac{\beta\, k}{2(k-1)} \leq 1 \,. \end{aligned} \tag{6.28}$$

Then, (6.27) and (6.28) imply:

$$\begin{aligned} E(T_{K_n}) &\leq \sum_{k=2}^{n} \frac{1}{\lambda\, \pi(C_k)} \\ &\leq \frac{2}{\lambda\, \beta} \sum_{k=2}^{n} (1 - \frac{1}{k}) \\ &\leq \frac{2}{\lambda\, \beta} ((n-2) - (H_n - 1)) \\ &\leq \frac{2}{\lambda\, \beta} (n - H_n - 1) \end{aligned}$$

Accordingly, the complexity of the stabilization time of this Leader Election with some local termination detection running over a complete graph under the Protocol Aware HS Scheduler is $O(n)$, as $n \to \infty$. □

Accordingly, we can notice that introducing some local termination detection to the Leader Election protocol and its scheduling with the Protocol Aware HS Scheduler in a complete graph allows us to have an expected value of the stabilization time which is $O(n)$ compared to $O(n^2)$ in case of the Random Scheduler and *HS Scheduler*.

6.5 The Time Complexity of a Particular Case: the Iterated Population Protocols

In this section, we focus on a particular case of the protocols with some local termination detection running under the Protocol Aware HS Scheduler. We call these protocols: the **iterated protocols**. In an iterated protocol, once a node participates in a computation, it directly goes to a final state and halts. As an example, we can consider the mediated population protocol Maximal Matching with some local termination detection (see Section 6.2.2) as an iterated mediated population protocol since once a pair of agents is picked by the scheduler, both agents go to the final state q_1. A study of the time complexity of these iterated protocols is presented while considering different interaction graph structures.

6.5.1 A General Upper Bound

As a general upper bound for the time complexity of an *iterated protocol*, we have the following lemma:

Lemma 6.10. *Let $\mathcal{A}$ be an iterated protocol. Then, the expected time to compute $\mathcal{A}$ is upper bounded by $O(n)$.*

Proof. Let T be the r.v. that counts the number of rounds before the agents of the population computing the *iterated protocol* $\mathcal{A}$ halt. For any $t \geq 1$, let X_t denote the number of simultaneous computations in the graph G at round t, and let Y_t be the r.v. such that: $Y_t = n - 2X_t$. It is clear that, for any $t \geq 1$, $T \leq t$ if, and only if, $Y_t \leq 1$.

We define the following (pessimistic) process $(Y'_t)_{t\geq 0}$:

$$Y'_t = \begin{cases} n & \text{if } t = 0 \\ Y'_{t-1} - 2 & \text{if } X_t \geq 1 \\ Y'_{t-1} & \text{if } X_t = 0. \end{cases} \tag{6.29}$$

The process $(Y'_t)_{t\geq 0}$ is an irreversible ergodic Markov chain whose states are in the set $\{n, n-2, \cdots, n-2\lfloor\frac{n}{2}\rfloor\}$ and by Lemma 4.3, the transition probabilities are given by: $\mathbb{P}r\left(Y'_t = n-2\lfloor\frac{n}{2}\rfloor \mid Y'_{t-1} = n-2\lfloor\frac{n}{2}\rfloor\right) = 1$ and for any $i > 0$, and any $j \in \{n, \cdots, 0\}$,

$$\mathbb{P}r\left(Y'_t = i \mid Y'_{t-1} = j\right) = \begin{cases} e^{\frac{-\beta}{2}} & \text{if } i = j \\ 1 - e^{\frac{-\beta}{2}} & \text{if } i = j-2 \\ 0 & \text{otherwise .} \end{cases} \tag{6.30}$$

Let T' denote the time for the process $(Y'_t)_{t\geq 0}$, starting at $Y'_0 = n$ to reach the absorbing state $Y'_t \leq 1$. Then, the expected value of T is such that :

$$\begin{aligned} \mathbb{E}(T') &= \sum_{i=0}^{\lfloor\frac{n}{2}\rfloor-1} \frac{1}{\mathbb{P}r\left(Y'_t = n-2(i+1) \mid Y'_{t-1} = n-2i\right)} \\ &= \sum_{i=0}^{\lfloor\frac{n}{2}\rfloor-1} \frac{1}{1-e^{\frac{-\beta}{2}}} \\ &= \frac{\lfloor\frac{n}{2}\rfloor}{1-e^{\frac{-\beta}{2}}} . \end{aligned}$$

On the other hand, it is clear that $T < T'$ and hence $\mathbb{E}(T) \leq \mathbb{E}(T')$, which ends the proof. □

6.5.2 Upper Bound when the Interaction Graph is Complete

$\mathcal{A}$ represents any iterated protocol running over a population $\mathcal{P}$ of n agents with a complete interaction graph $G = (V, E) = K_n$. In this case, we can state the following lemma:

Lemma 6.11. *Let T_{K_n} denote the time complexity of an iterated protocol $\mathcal{A}$ running over an interaction graph a complete communication graph G. Then, the expected value of T_{K_n} satisfies:*

$$\mathbb{E}(T_{K_n}) \leq \frac{n - \log(\frac{n}{2})}{\lambda\ \beta}, \ \textit{as } n \to \infty.$$

Proof. Let G_k denote the graph formed by the k agents that are still with non final states in the population. Initially, $k = n$, then k decreases until reaching $n - 2\lfloor\frac{n}{2}\rfloor$. $\forall\, n - 2\lfloor\frac{n}{2}\rfloor \leq k \leq n$, G_k is a complete graph. ξ_{C_k} is the event of obtaining at least one handshake in a round for at least one edge in G_k. C_k represents any possible configuration of the population $\mathcal{P}$ where there are k remaining agents with non final states.

Now, if we consider the pessimistic process, where at each round, there is at most one pair of agents that go to final states, then:

$$E(T_{K_n}) \leq \sum_{k=0}^{\lfloor\frac{n}{2}\rfloor} \frac{1}{\mathbb{P}r(\xi_{G_{n-2k}})} \tag{6.31}$$

Now, to evaluate $\mathbb{P}r(\xi_{G_{n-2k}})$, we focus on:

$$\begin{aligned}
\pi(G_{n-2k}) &= \sum_{a \in G_{n-2k}} \mathbb{P}r(\mathcal{HS}(a)) \\
&= \sum_{(u,v)\in G_{n-2k}} \frac{\beta}{d_{C_{n-2k}}(u) d_{C_{n-2k}}(v)} \\
&= \sum_{(u,v)\in G_{n-2k}} \frac{\beta}{(n-2k-1)^2} \\
&= \frac{\beta}{(n-2k-1)^2} \frac{(n-2k)(n-2k-1)}{2} \\
&= \frac{\beta(n-2k)}{2(n-2k-1)} \leq 1 \,.
\end{aligned}$$

Now, by Lemma 5.2, we obtain:

$$\begin{aligned}
E(T_{K_n}) &\leq \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \frac{1}{\lambda\, \pi(G_{n-2k})} \\
&\leq \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \frac{2(n-2k-1)}{\lambda\, \beta\, (n-2k)} \\
&\leq \frac{2}{\lambda\, \beta} \left(\left(\lfloor \frac{n}{2} \rfloor + 1 \right) - \sum_{k'=n \mod 2}^{\frac{n}{2}} \frac{1}{2k'} \right) \\
&\leq \frac{2}{\lambda\, \beta} \left(\lfloor \frac{n}{2} \rfloor + 1 - \frac{1}{2} H_{\frac{n}{2}} \right) \,.
\end{aligned}$$

□

We recall that the expected value of the stabilization time of the Maximal Matching protocol running in a complete graph is $O(n^2)$, either being scheduled by the Random Scheduler or the *HS Scheduler*. Now, according to this lemma, we can conclude that introducing some local termination to this protocol, and its computation under the Protocol Aware HS Scheduler allows us to have a lower upper bound of the stabilization time, which is $O(n)$.

6.5.3 Upper Bound when the Interaction Graph is with Bounded Degree

Let $\mathcal{A}$ be any iterated protocol running over a population $\mathcal{P}$ of n agents with $G = (V, E)$ its interaction graph. In this section, we consider interaction graphs with degrees bounded by Δ. Then, we have the following lemma:

Lemma 6.12. *Let T be the time complexity of an iterated protocol $\mathcal{A}$ running over an interaction graph G that has a degree bounded by Δ. The expected value of T satisfies:*

$$\mathbb{E}\,(T) \leq -\frac{\log(\frac{n\Delta}{2})}{\log\left(1 - \left(\frac{\beta}{\Delta^2}\right)\right)} \,.$$

Proof. Let t represent an iteration of the algorithm and we define the sequence of graphs $(G_t)_{(t\geq 0)}$ as follows: $G_0 = G$ and for all $t \geq 1$, G_{t+1} is the graph obtained by removing, from G_t, the pairs that execute a computation step and their incident edges. In other words, G_t is the graph formed by the agents of the population that are not with final states at iteration t.

We define the following random variables: for any $t \geq 0$, X_t denotes the number of edges of the graph G_t and Y_t denotes the number of edges removed from the graph G_t at the end of iteration t. Then, we have $X_{t+1} = X_t - Y_t$ and thus:

$$\begin{aligned} \mathbb{E}\left(X_{t+1} \mid G_t\right) &= \mathbb{E}\left(X_t \mid G_t\right) - \mathbb{E}\left(Y_t \mid G_t\right) \\ &= X_t - \mathbb{E}\left(Y_t \mid G_t\right) . \end{aligned} \tag{6.32}$$

On the other hand, for any pair $e = (u, v)$, if a computation is done by the pair (u, v) at iteration t, then the edge e is removed from the graph G_t. Hence, by (6.19),

$$\mathbb{E}\left(Y_t \mid G_t\right) \geq \sum_{(u,v)\in E(G_t)} \frac{\beta}{d_{C_t}(u) d_{C_t}(v)} , \tag{6.33}$$

where C_t stays for the configuration of the population at iteration t. Then:

$$\mathbb{E}\left(X_{t+1} \mid G_t\right) \leq X_t - \sum_{(u,v)\in E(G_t)} \frac{\beta}{d_{C_t}(u) d_{C_t}(v)} . \tag{6.34}$$

Since, for any C_t, for any u, $d_{C_t}(u) \leq d(u) \leq \Delta$, this becomes:

$$\mathbb{E}\left(X_{t+1} \mid G_t\right) \leq X_t \left(1 - \frac{\beta}{\Delta^2}\right) . \tag{6.35}$$

For $t \geq 0$, we define the r.v. $Z_t = \frac{X_t}{\left(1-\frac{\beta}{\Delta^2}\right)^t}$. Then, $\mathbb{E}\left(Z_{t+1} \mid G_t\right) \leq Z_t$. Thus, the r.v. Z_t is a super-martingale, and then:

$$\mathbb{E}\left(Z_{t+1}\right) = \mathbb{E}\left(\mathbb{E}\left(Z_{t+1} \mid G_t\right)\right) \leq \mathbb{E}\left(Z_t\right) . \tag{6.36}$$

A direct application of a theorem from [69] yields to $\mathbb{E}\left(Z_t\right) \leq Z_0 = m$. Thus:

$$\mathbb{E}\left(X_t\right) = \left(1 - \frac{\beta}{\Delta^2}\right)^t \mathbb{E}\left(Z_t\right) \leq m \left(1 - \frac{\beta}{\Delta^2}\right)^t . \tag{6.37}$$

We have $m \leq \frac{n\Delta}{2}$ and the algorithm halts when $X_t < 1$. This implies that t is upper bounded by $-\frac{\log\left(\frac{n\Delta}{2}\right)}{\log\left(1-\frac{\beta}{\Delta^2}\right)}$ which ends the proof. □

Corollary 6.13. *If Δ is a constant, then the expected time to compute the iterated protocol is $O\left(\log n\right)$.*

6.5.4 Upper Bound when the Interaction Graph is a Random Graph

In this section, we analyze the complexity of an *iterated protocol* in random graphs. We focus our study on the case where $G = G_{n,p}$ with $p = \frac{\alpha \log n}{n}$ (with $\alpha > 1$), that is on the random graphs which are connected with high probability. We have the following result:

Lemma 6.14. *Let $G_{n,p} = (V, E)$ be a random graph with $p = \frac{\alpha \log n}{n}$ (for $\alpha > 1$). Let T the time complexity of an iterated protocol $\mathcal{A}$ running over this $G_{n,p}$. The expected value of T satisfies:*

$$\mathbb{E}(T) \leq -\frac{\log n}{\log\left(1 - \frac{\beta}{\alpha \log n}\right)} .$$

Proof. The proof uses the same arguments as for the proof of Lemma 6.12. We define the sequence of graphs $(G_t)_{(t\geq 0)}$, and the two r.v. X_t as the size of G_t and Y_t as the number of edges removed at the end of the computation step t. Then we have $X_{t+1} = X_t - 2Y_t$, and:

$$\mathbb{E}(X_{t+1} \mid G_t) \quad = \quad X_t - 2\mathbb{E}(Y_t \mid G_t) .$$

Given (6.33), and as we have: for any C_t, for any u, $d_{C_t}(u) \leq d(u)$, we get:

$$\mathbb{E}(Y_t \mid G_t) \quad \geq \quad \sum_{(u,v)\in E(G_t)} \frac{\beta}{d(u)d(v)} . \tag{6.38}$$

Then, using (4.21), this becomes:

$$\mathbb{E}(Y_t \mid G_t) \quad \geq \quad \beta \frac{X_t}{2\alpha \log X_t} . \tag{6.39}$$

Consequently, we obtain:

$$\mathbb{E}(X_{t+1} \mid G_t) \quad \leq \quad X_t - \frac{\beta}{\alpha \log X_t} X_t .$$

Now, as $X_t \leq n$, $\forall t \geq 0$, we have:

$$\mathbb{E}(X_{t+1} \mid G_t) \quad \leq \quad \left(1 - \frac{\beta}{\alpha \log n}\right) X_t .$$

Then, the theorem holds by the same reasoning as for the proof of (6.36) and (6.37). □

We recall that the expected value of the stabilization time of the Maximal Matching protocol running in a $G_{n,p}$ graph is: $O(\frac{n^3}{\log n})$ under the Random Scheduler, and $O(n \log^2(n))$ under the *HS Scheduler*. Consequently, given Lemma 6.14, we can conclude that introducing some local termination to this protocol and using the Protocol Aware HS Scheduler, allows a faster stabilization of this protocol.

6.6 Conclusion

We presented in this chapter, the model of population protocol with absence detector proposed by Michail et al. that introduces the concept of termination in population protocol. We focused on the Leader Election protocol with absence detector. We studied the expected value of its stabilization time first when running under the Random Scheduler, then when running under the *HS Scheduler*. We concluded thanks to this study, that the Random Scheduler and the *HS Scheduler* are time equivalent with respect to this

protocol running in a complete interaction graph. They both have an expected value of the stabilization time which is $O(n^2 \log(n))$.

We then introduced the model of population protocol with some local termination detection. We designed for these protocols a protocol aware distributed scheduler: the Protocol Aware HS Scheduler. This scheduler is also based on randomized handshakes. It allows agents that reach a specific state, called final state, to stop participating in the pairwise interactions. We proved that this scheduler is fair with probability 1.

We took up the example of the Leader Election protocol with absence detector. This protocol running in a complete graph under the Protocol Aware HS Scheduler needs $O(n^2)$ computation steps to stabilize. It thus stabilizes faster compared to its computation under the Random Scheduler and the *HS Scheduler*.

We also studied the stabilization time of the Leader Election with some local termination detection and the iterated protocols that we proposed, when running under the Protocol Aware HS Scheduler.

We proved that introducing some local termination detection to the Leader Election protocol, and to the Maximal Matching protocol, and their computation under the Protocol Aware HS Scheduler gave lower upper bounds compared to their executions under the Random Scheduler and the *HS Scheduler*.

Table 6.1 summarizes the established results related to the stabilization time upper bounds.

	Graph Structure		**Scheduler**			**Time Complexity**
	K_n	$G_{n,p}$	Random	HS	P A HS	
Leader Election with absence detector	×		×			$O(n^2 \log(n))$
	×			×		$O(n^2 \log(n))$
	×				×	$O(n^2)$
Leader Election with some LTD	×				×	$O(n)$
MaxMatch with some LTD	×				×	$O(n)$
		×			×	$O(-\frac{\log n}{\log\left(1-\frac{\beta}{\alpha \log n}\right)})$

Table 6.1: Time complexity of protocols with Absence Detector and with some local termination detection

Chapter 7

Experimentation under ViSiDiA Platform

In this chapter, we introduce ViSiDiA: the Java framework designed for the visualization and simulation of distributed algorithms. Thanks to this framework, we are going to illustrate the possibility of the mappings that we presented in Chapter 3, between the population protocols and tasks with graph relabeling systems on the one hand, and between the population protocols and the anonymous asynchronous message passing model on the other hand. We thus represent the possibility of describing a (mediated) population protocol with each of these two models through the examples of the three protocols already studied in the previous chapters. Then, once the different studied schedulers (the Random Scheduler, the *HS Scheduler* and the Protocol Aware HS Scheduler) are implemented on ViSiDiA, we propose to validate the theoretical analyses established in the previous chapters with experiments under this platform.

Our publication in [1] can be a tool for the reader to get familiar with ViSiDiA platform.

7.1 ViSiDiA

ViSiDiA (Visualization and Simulation of Distributed Algorithms) is a Java framework that allows to implement, simulate and visualize distributed algorithms [1, 16]. It is based on the modeling of a network as a graph wherein each node represents a processor and the edges represent communication channels. ViSiDiA has a dual purpose: educational and research. Regarding the educational goals, ViSiDiA aims to facilitate the understanding of distributed algorithms. From the research point of view, the goal is to make easier the design, evaluation and analyses of new distributed algorithms.

7.1.1 Main Functionalities

The main functionalities of ViSiDiA, from a user's perspective, are depicted in Figure 7.1. In fact, a user can run an experiment, implement a new algorithm, or replay a simulation.

To run an experiment, the user should first define a network topology. He can either create it in ViSiDiA, or import it as a GML (Graph Modeling Language) file. Creating a network topology in ViSiDiA can be done manually, or automatically thanks to a new module that we added to this platform. This module offers the possibility of automatically generating three different types of graphs: Random Graphs (the $G_{n,p}$ model), Random Geometric Graphs (Unit Disk Graph model) and Small World Graphs. This makes the task of the user much easier especially when he has to generate graphs with huge sizes. No matter how the graph is generated, the status of the processes and edges are encoded by labels.

Thereafter, the user can select an algorithm among those provided in ViSiDiA's database which may be enhanced by user's defined algorithms. Once an algorithm is selected, the simulation can be carried out. The user can visualize the progress of the simulation through animations, but this is not mandatory.

Also, instead of running a new experiment, a user can replay a previously recorded simulation.

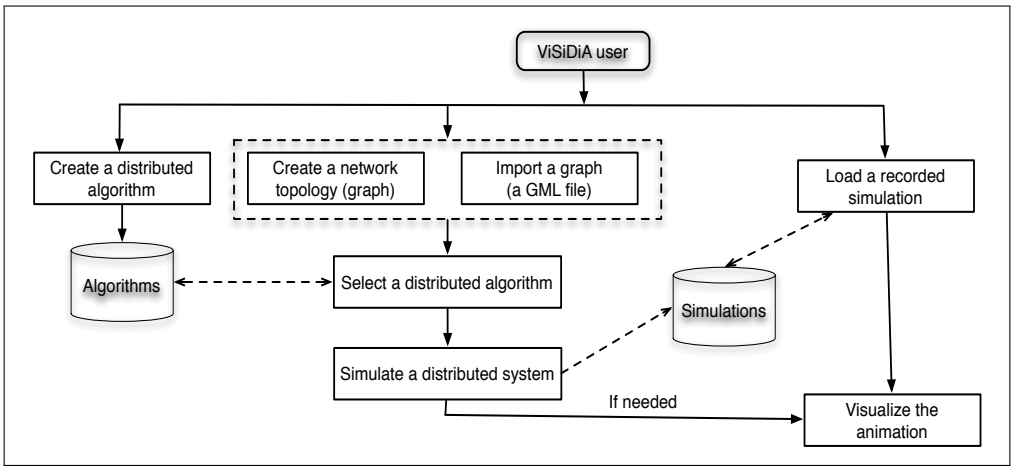

Figure 7.1: ViSiDiA's functionalities from users perspective

7.1.2 Architecture

ViSiDiA is implemented in Java and has been designed using a multi-layer architecture. This design choice allows to easily extend the functionalities of each layer independently of the other layers. For instance, this architecture facilitates the implementation of new algorithms. The flowchart of this multi-layer architecture is illustrated by Figure 7.2 (where plain arrows represent direct association, while the dashed arrow is an indirect association). This architecture relies on a Model-View-Controller (MVC) pattern where: the model is the graph representing a distributed system, the view is the Graphical User Interface (GUI), and the controller is the simulator module.

The GUI is the first layer. It enables to build the graph that represents the network topology, to visualize the progress of the simulation and to potentially influence this progress (speed up, slow down or pause the simulation).

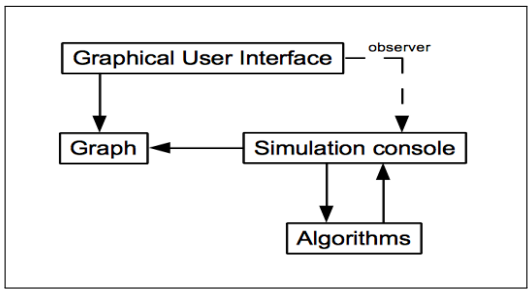

Figure 7.2: ViSiDiA's architecture

The second layer includes the graph description tool and the distributed algorithms simulator. The latter is the main module of ViSiDiA. It is worth noting that this module is independent from the GUI. Indeed, a simulation can run without any GUI. ViSiDiA uses an event-driven simulator. Any action leads to the generation of an event. These events are interpreted by the simulator which (eventually) modifies the graph. For example, the communication process by message passing can lead to several events:

- *event* 1: the source node issues a message.

- *event* 2: the GUI (if any) displays the message going from the source node to the destination(s).

- *event* 3: once the message reaches the destination, it is put in the receiver's queue.

- *event* 4: the receiver performs some actions according to the content of the message.

These events are transparently managed by the simulation module and algorithm developers do not have to deal with them.

The last layer contains an Application Programming Interface (API) that allows users to program their own distributed algorithms and run them on ViSiDiA.

Regarding this architectural pattern, we can draw several conclusions:

- the simulation console is independent from the GUI,

- the graph is only modified by GUI (through user interaction) and simulation console,

- the algorithms must refer to the simulation console to operate on the graph.

7.1.3 Distributed Communication Models

ViSiDiA implements various distributed communication models that are: message passing, mobile agents, and local computations with graph relabeling rules, using both synchronous and asynchronous systems. By asynchronous system, we mean there is no global clock shared by all nodes and messages can arrive at arbitrary times.

The algorithms database of ViSiDiA contains some well-known distributed algorithms such as: leader election, spanning tree computation, ring coloring, graph enumeration, etc, implemented with these different models of communication. Also, as already mentioned above, the API allows ViSiDiA users to implement their own algorithms. According to the distributed communication model on which the algorithm will be based, they can simply invoke the corresponding already defined functions.

Developing New Algorithms with Message Passing

We present the example of the implementation of a *broadcast algorithm* based on message passing. This algorithm is as follows: we suppose that a given node (say v), labeled "A" has an information to broadcast. v sends the message to all its neighbors. Upon receiving the message, each node changes its own label to "A" then forward this message to its respective neighbors (except the one which previously sent this message). We propose the asynchronous version of this broadcast algorithm (see Listing 7.1).

The `getDescription()` method is not mandatory. It is used to add a description of the algorithm that will be displayed in the GUI when the user will browse the list of the algorithm in order to select one of them (using a Java JFileChooser).

The `getMessageTypeList()` is used by ViSiDiA to get the list of messages that will be displayed on the GUI (the user can choose to display them or not). In this example, there is only one message type, called `wave`.

Listing 7.1: Asynchronous broadcast algorithm

```
1  import java.util.Collection;
2  import java.util.LinkedList;
3
4  import visidia.simulation.process.algorithm.Algorithm;
5  import visidia.simulation.process.edgestate.MarkedState;
6  import visidia.simulation.process.messages.Door;
7  import visidia.simulation.process.messages.Message;
8  import visidia.simulation.process.messages.MessageType;
9  import visidia.simulation.process.messages.StringMessage;
10
11 public class Broadcast extends Algorithm {
12
13   @Override
14   public String getDescription(){
15     return "This algorithm implements a broadcasting " +
16             "process initiated by a vertex labelled A.";
17   }
18
19   static MessageType wave = new MessageType("Wave", true);
20
21   @Override
22   public Collection<MessageType> getMessageTypeList() {
23     Collection<MessageType> typesList;
24     typesList = new LinkedList<MessageType>();
25     typesList.add(Broadcast.wave);
26     return typesList;
27   }
28
29   @Override
30   public void init() {
31     ...
32   }
33
34   @Override
35   public Object clone() {
36     return new Broadcast();
37   }
38 }
```

Listing 7.2: Asynchronous broadcast algorithm: the "init" method

```
1  public void init() {
2
3    String label = getProperty("label").toString();
4
5    if (label.equals("A")) {
6      sendAll(new StringMessage("Wave", wave));
7    } else {
8      Door door = new Door();
9      Message msg = receive(door);
10     setLocalProperty("label", new String("A"));
11
12     for (int i = 0; i < getArity(); i++) {
13       if (i != door.getNum())
14         sendTo(i, msg);
15     }
16   }
17 }
```

The core of the broadcast algorithm is described in the `init` method (Listing 7.2). The user does not have to handle the send and receive processes. He has only to invoke the corresponding function already defined in ViSiDiA, and that correspond in this example to: sendTo(), sendAll(), receive().

The instruction in line 3 allows to get the label of the node. On line 5, if the node's label equals to "A", this means the node is the initiator of the broadcast (decision taken

by the user). Therefore it sends to all its neighbors a message that contains the text " Wave" (line 6). If the label of the node is not "A", then the node waits for the first message coming on one of its doors (lines 8-9). Then the node changes its label (line 10) before forwarding the message it received (lines 12-15).

Developing New Algorithms with Graph Relabeling Systems

Now, if the user chooses to describe its distributed algorithm in ViSiDiA using the abstract model of graph relabeling systems, then ViSiDiA offers him an easier method to implement it without writing any code. In fact, he has just to draw the sequence of the corresponding relabeling rules that encode its algorithm. These relabeling rules can be described as $LC0$, $LC1$ or $LC2$ rules.

We consider the graph relabeling system $\mathcal{R}_{\mathcal{ST}}$, described in Section 3.1, that encodes a distributed algorithm computing a spanning tree of any graph. $\mathcal{R}_{\mathcal{ST}}$ consists of only one relabeling rule: R_{ST}. Thus, implementing the distributed algorithm encoded by $\mathcal{R}_{\mathcal{ST}}$ in ViSiDiA consists on drawing R_{ST} as it is shown in Figure 7.3.

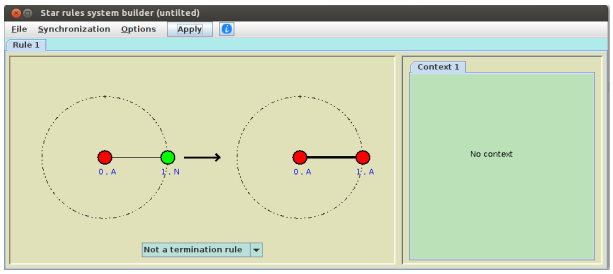

Figure 7.3: The spanning tree relabeling rule R_{ST} in ViSiDiA

Accordingly, in the left part of Figure 7.3, the relabeling rule that allows to relabel the graph is described. And, in the right part, some forbidden contexts can be added.

7.2 Population Protocols in ViSiDiA

Through Chapter 5 and Chapter 6, we established some theoretical results concerning the time complexity of some protocols running under different schedulers. We aim to validate these results through some experiments and simulations. We propose to use ViSiDiA as a simulation tool. We can notice that, according to the description that we provided in the previous section, ViSiDiA was not specially designed to handle the population protocol model and its extensions. However, we already established in Chapter 3 an approach that allows to describe the computation of a (mediated) population protocol as a realization of a task with graph relabeling system, or as a distributed algorithm in an anonymous asynchronous system based on message passing. We describe in this section how such mappings are possible with ViSiDiA.

7.2.1 From Population Protocols to Tasks with Graph Relabeling Systems in ViSiDiA

In this section, we illustrate the possibility of describing the computation a (mediated) population protocol as a realization of a task with a graph relabeling system in ViSiDiA. We show this through the following three examples: the OR population protocol, the Leader Election population protocol and the Maximal Matching mediated population protocol.

The OR Protocol

We recall the population protocol OR described in Section 4.6.3. We just update Q_{OR} such that $Q_{OR} = \{A, N\}$ with A standing for the state q_1 and N for the state q_0 (it is more adapted with the existing labels in ViSiDiA). Let T_{OR} be a task whose input and output labeling types are respectively: L_{OR_i} and L_{OR_o}. Let (G, σ_i) be a labeled graph in the domain of T_{OR}. And let $\mathcal{R}_{\mathcal{OR}} = \{\mathcal{L}_{\mathcal{OR}}, \mathcal{I}_{\mathcal{OR}}, \mathcal{P}_{\mathcal{OR}}\}$ be a graph relabeling system.

Now, with respect to the mapping approach that we established in Chapter 3, we have to define the parameters of T_{OR} and $\mathcal{R}_{\mathcal{OR}}$ such that the realization of T_{OR} on (G, σ_i) according to $\mathcal{R}_{\mathcal{OR}}$ corresponds to the computation of the OR protocol.

Consequently, these parameters should be as follows:

- $L_{OR_i} = L_{OR_o} = X_{OR} = \{0, 1\}$,
- $\mathcal{L}_{\mathcal{OR}} = Q_{OR} = \{A, N\}$,
- $\mathcal{I}_{\mathcal{OR}} = \mathcal{L}_{\mathcal{OR}}$,
- $\mathcal{P}_{\mathcal{OR}} = \phi(\delta_{OR}) = \{R_{OR_1}, R_{OR_2}\}$,
- the initialization function of this realization $\lambda_{OR} = I_{OR}$, and
- the output extraction function of this realization $\pi_{OR} = O_{OR}$.

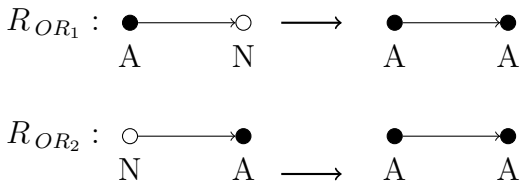

Figure 7.4: The relabeling rules of $\mathcal{R}_{\mathcal{OR}}$

Figure 7.4 describes the relabeling rules R_{OR_1} and R_{OR_2} that should be drawn in ViSiDiA.

The Leader Election Protocol

We consider now the population protocol Leader Election (see Section 2.2.3) where we just modify the notation of the states forming the set Q_{LE} such that $Q_{LE} = \{N, A\}$. The state N stands for the state L and A for F.

To map the computation of this protocol to a realization of task with a graph relabeling system, we define a task T_{LE} with an input labeling type $L_{LE_i} = X_{LE}$ and an output labeling type $L_{LE_o} = Y_{LE}$.

We also consider a graph relabeling system $\mathcal{R}_{\mathcal{LE}} = \{\mathcal{L}_{\mathcal{LE}}, \mathcal{I}_{\mathcal{LE}}, \mathcal{P}_{\mathcal{LE}}\}$ with: $\mathcal{L}_{\mathcal{LE}} = Q_{LE}$, $\mathcal{I}_{\mathcal{LE}} = \{N\}$, and $\mathcal{P}_{\mathcal{LE}} = \phi(\delta_{LE}) = \{R_{LE}\}$. The relabeling rule R_{LE} is depicted in Figure 7.5.

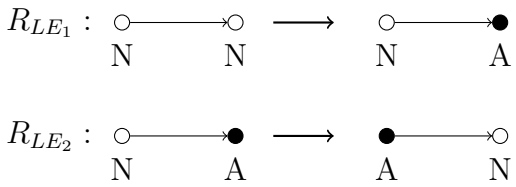

Figure 7.5: The relabeling rules of $\mathcal{R}_{\mathcal{LE}}$

For any labeled graph in the domain of T_{LE}, realizing this task according to $\mathcal{R}_{\mathcal{LE}}$ over this graph will have an initialization phase $\lambda_{LE} = I_{LE}$, and an output extraction function $\pi_{LE} = O_{LE}$.

We recall that in case of complete interaction graphs, the transition function of this protocol can restricted to only one transition rule which is $\delta_{LE}(N, N) = (N, A)$. Then $\mathcal{P}_{\mathcal{LE}}$ can also be restricted to the relabeling rule R_{LE_1} in this case.

The Maximal Matching Protocol

We consider the Maximal Matching mediated population protocol, introduced in Section 5.4 with a slight modification of the states denotation (to better suit the existing labels in ViSiDiA). Accordingly, a state A replaces the state q_1 and a state N replaces the state q_0. We focus now on describing the computation of this protocol as a realization of a task.

Let $T_{MaxMatch}$ be a task with an input labeling type $L_{M_i} = X_M = \{0\}$, and an output labeling type $L_{M_o} = Y_M = \{0, 1\}$. Let $\mathcal{R}_{\mathcal{M}} = \{\mathcal{L}_{\mathcal{M}}, \mathcal{I}_{\mathcal{M}}, \mathcal{P}_{\mathcal{M}}\}$ be a graph relabeling system with: $\mathcal{L}_{\mathcal{M}} = Q_M \cup S_M = \{A, N, 0, 1\}$, $\mathcal{I}_{\mathcal{M}} = \{N, 0\}$ and $\mathcal{P}_{\mathcal{M}} = \phi(\delta_M) = \{R_M\}$. The relabeling rule R_M is as shown in Figure 7.6.

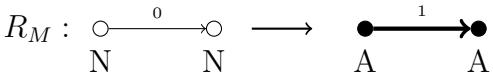

Figure 7.6: The relabeling rule R_M

Accordingly, the computation of the Maximal Matching protocol can be mapped to the realization of $T_{MaxMatch}$ according to $\mathcal{R}_{\mathcal{M}}$ over any labeled graph in the domain of this task and where the initialization function and the output extraction function are defined with respect to the specifications in Table 3.1.

Implementation and Simulation in ViSiDiA

The implementation of any of these protocols described above consists on generating a labeled graph in the domain of the corresponding task and drawing the relabeling rules of the corresponding graph relabeling system. We should mention that in ViSiDiA, when drawing the relabeling rules, we do not need to specify the direction of the edges to break the symmetry. In fact, if we focus on Figure 7.3, we can notice that a $LC0$ rule is described over an edge where one of its extremities is the center of the ball of radius 1. The center will thus be considered as being the initiator while the other node is the responder.

Before launching the simulation, the user should choose the scheduling scenario. This is possible through the specification of the corresponding synchronization when drawing the relabeling rules. Then, simulation with ViSiDiA can take place and consequently the number of rounds needed for this computation to stabilize can be obtained.

7.2.2 From Population Protocols to Anonymous Asynchronous Message Passing in ViSiDiA

We consider in this section the possibility of simulating a (mediated) population protocol in ViSiDiA by mapping it to an asynchronous algorithm running over an anonymous system based on message passing. This algorithm is a distributed one consisting of a collection of local algorithms that are a detailed implementation of the *Algo* algorithm (see Algorithm 1) where:

- The procedure $TrySynchronization()$ is replaced by the synchronization part of the algorithm related to the corresponding scheduler: either the Random Scheduler (Algorithm 4), the *HS Scheduler* (Algorithm 7) or the Protocol Aware HS Scheduler (Algorithm 8).

- Line 7 is replaced by an implementation of the application of any possible transition rule of the transition function of the corresponding protocol.

We continue with the same examples of the previous section by detailing, for each protocol, the part of the algorithm describing its transition function. We just give their descriptions as pseudo codes that are later implemented in Java under ViSiDiA. We should mention that we continue using, for each of these protocols, the same notation of the states already used in the previous section.

The OR Protocol

Describing the transition function δ_{OR} in the context of an algorithm based on message passing corresponds to Algorithm 9.

Algorithm 9 The Transition Function δ_{OR}

```
if ((state_u = N) and (state_c(u) = A)) then
    state_u ← A
end if
```

The Leader Election Protocol

We consider the following two cases:

- **The Leader Election Protocol in Complete Graphs** The transition function of this protocol is described in Algorithm 10.

Algorithm 10 The Transition Function δ_{LE} (Complete Graph)

if $(role = r)$ **then**
  if $((state_u = N)$ **and** $(state_{c(u)} = N))$ **then**
    $state_u \leftarrow A$
  end if
end if

- **The Leader Election Protocol in Random Graphs** Algorithm 11 corresponds to the transition function of this protocol designed for random interaction graphs.

Algorithm 11 The Transition Function δ_{LE} (Random Graph)

if $(role = i)$ **then**
  if $((state_u = N)$ **and** $(state_{c(u)} = A))$ **then**
    $state_u \leftarrow A$
  end if
else
  if $((state_u = N)$ **and** $(state_{c(u)} = N))$ **then**
    $state_u \leftarrow A$
  else
    if $((state_u = A)$ **and** $(state_{c(u)} = N))$ **then**
      $state_u \leftarrow N$
    end if
  end if
end if

The Maximal Matching Protocol

The transition function δ_M of the Maximal Matching protocol described in an algorithm based on message passing can be represented by Algorithm 12.

Note that $stateEdge_{(u,c(u))}$ denotes the state of the edge linking a node u to $c(v)$.

Algorithm 12 The Transition Function δ_M

if $((state_u = N)$ **and** $(state_{c(u)} = N))$ **then**
  $state_u \leftarrow A$
  $stateEdge_{(u,c(u))} \leftarrow 1$
end if

7.3 Simulation Results

Now to validate our theoretical results concerning the time complexity of some (mediated) population protocols under different scheduling scenarios and over different graph structures, we have the possibility of implementing them in ViSiDiA using one of the two models as described in the previous section.

In what follows, we present the results of the simulations of these different scenarios in ViSiDiA. For each scenario, the experiments were repeated 20 times.

In the sequel, we are going to use the following abbreviations:

- n: denoting the size of the interaction graph (the size of the population) where the protocol runs.
- Exp. Av.: denoting the experimental average of the stabilization time obtained with simulations. We recall that time is estimated in number of rounds of the protocol.
- Exp. U. B.: denoting the experimental upper bound, which is the highest value of the stabilization time of a given scenario reached in the simulations.
- Theo. U. B.: denoting the theoretical upper bound we established through the theoretical analyses of the stabilization time of the protocol.

7.3.1 The OR Protocol

The OR Protocol over Complete Graphs

We consider the population protocol OR running in a population whose interaction graph is the complete graph K_n. We present in Table 7.1 the results of the simulations of this protocol under the Random Scheduler, and under the *HS Scheduler*.

n	**Random Scheduler**		**HS Scheduler**		**Theo. U. B.**
	Exp. Av.	Exp. U. B.	Exp. Av.	Exp. U. B.	$O(n\ln(n))$
10	25	45	42	79	23
20	70	96	123	175	60
30	126	173	250	324	102
40	159	202	317	426	148
50	238	374	391	547	196
60	262	325	584	698	246
80	352	431	751	931	351
100	455	542	973	1078	461

Table 7.1: Simulations of the OR protocol over K_n

According to Corollary 5.6, the two schedulers are time equivalent with respect to this protocol. In fact, the expected value of the stabilization time of the OR protocol either running under the Random Scheduler, or under the *HS Scheduler*, is in this case $O(n\ln(n))$. This is confirmed by the results presented in Table 7.1.

However, we can notice that a computation under the *HS Scheduler* needs more computations steps (or more time) to stabilize. Indeed, we can conclude from Table 7.1 that the experimental average of this protocol running under the *HS Scheduler* is twice the experimental average under the Random Scheduler. This is related to the probability of an edge from the interaction graph K_n to be picked by each of these two schedulers.

In fact, we recall that this probability corresponds to $\frac{2}{n(n-1)}$ in the case of the Random Scheduler. However, it corresponds to $\frac{\beta}{(n-1)^2}$, with $0 < \beta < 1$, in the case of the *HS Scheduler*. We can conclude that, this probability is higher when the scheduler is the Random one. This can explain the differences between the obtained values of the two scheduling scenarios.

The OR Protocol over Random Graphs

We summarize in Table 7.2 the results of the simulations of the OR protocol running in a population whose interaction graph is $G_{n,p}$, with $p = \alpha\frac{\log n}{n}$ ($\alpha > 1$). We considered a scheduling of the pairwise interactions with the Random Scheduler on the one hand, and under the *HS Scheduler* on the other hand.

We can notice that, in both scenarios, the theoretical upper bounds of the stabilization time are respected. Also, according to these experimental results, the two schedulers are not time equivalent which validate the result of Corollary 5.9. In fact, the computation of the OR protocol in a graph $G_{n,p}$ stabilizes faster when running under the *HS Scheduler*.

n	α	**Random Scheduler**			**HS Scheduler**		
		Exp. Av.	Exp. U. B.	Theo. U. B. $O(n^2)$	Exp. Av.	Exp. U. B.	Theo. U. B. $O(n\log^2 n)$
40	2	297	567	1600	57	72	103
50	2.25	412	638	2500	60	69	144
60	2.69	392	555	3600	64	78	190
80	2.7	797	1035	6400	73	89	290
100	2.5	1204	1718	10000	82	108	400

Table 7.2: Simulations of the OR protocol over $G_{n,p}$

7.3.2 The Leader Election Protocol

The Leader Election Protocol over Complete Graphs

Let $\mathcal{P}$ be a population with a complete interaction graph K_n. We propose to simulate the Leader Election protocol running in this population by considering a scheduling under the Random Scheduler, or under the *HS Scheduler*. Table 7.3 shows the results of these simulations.

According to Theorem 5.10, the expected value of the stabilization time of the Leader Election is $O(n^2)$ when running in K_n under the Random Scheduler. Also, according to Theorem 5.11, the same upper bound holds for the computation of this protocol under the *HS Scheduler*. Results in Table 7.3 validate this upper bound.

As for the computation of the OR protocol, we can notice that the computation of the Leader Election in the complete graph under the *HS Scheduler* needs some additional

n	Random Scheduler		HS Scheduler		Theoretical U. B.
	Exp. Av.	Exp. U. B.	Exp. Av.	Exp. U. B.	$O(n^2)$
10	81	135	164	303	100
20	338	718	543	1027	400
30	918	2086	1361	2452	900
40	1406	3169	2977	6068	1600
50	2700	5769	4573	9786	2500
60	3284	5546	4169	7256	3600

Table 7.3: Simulations of the Leader Election protocol over K_n

computation steps compared to its computation under the Random Scheduler. The probability of an edge to be picked by the *HS Scheduler* has the minimum value when the graph is complete.

The Leader Election over Random Graphs

Now, we suppose that the population, over which the Leader Election protocol is running, has a connected random interaction graph $G_{n,p}$. We proceed on simulations of this protocol under this population by assuming that the pairwise interactions are handled by the Random Scheduler. Then, a second set of simulations is performed by assuming that the pairwise interactions are handled by the *HS Scheduler*. The results of these simulations are represented in Table 7.4.

n	α	Random Scheduler			HS Scheduler		
		Exp. Av.	Exp. U. B.	Theo. U. B. $O(\frac{n^3}{\log n})$	Exp. Av.	Exp. U. B.	Theo. U. B. $O(n \log^2 n)$
40	2	1615	3208	39949	364	747	103
50	2.25	2754	5287	73573	616	1482	144
60	2.69	3199	6164	121474	551	1208	190
80	2.7	6374	16134	269036	1118	1839	290
100	2.5	11475	18185	500000	1693	3414	400

Table 7.4: Simulations of the Leader Election protocol over $G_{n,p}$

Given Table 7.4, we can notice that both theoretical bounds related to the two considered scenarios are respected. Also, Corollary 5.15 is validated as the two schedulers are not time equivalent with respect to the Leader Election protocol. Indeed, the computations under the *HS Scheduler* stabilize faster.

Also, according to these experimental results, we think that a tighter theoretical bound related to the expected stabilization time of this protocol can be found.

7.3.3 The Maximal Matching Protocol

The Maximal Matching Protocol over Complete Graphs

We consider a population $\mathcal{P}$ whose interaction graph is the complete graph K_n. We suppose that the agents of this population are running the Maximal Matching mediated population protocol. Simulations of this computation are performed by assuming a scheduling under the Random Scheduler on the one hand, and under the *HS Scheduler* on the other hand. The obtained results are summarized in Table 7.5.

n	**Random Scheduler**		**HS Scheduler**		**Theoretical U. Bound**
	Exp. Av.	Exp. U. B.	Exp. Av.	Exp. U. B.	$O(n^2)$
10	95	230	89	214	100
20	405	857	459	1447	400
30	829	2371	983	2000	900
40	1345	3084	1959	5192	1600
50	2542	4528	2740	5576	2500
60	2676	5467	2914	5328	3600

Table 7.5: Simulations of the Maximal Matching protocol over K_n

Thanks to these simulation results, Corollary 5.18 is confirmed: the two schedulers are time equivalent. Both scenarios respect the theoretical upper bound $O(n^2)$.

7.3.4 The Leader Election with some Local Termination Detection

We consider the Leader Election population protocol with some local termination detection introduced in Section 6.2.2 running in a complete interaction graph K_n. We proceed on simulating the computation of this protocol under the Protocol Aware HS Scheduler. The results of these simulations are shown in Table 7.6.

We can notice that the theoretical upper bound $O(n)$ is respected.

We recall that introducing some local termination detection does not affect the stabilization time of a population protocol. Hence, the results related to the Leader Election population protocol that we presented in the previous sections hold also for the Leader Election population protocol with some local termination detection. Accordingly, comparing the results of Table 7.6 to those of Table 7.3, we can confirm that the computation of the Leader Election population protocol with some local termination detection stabilizes faster when running under the Protocol Aware HS Scheduler, compared to its computation under the Random Scheduler or the *HS Scheduler*.

7.3.5 The Iterated Protocols

We consider in this section the iterated protocols defined in the previous chapter. As an example of these protocols, we consider the Maximal Matching with some local termination detection (see Section 6.2.2).

n	Protocol Aware HS Scheduler		
	Exp. Av.	Exp. U. B.	Theo. U. B. $O(n)$
100	192	224	100
150	255	319	150
200	392	433	200
250	490	536	250
300	584	610	300

Table 7.6: Simulations of the Leader Election with some local termination detection protocol in K_n

The Iterated Maximal Matching in Complete Graphs

We focus on the computation of the Maximal Matching with some local termination detection in a complete graph under the scheduling of the Protocol Aware HS Scheduler. We proceeded on simulating this protocol, and the results are given in Table 7.7.

n	Protocol Aware HS Scheduler		
	Exp. Av.	Exp. U. B.	Theo. U. B. $O(n)$
100	98	105	100
150	140	152	150
200	188	204	200
250	245	255	250
300	296	306	300

Table 7.7: Simulations of the iterated Maximal Matching protocol under Protocol Aware HS Scheduler in K_n

Through the results of this table, we experimentally validate Lemma 6.11 related to the upper bound of this protocol running under the Protocol Aware HS Scheduler and which is $O(n)$.

Also, by referring to the experimental results presented in Table 7.5, we can conclude that an iterated Maximal Matching stabilizes faster when running under the Protocol Aware HS Scheduler compared to its computation either under the Random Scheduler or the *HS Scheduler*.

The Iterated Maximal Matching in Random Graphs

Let $\mathcal{P}$ be a population whose interaction graph is the connected random graph $G_{n,p}$. We simulate the computation of the iterated Maximal Matching in this population where the pairwise interactions are handled by the Protocol Aware HS Scheduler. We present the results of these experiments in Table 7.8.

The upper bound of this scenario, established in Lemma 6.14, is validated through these experimental results. Thanks to the Protocol Aware HS Scheduler, we have computations of the Maximal Matching protocol that quickly stabilize.

n	α	**Protocol Aware HS Scheduler**		
		Exp. Av.	Exp. U. B.	Theo. U. B. $O(-\frac{\log n}{\log\left(1-\frac{\beta}{\alpha \log n}\right)})$
100	2	7	8	16
1000	2	10	11	38
2000	2	11	13	46
4000	2	14	15	55
6000	2	16	18	61
8000	2	19	21	65
10000	2	22	25	69

Table 7.8: Simulations of the iterated Maximal Matching protocol under Protocol Aware HS Scheduler in $G_{n,p}$

7.4 Conclusion

We presented in this chapter the ViSiDiA platform, the Java framework designed for implementing and simulating distributed algorithm. We realized through this platform the possible mappings of the population protocol model to tasks with graph relabeling systems, and to anonymous asynchronous distributed algorithms based on message passing. We illustrated these mapping through some examples of protocols, so that the user of ViSiDiA can reproduce this for any protocol.

We implemented the different schedulers introduced in this work in this platform. We then simulated each of the studied protocols by considering different scenarios. Thanks to the experimental results of these simulations, we validated the theoretical results we obtained for the stabilization times of these protocols.

Conclusion

The pairwise computational model of population protocols was designed for anonymous asynchronous networks of tiny objects that can be passively mobile. A pairwise interaction in fact synchronizes two agents that exchange their states and update them accordingly. However, Angluin et al. do not specify how this synchronization and this simultaneous bidirectional exchange take place.

We noticed that the population protocols present some similarities with a local computational model in graphs, which is tasks with graph relabeling systems. This latter also uses an abstract communication model. Also, among its possible interaction models, there are pairwise interactions. We established a comparative study to explore the existing correspondences and the key differences between the (mediated) population protocols and tasks with graph relabeling system. As a result, we proposed a mapping approach. We proved that the mapping of the execution of a (mediated) population protocol to a realization of a task according to a graph relabeling system is always possible. However, the reverse mapping is possible only under some conditions.

Then, we proposed to describe the computation of a (mediated) population protocol, and more particularly, the simultaneous bidirectional pairwise interaction, through a less theoretical model. We considered the model of anonymous asynchronous message passing with port numbering. Hence, a population protocol is described as a distributed algorithm based on message passing and running in an anonymous asynchronous system. The pairwise interaction is thus split into two phases: a synchronization phase and a simultaneous send and receive phase.

Based on these two comparative studies, we established a bridge between the: population protocol model and tasks with graph relabeling systems model that are based on abstract communications and anonymous asynchronous message passing model that uses explicit communications.

The pairwise interactions in population protocols are under the control of a scheduler. In the context we presented above, the role of the scheduler includes the synchronization of the communicating pair of agents. We proposed an implementation of the Random Scheduler, introduced by Angluin et al., based on message passing according to the mapping we defined. Also, as all the existing schedulers in population protocols are sequential, we proposed a $1-$central scheduler called the *HS Scheduler*. This scheduler is able to capture concurrent interactions. It is based on randomized handshakes. We also proposed a suitable algorithm describing this scheduler and based on message passing. Through the analysis of this algorithm, we proved that this scheduler is fair with probability 1.

We then established a comparative study to investigate the time equivalence of the Random Scheduler and this *HS Scheduler*. This was presented through the analyses of

the stabilization time of three protocols: the OR protocol, the Leader Election and the Maximal Matching. We proved that the two schedulers are time equivalent with respect to these protocols when the computations are over complete interaction graphs. However, in random interaction graph, the *HS Scheduler* allows faster stabilization.

We then introduced population protocols with some detection of local termination. We proposed a slightly modified version of the *HS Scheduler*, called the Protocol Aware HS Scheduler, to introduce termination to this model of population protocols. We also proved that this version of the protocol is fair with probability 1. We concluded that, with respect to the Leader Election protocol and to the Maximal Matching protocol, the Protocol Aware HS Scheduler is not time equivalent to the Random Scheduler and *HS Scheduler*. In fact, computations that take place under its scheduling stabilize faster.

Finally, we illustrated through the ViSiDiA platform the mapping approach of a (mediated) population protocol to a task with graph relabeling system, or to a distributed anonymous asynchronous algorithm based on message passing. We also implemented in this platform the protocols we studied, as well as the schedulers procedures we proposed. The experimental results we obtained through simulations allowed us to validate our theoretical results about the stabilization times of these protocols under different scheduling scenarios.

Future Directions

As a future direction of this thesis, we can consider the analysis of the Handshake Scheduler Algorithm under other interaction graphs representing the network of tiny wireless objects.

An interesting graph structure for such networks is a random geometric graph. A random geometric graph $G_{n,r}$ is an undirected graph with n nodes placed randomly in a given area. Two nodes share a communication link if and only if the distance separating them is smaller than r. Indeed, it is a realistic representation of the structure of the network where the objects communicate via a wireless media. The distance r can represent the communication range of this wireless media.

Another interesting graph structure is a small world graph. A small world graph is a random graph with short average path length and high clustering. In fact, there are recently many works proposing to represent networks of tiny objects, such as wireless sensor networks, as small world networks.

Then, the stabilization times of the studied protocols running in these graphs under the *HS Scheduler* can be analyzed. The obtained theoretical results can be validated through simulations in ViSiDiA.

Another future direction is to extend the pairwise interaction model of population protocols. In fact, wireless networks allow, in addition to point to point communications, one to many communications. That is, an agent in a wireless network can communicate with all its direct neighborhood. Therefore, extending the population protocols to handle such interactions is interesting. We can consider them as star communications where the

center node and all its neighborhood exchange their respective states. This can correspond to $LC1$ or $LC2$ interactions in local computations in graphs.

A probabilistic scheduler can also be designed for these interactions. It can be inspired from the randomized algorithms implementing the synchronizations of these interactions (either $LC1$ or $LC2$). We can work on proving the fairness of this scheduler.

We can then analyze the stabilization times of the studied protocols extended to communicate in stars and running under this scheduler. A possible direction is to investigate the time equivalence of the different proposed schedulers with respect to some protocols.

Bibliography

[1] W. Abdou, N. Ouled Abdallah, and M. Mosbah. ViSiDiA: A Java Framework for Designing, Simulating, and Visualizing Distributed Algorithms. In *IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pages 43–46, 2014.

[2] D. Angluin. Local and global properties in networks of processors. In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, pages 82–93. ACM, 1980.

[3] D. Angluin, J. Aspnes, M. Chan, Michael J. Fischer, H. Jiang, and R. Peralta. Stably computable properties of network graphs. In Viktor K. Prasanna, Sitharama Iyengar, Paul Spirakis, and Matt Welsh, editors, *Proceedings of First IEEE International Conference on Distributed Computing in Sensor Systems*, volume 3560 of *Lecture Notes in Computer Science*, pages 63–74. Springer-Verlag, June 2005.

[4] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18:235–253, 2006.

[5] D. Angluin, J. Aspnes, Z. Diamadi, Michael J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. In *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing, PODC 2004, St. John's, Newfoundland, Canada, July 25-28, 2004*, pages 290–299, 2004.

[6] D. Angluin, J. Aspnes, and D. Eisenstat. Stably computable predicates are semilinear. In *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing*, 2006.

[7] D. Angluin, J. Aspnes, and D. Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, 21:183–199, 2008.

[8] D. Angluin, J. Aspnes, and D. Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 21(2):87–102, July 2008.

[9] D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. On the power of anonymous one-way communication. In *9th International Conference on Principles of Distributed Systems*, volume 3974 of *Lecture Notes in Computer Science*, pages 396–411, December 2005.

[10] D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, November 2007.

[11] D. Angluin, J. Aspnes, Michael J. Fischer, and H. Jiang. Self-stabilizing population protocols. *ACM Transactions on Autonomous and Adaptive Systems*, 3(4):13, 2008.

[12] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics.* John Wiley & Sons, 2004.

[13] Bollobas B. Random graphs. *Cambridge University Press*, 2001.

[14] M. Bauderon, Y. Métivier, M. Mosbah, and A. Sellami. From local computations to asynchronous message passing systems, 2002.

[15] M. Bauderon, Y. Métivier, M. Mosbah, and A. Sellami. Graph relabelling systems: A tool for encoding, proving, studying and visualizing distributed algorithms. *Electronic Notes in Theoretical Computer Science*, 51:93 – 107, 2002. {GETGRATS} Closing Workshop.

[16] M. Bauderon and M. Mosbah. A unified framework for designing, implementing and visualizing distributed algorithms. *Electronic Notes in Theoretical Computer Science*, 72(3):13 – 24, 2003.

[17] J. Beauquier, C. Johnen, and S. Messika. *Proceedings 8th International Symposium: Stabilization, Safety, and Security of Distributed Systems*, chapter All k-Bounded Policies Are Equivalent for Self-stabilization, pages 82–94. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[18] L. Becchetti, L. Bergamini, F. Ficarola, F. Salvatore, and A. Vitaletti. First Experiences with the Implementation and Evaluation of Population Protocols on Physical Devices. In *IEEE International Conference on Green Computing and Communications (GreenCom) 2012*, pages 335–342, 2012.

[19] C. Berge. *Graphs and Hypergraphs.* Elsevier Science Ltd., Oxford, UK, UK, 1985.

[20] M. Billaud, P. Lafon, Y. Metivier, and E. Sopena. *Graph rewriting systems with priorities*, pages 94–106. Springer Berlin Heidelberg, Berlin, Heidelberg, 1990.

[21] P. Boldi, S. Shammah, S. Vigna, B. Codenotti, P. Gemmell, and J. Simon. Symmetry breaking in anonymous networks: Characterizations, 1996.

[22] Paolo Boldi and Sebastiano Vigna. Fibrations of graphs. *Discrete Mathematics*, 243(1):21 – 66, 2002.

[23] P. Castéran and V. Filou. Tasks, types and tactics for local computation systems. *Stud. Inform. Univ.*, 9(1):39–86, 2011.

[24] J. Chalopin and Y. Métivier. *Mathematical Foundations of Computer Science 2005: 30th International Symposium, MFCS 2005, Gdansk, Poland, August 29–September 2, 2005. Proceedings*, chapter A Bridge Between the Asynchronous Message Passing Model and Local Computations in Graphs, pages 212–223. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[25] I. Chatzigiannakis, Sh. Dolev, Sándor P. Fekete, O. Michail, and Paul G. Spirakis. Not all fair probabilistic schedulers are equivalent. In *Principles of Distributed Systems*. Springer Berlin Heidelberg, 2009.

[26] I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, and Paul G. Spirakis. Passively Mobile Communicating Logarithmic Space Machines. *CoRR*, abs/1004.3395, 2010.

[27] I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, and Paul G. Spirakis. Passively mobile communicating machines that use restricted space. *Theoretical Computer Science*, 412(46):6469 – 6483, 2011.

[28] I. Chatzigiannakis, O. Michail, S. Nikolaou, and Paul G. Spirakis. The computational power of simple protocols for self-awareness on graphs. *Theoretical Computer Science*, 512:98 – 118, 2013.

[29] I. Chatzigiannakis, O. Michail, and P. G. Spirakis. Mediated population protocols. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part II*, 2009.

[30] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974.

[31] S. Dubois and S. Tixeuil. A taxonomy of daemons in self-stabilization. *CoRR*, abs/1110.0334, 2011.

[32] P. Duchon, N. Hanusse, N. Saheb, and A. Zemmari. Broadcast in the rendezvous model. *Information and Computation*, 204(5):697 – 712, 2006.

[33] P. Erdős and A. Rényi. On random graphs i. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959 1959.

[34] P Erdős and A. Rényi. On the evolution of random graphs. *Publication of the Mathematical Institute of the Hungarian Academy of Sciences*, 5:17–61, 1960.

[35] F. Fich and E. Ruppert. Hundreds of impossibility results for distributed computing. *Distributed Computing*, 16(2):121–163, 2003.

[36] M. Fischer and H. Jiang. Self-stabilizing leader election in networks of finite-state anonymous agents. In *10th International Conference on Principles of Distributed Systems*, pages 395–409. Springer, 2006.

[37] A. Fontaine, M. Mosbah, M. Tounsi, and A. Zemmari. A fault-tolerant handshake algorithm for local computations. In *30th International Conference on Advanced Information Networking and Applications Workshops, AINA 2016 Workshops, Crans-Montana, Switzerland, March 23-25, 2016*, pages 475–480, 2016.

[38] S. Ghosh. *Distributed systems : an algorithmic approach*. Chapman & Hall/CRC, Boca Raton, London, New York, 2007.

[39] E. Godard, Y. Métivier, and A. Muscholl. Characterizations of classes of graphs recognizable by local computations. *Theory of Computing Systems*, 37(2):249–293, 2004.

[40] E. Godard, Y. Métivier, and G. Tel. Termination detection of local computations. *CoRR*, abs/1001.2785, 2010.

[41] R. Guerraoui and E. Ruppert. Names trump malice: Tiny mobile agents can tolerate byzantine failures. In *Proceedings of the 36th Internatilonal Colloquium on Automata, Languages and Programming (ICALP)*, pages 484–495, 2009.

[42] R. Gupta, S. A. Smolka, and S. Bhaskar. On randomization in sequential and distributed algorithms. *ACM Comput. Surv.*, 26(1):7–86, March 1994.

[43] J.G. Kemény and J.L. Snell. *Finite markov chains*. University series in undergraduate mathematics. Van Nostrand, 1960.

[44] C. Lavault. *Evaluation des algorithmes distribués – Analyse, complexité, méthodes*. collection Informatique. Lavoisier 2000-2010, 1995.

[45] I. Litovsky, Y. Métivier, and E. Sopena. Different local controls for graph relabeling systems. *Mathematical systems theory*, 28(1):41–65, 1995.

[46] I. Litovsky, Y. Métivier, and E. Sopena. Graph relabelling systems and distributed algorithms. In *Handbook of graph grammars and computing by graph transformation*, pages 1–56. World Scientific Publishing Co., Inc., 2001.

[47] I. Litovsky, Y. Metivier, and W. Zielonka. On the recognition of families of graphs with local computations. *Information and Computation*, 118(1):110 – 119, 1995.

[48] Igor Litovsky, Yves Mtivier, and Eric Sopena. Checking global graph properties by means of local computations: the majority problem. *Electronic Notes in Theoretical Computer Science*, 2:199 – 206, 1995.

[49] N. Lynch. A hundred impossibility proofs for distributed computing. In *Proceedings of the eighth annual ACM Symposium on Principles of distributed computing*, pages 1–28. ACM, 1989.

[50] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[51] Y. Métivier, M. Mosbah, R. Ossamy, and A. Sellami. *Synchronizers for Local Computations*, pages 271–286. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[52] Y. Métivier, N. Saheb, and A. Zemmari. *Randomized Rendezvous*, pages 183–194. Birkhäuser Basel, Basel, 2000.

[53] Y. Métivier, N. Saheb, and A. Zemmari. Analysis of a randomized rendezvous algorithm. *Inf. Comput.*, 184(1):109–128, 2003.

[54] O. Michail, I. Chatzigiannakis, and Paul G. Spirakis. *New Models for Population Protocols*. N. A. Lynch (Ed), Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool, 2011.

[55] O. Michail, I. Chatzigiannakis, and Paul G. Spirakis. Terminating Population Protocols via Some Minimal Global Knowledge Assumptions. In *Stabilization, Safety, and Security of Distributed Systems (SSS)*, volume 7596 of *Lecture Notes in Computer Science*, pages 77–89. Springer Berlin Heidelberg, 2012.

[56] O. Michail and Paul G. Spirakis. Terminating population protocols via some minimal global knowledge assumptions. *Journal of Parallel and Distributed Computing*, 8182:1 – 10, 2015.

[57] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995.

[58] R. Motwani and P. Raghavan. Randomized algorithms. *ACM Comput. Surv.*, 28(1):33–37, March 1996.

[59] B. Neggazi, M. Haddad, and H. Kheddouci. *Stabilization, Safety, and Security of Distributed Systems: 14th International Symposium, SSS 2012, Toronto, Canada, October 1-4, 2012. Proceedings*, chapter Self-stabilizing Algorithm for Maximal Graph Partitioning into Triangles, pages 31–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[60] N. Ouled Abdallah, H. Hadj Kacem, M. Mosbah, and A. Zemmari. Randomized broadcasting in wireless mobile sensor networks. *Concurrency and Computation: Practice and Experience*, 25(2):203–217, 2013.

[61] N. Ouled Abdallah, M. Jmaiel, M. Mosbah, and A. Zemmari. A totally distributed fair scheduler for population protocols by randomized handshakes. In Lecture Notes in Computer Science, editor, *ICTAC*, volume 9399. Springer, 2015.

[62] M. Presburger. Über die vollständigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchem die addition als einzige operation hervortritt. In *Comptes Rendus du I congrs de Mathmaticiens des Pays Slaves*, page 92101, Warszawa, 1929.

[63] Kenneth H. Rosen. *Handbook of discrete and combinatorial mathematics*. CRC Press, Boca Raton (Fla.), 2000. Contient des biographies de mathmaticiens.

[64] P. Rosenstiehl, J.R. Fiksel, and A. Holliger. Intelligent graphs: Networks of finite automata capable of solving graph problems. In RONALD C. READ, editor, *Graph Theory and Computing*, pages 219 – 265. Academic Press, 1972.

[65] S. Schmid and R. Wattenhofer. Algorithmic models for sensor networks. In *Proceedings of the 20th International Conference on Parallel and Distributed Processing*, IPDPS'06, pages 177–177, Washington, DC, USA, 2006. IEEE Computer Society.

[66] J. Suomela. Survey of local algorithms. *ACM Comput. Surv.*, 45(2):24:1–24:40, March 2013.

[67] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, Cambridge, 2nd edition, 2000.

[68] J. Von Neumann and A.W. Burks. Theory of self-reproducing automata. *University of Illinois Press, Urbana, Illinois*, 1966.

[69] D. Williams. *Probability with Martingales*. Cambridge University Press, 1991.

[70] M. Yamashita and T. Kameda. Computing on anonymous networks. i. characterizing the solvable cases. *IEEE Transactions on Parallel and Distributed Systems*, 7(1):69–89, Jan 1996.

[71] A. Zemmari. On handshakes in random graphs. *Information Processing Letters*, 108(3):119 – 123, 2008.