UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

LE TEMPS DU RENDEZ-VOUS ANONYME DANS LES ARBRES : ALGORITHMES DÉTERMINISTES VERSUS ALGORITHMES ALÉATOIRES

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR

SAMIR ELOUASBI

SOUS LA SUPERVISION DU PROFESSEUR ANDRZEJ PELC

MAI 2012

UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

Département d'informatique et d'ingénierie

Ce mémoire intitulé :

LE TEMPS DU RENDEZ-VOUS ANONYME DANS LES ARBRES : ALGORITHMES DÉTERMINISTES VERSUS ALGORITHMES ALÉATOIRES

présenté par

Samir Elouasbi

pour l'obtention du grade de maître ès science (M.Sc.)

a été évalué par un jury composé des personnes suivantes :

Dr. Andrzej Pelc ..... Directeur de recherche
Dr. Luigi Logrippo ..... President du jury
Dr. Jurek Czyzowicz ..... Membre du jury

Mémoire accepté le : 9 mai 2012

À mon père pour tous les sacrifices qu'il a faits pour moi et pour la confiance qu'il m'a accordée.

À ma chère mère qui était mon premier professeur et qui était toujours souciée de mes études depuis mon enfance jusqu'à ce que j'ai eu mes enfants.

À mon épouse Kaoutar qui mérite que son nom soit mis à côté du mien dans ce mémoire. Sans son support et son aide, je n'aurais jamais eu le temps de poursuivre mes études.

À mon grand Omar, ma princesse Siryne et mon petit Ali qui ont sacrifié des heures de jeux pour laisser leur papa faire ses études.

À mon frère Zakaria, mes soeurs et leurs enfants.

À mon grand frère siDriss Iraqi et à ma grande soeur Ikhlass pour tout le support et l'aide qu'ils ont consacrés à moi et à ma famille pour me permettre de réaliser ce travail.

Remerciement

Je tiens fortement à remercier mon professeur et mon directeur de recherche Monsieur Andrzej Pelc pour son encadrement, ses conseils et sa patience. Je lui serais toujours reconnaissant pour son soutien et pour ses encouragements pour la réalisation de ce travail.

J'aimerais aussi remercier Monsieur Luigi Logrippo et Monsieur Jurek Czyzowicz pour le temps qui ont consacré pour valider ce mémoire et pour les remarques pertinentes qu'ils ont faites pour améliorer la qualité de ce mémoire.

Je profite aussi de cette occasion pour remercier ma directrice au travail JoAnne St-Gelais pour son soutien, sa flexibilité et son encouragement.

Enfin, un grand MERCI à tous mes amis qui m'ont soutenu par leurs encouragements et leurs belles paroles.

Table des matières

Résumé

Deux agents mobiles identiques (anonymes) démarrent à partir des nœuds arbitraires distincts d'un arbre inconnu et se déplacent le long de ses arêtes afin de se rencontrer dans un certain nœud. Les agents se déplacent durant des rondes synchrones : durant chaque ronde, un agent peut rester dans son nœud actuel ou se déplacer vers un de ses voisins. Nous étudions le temps optimal nécessaire pour accomplir ce rendez-vous. Dans le cas déterministe, nous cherchons des algorithmes qui permettent d'accomplir le rendez-vous quand c'est possible, tandis que pour le cas aléatoire nous cherchons des algorithmes presque certains, qui permettent de faire le rendez-vous avec une probabilité de succès d'au moins $1-1/n$ dans des arbres d'ordre n, pour n suffisamment grand.

Notre contribution pour le cas déterministe peut se résumer dans les points suivants :

- Nous avons présenté un algorithme déterministe qui permet à des agents de faire un rendez-vous, lorsque c'est possible, dans n'importe quel arbre en temps optimal $O(n)$ où n représente le nombre de nœuds de cet arbre.
- Nous avons prouvé que le rendez-vous s'avère impossible si et seulement si les agents se trouvent sur des positions initiales symétriques et que leur départ est simultané.

Pour ce qui est du cas aléatoire :

- Nous avons réalisé un algorithme presque certain qui permet aux agents d'accomplir le rendez-vous en temps espéré optimal $O(n)$ dans n'importe quel arbre d'ordre n quel que soit la situation dans laquelle se trouvent les agents.
- Nous avons amélioré le coût espéré à $O(\log n)$ dans le cas où les agents connaissent la distance initiale D constante qui les séparent, Δ le degré maximum constant de

l'arbre et le nombre de noeuds n de l'arbre.

mots clés : agent mobile, rendez-vous, algorithme déterministe, algorithme aléatoire.

Abstract

Two identical (anonymous) mobile agents start from arbitrary nodes of an unknown tree and move along its edges with the goal of meeting at some node. Agents move in synchronous rounds : in each round an agent can either stay at the current node or move to one of its neighbors. We study optimal time of completing this rendezvous task. For deterministic rendezvous we seek algorithms that achieve rendezvous whenever possible, while for randomized rendezvous we seek almost safe algorithms, which achieve rendezvous with probability at least $1 - 1/n$ in n-nodes tree, for sufficiently large n.

Our contribution to the deterministic case can be summarized in the following points :

- We presented a deterministic algorithm that allows agents to make rendezvous, when it is possible, in any n-node tree in optimal time $O(n)$.
- We proved that the rendezvous is impossible if and only if agents are on symmetric initial positions and that their departure is simultaneous.

For the random case :

- We presented an almost safe algorithm that allows agents to perform the rendezvous in optimal expected running time $O(n)$ in any n-node tree for any initial positions of the agents.
- We improved the expected cost to $O(\log n)$ in the case where agents know the constant distance D between there initial positions, the constant maximum degree Δ of the tree and the number of nodes n of the tree.

Keywords : mobile agent, rendezvous, deterministic algorithm, randomized algorithm.

Chapitre 1

Introduction

Dans la vie quotidienne, il est souvent très simple de faire un rendez-vous : utiliser un téléphone pour appeler l'autre personne, lui envoyer une invitation par la poste ou par courriel, etc.. Mais en réalité, un rendez-vous peut être aussi un problème très compliqué. Supposons que vous avez perdu votre ami dans une grande forêt et que chacun de vous essaye de trouver l'autre, comment faire dans l'absence des moyens de communication ?

Dans le domaine de l'informatique, le problème du *rendez-vous* est fort présent en calcul distribué, dans les réseaux et dans l'Internet, pour effectuer le partage des tâches et l'échange des informations à l'aide des entités mobiles. Pour mieux comprendre ce problème, nous présentons la situation suivante.

Deux entités mobiles doivent se rencontrer dans un nœud du réseau de communication ou dans une labyrinthe pour effectuer ensemble une tâche de calcul ou se partager de l'information. Le réseau est modélisé par un graphe connexe et non-orienté, et les entités mobiles sont des logiciels ou des robots qui se déplacent d'un nœud à un autre du graphe à travers ses arêtes. Dans tout le reste de ce mémoire, nous allons attribuer à une entité mobile le nom *agent mobile* ou *agent* tout court.

Pour accomplir le rendez-vous, les agents se déplacent dans le réseau durant des rondes synchrones. La tâche est accomplie lorsqu'ils se rencontrent dans le même nœud au même moment ; les agents ne peuvent pas faire le rendez-vous à l'intérieur d'une arête même si durant une certaine ronde, leurs chemins se croisent.

Dans ce mémoire, nous présentons des algorithmes de rendez-vous qui ne dépendent pas de la connaissance des étiquettes des nœuds et qui peuvent permettre la réalisation de la rencontre, lorsque c'est possible, dans des graphes anonymes. Notre intérêt à concevoir de tels algorithmes est motivé par les deux situations suivantes : la première est que si les

agents peuvent connaître les étiquettes des nœuds, ils peuvent par exemple se rencontrer dans le nœud qui possède le plus grand identifiant. Dans ce cas, le problème du rendez-vous peut être transformé en une tâche de parcours du graphe pour trouver l'étiquette désignée. La deuxième est que souvent les agents ne peuvent pas identifier les nœuds car ils peuvent être incapable de lire les étiquettes qui leur sont associées ou bien pour des raisons de sécurité et de confidentialité, les nœuds refusent de révéler leurs identifiants.

D'autre part, nous considérons que les arêtes incidentes à un nœud v ont des étiquettes distinctes appartenant à $\{0, 1, ..., d-1\}$, où d est le degré de v. En d'autres termes, chaque arête $\{u, v\}$ du graphe possède deux étiquettes qui s'appellent des *numéros de port* en u et en v. Ces numéros de port sont visibles aux agents. Dans le cas contraire, un agent pourra être bloqué après la visite du deuxième voisin parce qu'il ne sait pas si le troisième port qu'il va choisir va l'amener à un nouveau voisin ou à un voisin déjà visité ; et dans cette situation, le rendez-vous s'avère souvent impossible. La numérotation des ports est *locale*, c'est à dire qu'il n'existe aucune relation entre les numéros de ports associés à deux nœuds différents.

Dans cette recherche, nous nous intéressons au problème du rendez-vous dans les arbres. Notre but est de réaliser des algorithmes optimaux qui permettent aux agents de se rencontrer en un temps minimal dans cette structure de réseaux. Deux types d'algorithmes vont être présentés : déterministes et aléatoires. Dans le cas déterministe, nous présentons un algorithme qui permet d'accomplir le rendez-vous dans tous les cas ou la rencontre est possible ; tandis que pour le cas aléatoire, nous présentons des algorithmes presque certains, c'est à dire des algorithmes qui accomplissent le rendezvous avec une probabilité de succès d'au moins $1 - 1/n$ dans un arbre à n nœuds, pour n suffisamment grand.

Pour atteindre ce but, nous avons structuré ce document de la manière suivante :

- Le chapitre 1 présente une introduction du problème du rendez-vous dans le réseau et une brève description des objectifs de ce mémoire.
- Le chapitre 2 comporte une revue de la littérature et quelques exemples des travaux qui ont été réalisés concernant le problème du rendez-vous.
- Le chapitre 3 présente le modèle en détail.
- Le chapitre 4 donne un résumé des résultats de nos recherches.
- Le chapitre 5 traite du rendez-vous déterministe dans la ligne et présente notre algorithme qui résout ce problème.

- Le chapitre 6 traite du rendez-vous déterministe dans l'arbre et présente notre algorithme qui résout ce problème, ainsi qu'une borne inférieure pour le temps du rendez-vous.
- Le chapitre 7 traite du rendez-vous aléatoire dans l'arbre et présente les algorithmes presque certains qui résolvent ce problème, ainsi qu'une borne inférieure pour le temps du rendez-vous.
- Le chapitre 8 porte sur les simulations pour tester les algorithmes aléatoires.
- Le chapitre 9 présente une conclusion de tout le travail qui a été fait dans cette recherche et l'énoncé des travaux futurs.
- Ce document se termine par une annexe qui contient le code de la simulation et par la bibliographie utilisée pour réaliser ce mémoire.

Chapitre 2

Revue de la littérature

2.1 Introduction

Parmi les anciens écrits dans lesquels a été annoncé la notion du rendez-vous est celui de Schelling écrit en 1960. L'auteur propose à ses lecteurs la situation suivante [26] : deux personnes sont parachutées dans deux endroits différents sans aucun moyen de communication. Ils savent que chacun d'eux possède la même carte mais aucun d'eux ne connaît l'emplacement de l'autre. Comment ces deux parachutistes peuvent se rencontrer dans un temps raisonnable ? Est ce que les informations qui se trouvent dans les cartes peuvent les pousser à penser à un endroit précis pour se rencontrer ? Schelling affirme que le point commun entre toutes les « solutions » pour résoudre ce problème est de trouver un *emplacement spécifique* pour faire la rencontre. On peut dire que cet emplacement va dépendre du lieu, de l'environnement et aussi des connaissances que chaque participant possède. Le problème du rendez-vous n'a été baptisé qu'en 1990 par Anderson et Weber [4] et c'était à partir de cette année que plusieurs chercheurs commencèrent à s'intéresser à cette classe de problèmes, et parmi eux ceux du domaine de l'informatique.

Tel que mentionné dans la partie introduction de ce mémoire, nous allons nous intéresser aux écrits et aux résultats obtenus pour résoudre le problème du rendez-vous pour deux agents mobiles qui se trouvent dans un réseau qui est modélisé par un graphe (ligne, anneau, arbre, etc.). Dans cette revue de la littérature, nous allons diviser les écrits en deux parties : la première est celle qui traite le cas où les agents se déplacent d'une manière *déterministe* dans le réseau ; et la deuxième est celle qui s'intéresse au cas des déplacements *aléatoires*.

2.2 Le rendez-vous synchrone déterministe

La tâche la plus cruciale pour réaliser un rendez-vous avec un comportement déterministe des agents est de casser la symétrie du réseau. Examinons cet exemple [10] : deux agents mobiles identiques se trouvent sur des positions différentes d'un graphe composé de deux nœuds liés par une arête. Ils commencent à exécuter le même algorithme déterministe au même moment. Il est clair que ces agents ne vont jamais se rencontrer parce que à chaque ronde, soit ils sont immobiles sur des nœuds différents, soit ils se déplacent vers deux positions différentes. Pour briser la symétrie, il existe trois façons [16] :

La première est de distinguer les agents en leur attribuant des étiquettes différentes. Dans ce cas, chaque agent connaît son identifiant sans connaître celui de l'autre. Si on reprend l'exemple du graphe à deux nœuds, un algorithme très simple pour réaliser le rendez-vous serait de parcourir l'arête L fois, où L est l'étiquette de l'agent ; et dans ce cas, celui qui a le plus grand identifiant va rejoindre l'autre agent, une fois ce dernier est immobilisé.

La deuxième façon pour briser la symétrie est celle du marquage des nœuds à l'aide des jetons ou des tableaux blancs (*whiteboards*). Il existe deux types de jetons : un jeton *immobile* qui ne peut pas être enlevé une fois utilisé pour marquer un nœud, et un jeton *mobile* qui peut être assigné par un agent et enlever par lui même ou par un autre agent. Pour ce qui est du tableau blanc, chaque nœud en possède un qui est initialement vide et qui permet au visiteur de lire, d'écrire ou d'effacer un message tout en respectant la taille de la mémoire alloué à ce tableau. Les jetons et les tableaux permettent d'exploiter la non-symétrie initiale des positions des agents.

Et enfin, la troisième façon est celle d'exploiter la non-symétrie soit du réseau (par exemple une ligne avec un nombre impair de nœuds), soit des positions initiales des agents même si le réseau est symétrique [30]. Ceci est souvent possible même si les nœuds ne peuvent pas être marqués par les agents.

Lorsqu'on parle du problème du rendez-vous dans le cas synchrone et déterministe, le déplacement des agents dans le réseaux se fait d'un nœud à un autre durant des rondes synchrones, et la rencontre doit se faire dans un nœud durant une certaine ronde.

Les trois problèmes suivants sont étudiés dans la littérature de ce sujet :

1. la faisabilité du rendez-vous : est ce que la rencontre est possible ou non selon les cas à étudier. Ceci va dépendre de la topologie du réseau, les positions initiales des agents et les informations dont ces derniers disposent.

2. la réalisation du rendez-vous dans un temps optimal ; dans la littérature, on parle du coût de la rencontre. On peut trouver plusieurs algorithmes qui peuvent résoudre le problème, mais à quel prix ? Dans ce cas, le coût va être calculé en nombre de rondes effectuées jusqu'à ce que les agents se rencontrent.
3. la capacité de la mémoire : puisqu'on travaille avec des agents intelligents, chacun d'eux est équipé d'une mémoire. Donc, il faut chercher des solutions qui utilisent le moins d'espace mémoire possible pour réaliser la rendez-vous.

Pour aider le lecteur à bien comprendre les résultats de recherche et suivre leurs améliorations, nous allons diviser la littérature concernant le rendez-vous déterministe en deux parties : le rendez-vous sans marquage des nœuds et celui avec le marquage possible. Nous allons aussi utiliser cette approche lorsqu'on va présenter le problème du rendez-vous aléatoire dans la deuxième partie de ce chapitre.

2.2.1 Le rendez-vous sans aucun moyen de marquer les nœuds

Dessmark, Fraigniaud, Kowalski et Pelc étaient les premiers à donner une solution complète pour résoudre le problème du rendez-vous dans un graphe connexe quelconque [10]. Voici la situation : deux agents se déplacent dans un réseau anonyme d'une manière synchrone. Durant chaque ronde, un agent peut soit rester immobile sur le même nœud, soit se déplacer vers un nœud adjacent. Chaque agent possède un identifiant unique sous forme d'un entier positif, ce qui veut dire que l'un des deux identifiants est plus petit que l'autre et va être noté par l. Un agent connaît son propre identifiant sans avoir la moindre idée sur celui de l'autre. Le rendez-vous doit se réaliser dans un nœud durant la même ronde et les auteurs [10] distinguent 2 cas : le premier est celui lorsque les deux agents commencent l'exécution de l'algorithme du rendez-vous au même moment et on fait référence au départ simultané des agents, et le deuxième qui s'appelle le départ arbitraire, c'est lorsque l'instant de départ de chaque agent est arbitraire.

Les auteurs de [10] se sont intéressés à calculer le coût du rendez-vous dans le graphe anonyme (sa topologie n'est pas connue par les agents) en fonction de l, de n qui est le nombre de nœuds dans le graphe, de D qui représente la distance entre les positions initiales des agents et τ qui est la différence de temps entre les instants de départ des agents.

Le premier résultat s'applique aux arbres. Les auteurs présentent un algorithme qui permet aux agents de réaliser le rendez-vous en temps $O(n + \log l)$ dans n'importe quel

arbre d'ordre n et même dans le cas du départ arbitraire des agents. Ils prouvent aussi que ce résultat est optimal, c'est à dire qu'il n'existe aucun algorithme déterministe, qui pour une classe d'arbres à n nœuds, peut résoudre le problème du rendez-vous en un temps meilleur que $\Omega(n + \log l)$, même si le départ des agents est simultané.

Puisque les graphes peuvent contenir des cycles, les auteurs ont construit un deuxième algorithme pour réaliser une rencontre dans un anneau quelconque à n nœuds. Dans le cas d'un départ simultané des agents, le coût du rendez-vous est dans l'ordre de $\Theta(D \log l)$ et il n'existe aucun algorithme déterministe qui pourrait en faire mieux. Dans le cas d'un départ arbitraire des agents, les auteurs présentent deux autres algorithmes : le premier fonctionne en temps $O(n \log l)$ lorsque les agents connaissent le nombre de nœuds qui forment l'anneau, et le deuxième fait le rendez-vous en temps $O(l\tau + ln^2)$ dans le cas contraire ; ils prouvent aussi qu'il n'existe aucun algorithme déterministe qui peut permettre aux agents d'accomplir le rendez-vous dans un temps meilleur que $\Omega(n + D \log l)$.

Pour conclure cette contribution, les auteurs de [10] ont présenté un algorithme pour résoudre le problème du rendez-vous déterministe qui fonctionne en temps $O(n^5\sqrt{\tau \log l} \log n + n^{10} \log^2 n \log l)$ dans n'importe quel graphe.

Deux ans plus tard, Kowalski et Malinowski [15] ont réussi à rendre optimal le temps du rendez-vous déterministe dans l'anneau en fournissant un algorithme qui fonctionne en temps $\Theta(n \log l)$. D'autre part, ils ont amélioré le coût du rendez-vous déterministe dans un graphe arbitraire grâce à leur algorithme qui travaille en temps $O(\log^3 l + n^{15} \log^{12} n \log l)$. On constate bien que ce temps ne dépend pas du délai τ ce qui répond à la question posée par les auteurs de [10] à la fin de leur article.

Concernant les résultats présentés dans [10] et [15] pour résoudre le problème du rendez-vous dans le graphe, Pelc [23] ajoute une remarque très importante et qui a été aussi soulevée dans [10] : les algorithmes présentés utilisent un *ingrédient non-constructif*, c'est à dire un objet dont l'existence est prouvé d'une manière probabiliste. Malgré que l'exécution de ces algorithmes se fait d'une manière déterministe par les agents, ça n'empêche pas que le temps des calculs locaux pour obtenir un tel objet peut être énorme. Les auteurs ont contourné ce problème en utilisant un modèle qui utilise le nombre de rondes pour calculer le coût de l'exécution sans tenir compte des calculs locaux exécutés par les agents.

Une question qui s'impose est la suivante : existe-t-il un algorithme déterministe dont le coût est polynomial en n et $\log l$ et qui n'utilise aucun *ingrédient non-constructif* ?

Ta-Shma et Zwick [28] répondent positivement à cette question en fournissant une solution déterministe qui garantit le rendez-vous en temps $\tilde{O}(n^5 \log l)$ [1]. Ils ont utilisé le résultat prouvé par Reingold [24] qui a prouvé que le calcul de la *séquence d'exploration universelle* proposée par Kouckỳ [14] pour n'importe quel graphe connexe d'ordre n se fait en temps polynomial en n. De plus si les agents sont placés dans un graphe de degré borné d, les auteurs de [28] prouvent que leur solution permet aux agents de se rencontrer dans un temps au plus $\tilde{O}(d^2 n^5 \log l)$.

Jusqu'à maintenant, nous avons parlé du coût de réalisation du rendez-vous sans tenir compte de la capacité de la mémoire. Toutes les solutions proposées ci-dessus assument que les agents sont équipés d'une mémoire assez grande pour stocker toutes les informations nécessaires. Nous allons voir maintenant les résultats obtenus jusqu'à date pour l'optimisation du coût de la mémoire des agents pour la résolution du problème du rendez-vous.

Pour calculer le coût de la taille de la mémoire nécessaire pour réaliser un rendez-vous lorsque c'est possible, les agents sont présentés comme des copies identiques de la même machine abstraite à états (automate) [12]. Durant chaque ronde, un agent passe d'un état à un autre. Le modèle à utiliser dans ce cas reste le même que discuté ci-dessus. L'objectif est de déterminer à quel coût de mémoire cette paire d'agents [12] peut résoudre le problème du rendez-vous dans le cas d'un départ arbitraire. Lorsqu'on parle du coût de la mémoire, on fait référence au nombre d'états par lesquels l'agent (un automate) a passé [12]. Donc, n états nécessite $\Theta(\log n)$ bits de mémoire pour les coder.

Fraigniaud et Pelc [13] ont construit un algorithme qui utilise une taille de mémoire dans l'ordre de $\Theta(\log n)$ pour résoudre le problème de rendez-vous avec délai arbitraire pour n'importe quel arbre d'ordre n. Ce résultat est optimal puisque la borne inférieure sur le nombre de bits de mémoire pour le rendez-vous avec délai arbitraire dans les arbres est $\Omega(\log n)$.

Ils montrent aussi que dans le cas de départ simultané de la paire des agents, la taille de la mémoire dépend d'un autre paramètre, autre que le nombre de nœuds de l'arbre ; il s'agit du nombre l de ses feuilles. Pour un tel départ, ils montrent un algorithme qui utilise $O(\log l + \log \log n)$ bits de mémoire pour réaliser une rencontre dans n'importe quelle arbre d'ordre n et qui possède l feuilles. On peut bien constater la différence exponentielle du coût de mémoire pour une classe d'arbres dont le nombre de feuilles se trouve dans $O(\log n)$ entre le cas d'un départ arbitraire et d'un départ simultané.

1. $\tilde{O}$ signifie un O à un facteur polylogarithmique près.

Pour ce qui est de la borne inférieure, Fraigniaud et Pelc [13] montrent que pour un départ simultané, la paire des agents a besoin d'au moins $\Omega(\log \log n)$ bits de mémoire pour réaliser le rendez-vous dans une ligne de longueur $n-1$. D'autre part, les auteurs montrent qu'il existe une classe d'arbres de degré maximum 3 et avec l feuilles, pour laquelle l'accomplissement du rendez-vous nécessite au moins $\Omega(\log l)$ bits de mémoire. Donc la taille $O(\log l + \log \log n)$ de mémoire utilisée par leur algorithme est optimale.

Pour ce qui est de l'anneau, Czyzowicz, Kosowski et Pelc [8] prouvent que pour tout $n \geq 9^5$, une paire d'agents anonymes à besoin d'au moins $(\frac{1}{5}\log n - 1)$ bits de mémoire pour résoudre le problème du rendez-vous dans les anneaux avec n nœuds, même dans le cas du départ simultané. Dans le même article [8], les auteurs prouvent un résultat très important : la taille minimale de mémoire requise pour garantir le rendez-vous déterministe, quand c'est possible, est $\Theta(\log n)$, dans n'importe quel graphe à n nœuds, pour n'importe quel délai de démarrage τ des agents.

2.2.2 Le rendez-vous avec marquage des nœuds

Nous avons vu dans la section précédente que le problème majeure dans la résolution du problème du rendez-vous déterministe est celui de briser la symétrie. Dans cette section, nous allons voir un autre moyen qui permet de contourner ce problème afin de pouvoir réaliser la rencontre lorsque c'est possible. L'idée est d'utiliser des jetons.

Kranakis, Krizanc, Santoro et Sawchuk [21] présentent des algorithmes permettant à chaque agent mobile d'employer un seul jeton pour résoudre le problème du rendez-vous. Ces deux jetons ne peuvent être utilisés qu'une seule fois et ils sont identiques. Le premier résultat que les auteurs présentent est le suivant : les agents se trouvent sur des positions initiales dans un anneau anonyme possédant un nombre paire de nœuds. Chaque agent marque sa position initiale par son unique jeton. Si la distance qui sépare leurs positions initiales est $\frac{n}{2}$, alors il n'existe aucun algorithme déterministe qui permet de faire le rendez-vous. Les auteurs proposent ensuite une série d'algorithmes qui résolvent le problème du rendez-vous en tenant compte de la connaissance du nombre de nœuds n, de la distance d qui sépare les positions initiales des agents et de l'orientation de l'anneau dans le cas ou $d < \frac{n}{2}$ (voir le tableau récapitulatif dans [8]). D'autre part, les auteurs prouvent que si les agents sont équipés de mémoire dont la taille est d'ordre $O(1)$ et s'ils ne connaissent ni les valeurs de n et de d ni l'orientation de l'anneau, alors

il n'existe aucun algorithme déterministe qui permet aux agents de vérifier si $d = \frac{n}{2}$ ou non.

Sawchuk [25] étudie les agents équipés de jetons transportables. L'auteur présente un algorithme qui résout le problème du rendez-vous en temps $O(n^2)$ dans un anneau bidirectionnel lorsque les deux agents sont équipés d'une mémoire de taille constante et chacun d'eux possède un jeton transportable. L'autre résultat que présente Sawchuk [25] concerne l'anneau unidirectionnel : un algorithme qui permet de faire la rencontre en temps $O(n^2)$ quand les deux agents sont équipés d'une mémoire dans $O(1)$ et chacun d'eux possède 2 jetons transportables. De plus, cet algorithme permet aux agents de savoir si le rendez-vous est faisable ou non. Si les agents possèdent un nombre de jetons $t \geq 3$ et qui sont transportables alors le rendez-vous est faisable en temps $O(n^{\frac{t-1}{t-2}t})$ dans un anneau bidirectionnel et la taille de la mémoire est d'ordre $O(\log t)$; si $t = \log n$ alors l'algorithme permet le rendez-vous en temps $O(n \log n)$.

Un autre modèle étudié par Kranakis, Krizanc et Markou ([19], [18]) est celui du tore orienté. Deux agents sont placés dans un tore anonyme de dimension $n \times m$. Les auteurs présentent des solutions qui résolvent le problème du rendez-vous sans détection, noté **RV**, et celui avec détection, noté **RVD**. La notion de détection fait référence à ce que l'algorithme exécuté permet de vérifier si le rendez-vous est faisable ou non. Les résultats présentés par les auteurs [18] sont les suivants :

- Avec un jeton non-transportable et une mémoire de taille $O(\log n + \log m)$, le rendez-vous avec détection est faisable en temps $O(nm)$.
- Si chaque agent possède 2 jetons transportables et une mémoire de taille constante, alors les agents peuvent se rencontrer en temps $O(n^2 + m^2)$.
- Dans le cas de 3 jetons transportables et d'une mémoire constante, le rendez-vous avec détection est réalisable en temps $O(n^2 + m^2)$.

Pour ce qui est de la taille de la mémoire [18], deux agents ont besoin d'au moins $\Omega(\log n)$ bits de mémoire pour réaliser le rendez-vous si chacun d'eux possède un nombre constant de jetons non-transportables. Si chaque agent est équipé d'un seul jeton transportable, ils auront aussi besoin d'au moins $\Omega(\log n)$ bits de mémoire.

Nous terminons cette partie par la notion de perte de jeton étudiée par Flocchini, Kranakis, Krizanc, Luccio, Santoro, Sawchuk [11]. Les auteurs s'intéressent au rendez-vous réalisé par un nombre k d'agents, où $k \geq 2$. Les agents sont équipés chacun d'un seul jeton non-transportable et sont situés sur des positions différentes d'un anneau non-

orienté à n nœuds. Au plus $k-1$ jetons peuvent tomber en panne, c'est à dire qu'ils ne sont pas visibles pour n'importe quel agent. Les auteurs affirment les résultats suivants :

- Si tous les jetons tombent en panne ou $pgcd(h,n) \neq 1$ pour certains $h \leq k$, alors le rendez-vous n'est pas faisable.
- Si les agents ne connaissent pas les valeurs de n et de k, alors le rendez-vous n'est pas faisable.
- Si les agents connaissent la valeur de n ou de k et $pgcd(h,n) = 1$ pour tout $h \leq k$, alors le rendez-vous est réalisable en utilisant $O(k \log n)$ bits de mémoire.

2.3 Le rendez-vous synchrone aléatoire

Dans cette section, nous allons nous intéresser aux algorithmes aléatoires pour résoudre le problème du rendez-vous synchrone. Lorsqu'on parle du rendez-vous aléatoire, ceci veut dire que les agents utilisent la probabilité pour effectuer leurs déplacements dans le réseau. La meilleure référence est le livre de Alpern et Gal [3] dans lequel ils ont bâti les fondements théoriques du problème du rendez-vous. Les auteurs offrent des solutions probabilistes pour réaliser le rendez-vous dans des espaces géométriques et dans différentes structures de données. Les algorithmes aléatoires sont surtout utilisés pour résoudre les situations qui n'ont pas de solution dans le cas déterministe. Ces algorithmes utilisent en général soit le lancé de la monnaie (par exemple si c'est pile l'agent se déplace vers un nœud adjacent, si c'est face il reste immobile), soit la marche aléatoire.

La marche aléatoire [7] dans un graphe G non orienté et connexe est un parcours de G. Étant donnée un nœud de départ, le choix de déplacement vers un de ses voisins se fait d'une manière aléatoire et équiprobable. Plus formellement, une marche aléatoire est une chaîne de Markov qui a comme ensemble d'état les nœuds du graphe G et la probabilité p des transitions est définie comme suit : soient x et y deux nœuds qui appartiennent au graphe G. Si x et y ne sont pas adjacents alors $p(x,y) = 0$, sinon $p(x,y) = \frac{1}{d(x)}$ où $d(x)$ représente le degré du nœud x.

Pour ce qui est du modèle, on s'intéresse toujours à des graphes dont les nœuds ne sont pas étiquetés. En revanche, un agent peut identifier les numéros de ports associés à chaque nœud.

La plupart des auteurs s'intéressent à la valeur espérée du temps du rendez-vous aléatoire. Pour alléger le texte, nous allons utiliser le terme « temps » au lieu de « temps espéré ».

2.3.1 Le rendez-vous sans marquage des nœuds

Dans leur étude du cas de l'anneau à n nœuds, Kranakis and Krizanc [17] fournissent un algorithme aléatoire qui utilise la marche aléatoire et qui fonctionne en temps $O(n^2)$ pour deux agents qui se trouvent initialement à une distance d. Si cette distance est constante, $d \leq \frac{n}{2}$ et d et n sont paires, alors le rendez-vous peut se faire en temps linéaire de valeur $\frac{d}{2}(n-d)$.

Alpern, Baston et Essegaier [2] montrent que le rendez-vous synchrone se fait en temps $O(n)$ dans le cas du graphe complet K_n et de l'anneau d'ordre n. La stratégie qu'ils ont utilisée, et qui peut être appliquée à n'importe quel graphe d'ordre n, est la suivante : tous les nœuds de n'importe quel graphe peuvent être visités durant $2n$ rondes en maximum. Les nœuds sont anonymes. Supposons maintenant que dans chaque intervalle de temps de longueur $2n-1$, les agents se déplacent dans le graphe avec une probabilité de $\frac{1}{2}$ et restent immobile dans un nœud avec une probabilité de $\frac{1}{2}$. Donc, dans chaque intervalle de temps de longueur $2n-1$, la probabilité qu'un agent parcours le graphe et l'autre reste immobile est $\frac{1}{2}$. Par conséquent, les agents peuvent se rencontrer dans un temps qui ne dépasse pas $(1.\frac{1}{2}+2.\frac{1}{4}+3.\frac{1}{8}+...)(2n-1) = \frac{\frac{1}{2}}{(1-\frac{1}{2})^2}(2n-1) = 2(2n-1)$.

Pour améliorer le temps quadratique de leur algorithme précédent de rendez-vous dans l'anneau, Kranakis and Krizanc [17] s'inspirent de la stratégie du lancé de monnaie « coin half tour » proposée par Alpern [1]. L'agent lance la monnaie : si c'est *pile*, il choisit la direction selon les aiguilles de la montre et parcourt l'anneau dans cette direction durant $\frac{n}{2}$ rondes ; si c'est *face*, il parcourt l'anneau dans le sens contraire des aiguilles de la montre durant $\frac{n}{2}$ rondes. En utilisant ce principe, Kranakis and Krizanc [17] améliorent leur algorithme pour réaliser le rendez-vous entre les agents en n rondes. De plus, les auteurs montrent qu'avec cet algorithme, les agents ont besoin seulement d'une mémoire d'ordre $O(\log n)$ pour que la rencontre soit réalisable. Il est aussi évident de voir que la borne inférieure sur le temps du rendez-vous est $\Omega(n)$ puisque chaque agent a besoin de parcourir en pire cas au moins la moitié de l'anneau d'ordre n.

Kranakis, Krizanc et Morin [20] viennent ensuite pour diminuer la capacité de la mémoire utilisée dans l'algorithme de Kranakis et Krizanc [17] de $O(\log n)$ à $O(\log \log n)$. Le nouveau *algorithme de comptage approximatif* est une amélioration de l'algorithme de Kranakis et Krizanc [17] de la manière suivante : au lieu que l'agent parcourt l'anneau durant exactement $\frac{n}{2}$ rondes dans une direction après le lancé de la monnaie, il va le faire durant $O(n)$ rondes. Les auteurs prouvent en premier lieu que deux agents

qui sont équipés d'une mémoire de capacité k bits chacun et dont la distance initiale qui les séparent est inférieure ou égale à $\frac{n}{2}$ peuvent accomplir le rendez-vous en temps $O(\frac{n^2}{2^{2k}} + 2^{2^k})$. En deuxième lieu, ils montrent que tout algorithme synchrone qui permet à deux agents situés sur deux positions différentes d'un anneau non-orienté à n nœuds de réaliser le rendez-vous dans un temps $\Theta(n)$ requiert au moins $\Omega(\log \log n)$ bits de mémoire.

2.3.2 Le rendez-vous avec marquage des nœuds

Nous allons voir dans cette section, comment l'utilisation du marquage des nœuds avec des jetons aide beaucoup à résoudre le problème du rendez-vous synchrone dans le cas aléatoire.

Baston et Gal [6] présentent une solution du problème du rendez-vous en permettant à chaque agent de laisser son jeton dans sa position initiale. Chaque agent essaye de trouver le nœud de départ de l'autre en visitant aléatoirement les autres nœuds. Le nœud de départ va être trouvé quand l'agent va trouver l'autre jeton. Les auteurs ajoutent la condition que chaque agent doit retourner à sa position initiale après chaque 100 rondes lorsqu'il effectue la recherche des jetons. Une fois qu'un agent trouve le nœud de départ de l'autre, il commence à osciller entre son nœud de départ et celui de l'autre agent avec une probabilité de $\frac{1}{2}$. Baston et Gal [6] montrent qu'en utilisant cette stratégie dans un graphe complet d'ordre n, les agents peuvent se rencontrer dans un temps au plus $\frac{n}{3} + o(n)$.

Les auteurs de [6] étudient aussi le cas du cercle de circonférence 1. Ils affirment que le problème est difficile à analyser mais ils présentent une solution pour $\frac{1}{3} \leq d < \frac{1}{2}$ où d représente la distance qui séparent les deux positions initiales des agents. Cette distance est connue par les agents. La procédure de la rencontre est la suivante : les agents se déplacent dans le cercle à la même vitesse. À partir de leurs positions initiales qu'ils ont marquées avec leurs jetons, les agents choisissent d'une manière aléatoire une direction et se déplacent dans ce sens durant un temps d qu'ils peuvent calculer grâce à leur vitesse commune. Si les agents ne se rencontrent pas après ce temps d, ils utiliseront une autre stratégie : chaque agent revient sur son nœud de départ puis se déplace à une distance d dans le sens des aiguilles de la montre. Si un agent rencontre le jeton de l'autre, il continue son parcours dans le même sens ; sinon, il parcourt l'anneau dans le sens inverse. Le rendez-vous se fait dans un temps égal à $\frac{1+6d}{8}$.

Dans sa thèse, Sawchuk [25] élimine les conditions ajoutées pour la résolution du problème du rendez-vous dans [6] et présente un algorithme qui réalise le rendez-vous en temps linéaire dans un anneau à n nœuds. Ceci veut dire que les agents n'ont pas besoin de connaître ni n, ni d, ni l'orientation de l'anneau et l'algorithme fournit par Sawchuk [25] permet aux agents de faire le rendez-vous en temps $O(n)$ si chacun d'eux est équipé d'une mémoire de capacité constante.

2.4 Le rendez-vous déterministe asynchrone

Dans tout ce qu'on a vu jusqu'à maintenant, les agents essayent de réaliser le rendez-vous d'une manière synchrone. Ci-dessous, nous allons parler d'une autre façon par laquelle les agents peuvent faire une rencontre : c'est le scénario asynchrone. Le modèle utilisé pour essayer d'accomplir le rendez-vous est le suivant [22] : Les agents ont des identifiants différents et chacun d'eux connaît seulement son propre identifiant. Les nœuds du réseaux ne sont pas identifiés mais les agents peuvent lire les numéros de ports associés à chaque nœud. Le rendez-vous des agents peut se faire, pas seulement dans un nœud, mais aussi à l'intérieur d'une arête si les agents se croisent le chemin ; la chose qui n'était pas offerte dans le cas synchrone.

À chaque ronde, l'agent choisit une arête incidente à sa position courante et traverse cette arête avec une vitesse arbitraire possiblement variable. Un rendez-vous asynchrone ne peut se faire que si les agents se rencontrent au même instant à la même place qui peut être soit dans un nœud soit à l'intérieur d'une arête. On s'intéresse aux algorithmes qui garantissent le rendez-vous pour n'importe quelles vitesses des agents. Le coût d'un algorithme de rendez-vous c'est le nombre d'arêtes parcourues par les agents.

De Marco, Gargano, Kranakis, Krizanc, Pelc et Vaccaro [22] commencent par résoudre le problème du rendez-vous asynchrone dans une ligne de longueur infinie. Soient D la distance qui sépare les positions initiales des agents et L_{max} et L_{min} représentent les identifiants des agents tels que $|L_{max}| < |L_{min}|$. Si les agents connaissent D, alors le rendez-vous est réalisable à un coût d'ordre $O(D|L_{min}|^2)$; si les agents ne connaissent pas la valeur de cette distance, alors la rencontre peut se faire à un coût d'ordre $O((D + |L_{max}|)^3)$. Ce coût a été ensuite amélioré par l'algorithme de Stachowiak [27] qui permet de réaliser la rencontre à un coût d'ordre $O(D \log^2 D + D \log D|L_{max}| + D|L_{min}|^2 + |L_{max}||L_{min}| \log |L_{min}|)$ lorsque les agents ne connaissent pas D.

Dans le cas d'un anneau à n nœuds, les auteurs de [22] présentent un algorithme qui permet aux agents, sans connaître la valeur de D, de faire le rendez-vous à un coût d'ordre $O(n|L_{max}|)$ s'ils ne connaissent pas n, et de le faire à un coût d'ordre $O(n|L_{min}|)$ lorsque les agents connaissent n. Les auteurs démontrent que ce coût est optimal dans ce dernier cas.

Les auteurs de [22] prouvent ensuite un résultat très important : deux agents peuvent réaliser un rendez-vous asynchrone dans n'importe quel graphe s'ils connaissent une borne supérieure sur l'ordre du graphe. Dans ce cas leur algorithme n'est pas efficace. D'autre part, si les agents connaissent la taille n du graphe et la valeur D de la distance initiale entre les agents, alors ils peuvent accomplir le rendez-vous asynchrone à un coût d'ordre $\Theta(D|L_{min}|)$.

Quatre ans plus tard, Czyzowicz, Labourel et Pelc [9] fournissent un algorithme qui permet de faire le rendez-vous asynchrone dans n'importe quel graphe connexe d'ordre n fini ou dans un graphe infini. Nous voyons bien que ce résultat est très fort puisqu'il permet de faire le rendez-vous non seulement dans un graphe sans la connaissance d'une borne supérieure de n, mais aussi dans n'importe quel graphe qui possède un nombre infini et dénombrable de nœuds. Malgré l'importance de cette solution, elle ne nous offre pas le coût de l'algorithme ; c'est vrai que le rendez-vous est réalisable mais à quel prix ?

Une réponse est proposée par Bampas, Czyzowicz, Gasieniec, Ilcinkas et Labourel [5] dans le cas du rendez-vous asynchrone dans une grille infinie dans un espace de dimension δ. Les auteurs utilisent un modèle beaucoup plus fort : chaque agent connaît les coordonnés de sa position initiale par rapport à un système commun de coordonnées. Dans ce type de structure, deux agents peuvent se rencontrer à un coût d'ordre $O(D^\delta \log^{\delta^2+\delta+1} D)$. Pour ce qui est de la borne inférieure, les agents ne peuvent pas accomplir le rendez-vous à un coût meilleur que $\Omega(D^\delta)$ pour n'importe quel algorithme déterministe asynchrone.

Chapitre 3

Le modèle

3.1 La symétrie

Supposons que deux agents sont situés sur deux nœuds différents liés par une arête. Les ports à ses deux extrémités portent le même numéro. Dans ce simple cas, on dit que les deux agents sont situés sur des positions symétriques. Il est évident de constater que dans cette situation et si les agents commencent leurs déplacements au même moment, la rencontre est impossible pour n'importe quel algorithme synchrone et déterministe puisque les agents vont soit rester immobile sur leurs positions, soit qu'ils vont toujours se croiser le chemin. Par conséquent, le grand défi pour accomplir le rendez-vous revient à casser *la symétrie*.

Définition 3.1.1 *Un arbre $T = (V, E)$ est un graphe connexe sans cycles.*

Soient $T = (V, E)$ et $T' = (V', E')$ des arbres tels que V et V' forment respectivement l'ensemble des nœuds appartenant à T et T' et E et E' sont les ensembles des arêtes. Nous considérons les arbres dont chaque arête $\{v, w\}$ a deux étiquettes p_v et q_w appelés ports à v et à w, respectivement. Pour un nœud v, tous les ports à v sont des entiers différents dans l'ensemble $\{0, 1, ..., d-1\}$, où d est le degré de v.

Définition 3.1.2 *Un isomorphisme entre deux arbres T et T' est une fonction $f : V \to V'$ telle que :*

1. *f est bijective ;*
2. *$\forall v, w \in V$, v est adjacent à w si et seulement si $f(v)$ est adjacent à $f(w)$;*

3. *$\forall v, w \in V$, si $e = \{v, w\}$ est une arête de l'arbre et p et q sont respectivement les ports associés à v et w correspondant à l'arête e, alors p et q sont respectivement les ports à $f(v)$ et $f(w)$ correspondant à l'arête $\{f(v), f(w)\}$.*

Dans le cas où $T = T'$, on dit que f est un *automorphisme*. Deux nœuds distincts u et v d'un arbre T sont *symétriques* s'il existe un automorphisme f de l'arbre tel que $f(u) = f(v)$.

Définition 3.1.3 *Un arbre enraciné est un arbre dont un des nœuds u est distingué comme la « racine ». Nous disons que l'arbre est enraciné en u.*

Soient maintenant $T = (V, E)$ et $T' = (V', E')$ deux arbres distincts enracinés respectivement en $u \in V$ et en $u' \in V'$.

Définition 3.1.4 *T et T' sont dits symétriques s'il existe un isomorphisme $f : V \to V'$ tel que $f(u) = f(u')$, où u est la racine de T et u' est la racine de T'.*

3.2 Restriction sur les agents et sur leurs connaissances

Dans ce mémoire, nous considérons des agents identiques, c'est à dire qu'ils utilisent le même algorithme et ils n'ont pas d'identifiants différents pour les différencier. Plus précisément, nous supposons que les agents sont des copies A et A' du même automate A ; c'est pour cela qu'on va les nommer une *paire d'agents* à cause qu'ils sont identiques et font référence à la même machine d'états abstraite. A et A' sont situés initialement sur des nœuds différents v_A et $v_{A'}$ qu'on appelle *les positions initiales*.

Dans la pratique et pour des raisons de sécurité et de confidentialité, les agents n'ont pas la possibilité de lire ou de connaître les identifiants des nœuds qu'ils visitent ; pour cette raison, nous nous intéressons aux arbres dont les nœuds ne sont pas étiquetés. De plus, il n'existe aucun moyen de communication entre les agents. ces derniers peuvent seulement lire les numéros de ports de chaque nœud de l'arbre.

3.3 La synchronisation et le moment initial d'exécution

Afin d'accomplir le rendez-vous, une paire d'agents exécute le même algorithme en utilisant des *horloges* identiques pour se déplacer et calculer le temps. Chaque horloge

fonctionne en rondes et n'est active que lorsque son propriétaire commence l'exécution de son algorithme. Durant une ronde, un agent peut changer de position vers un nœud adjacent ou rester sans action sur sa position actuelle.

On note par θ la différence de temps entre les premiers instants durant lesquels chaque agent commence l'exécution de l'algorithme du rendez-vous.

Nous construisons des algorithmes synchrones qui fonctionnent dans les deux cas possibles suivants : le premier est celui où les deux agents commencent leurs déplacements au même moment, c'est à dire que $\theta = 0$, on parle ici d'un *départ simultané* des agents. Le deuxième cas est celui où ces derniers commencent l'exécution de l'algorithme du rendez-vous à des instants différents, c'est à dire lorsque $\theta \neq 0$; dans cette situation, on parle d'un *départ arbitraire* des agents. Ces derniers ne connaissent pas la valeur de θ puisqu'aucun d'eux ne connaît le moment du départ de l'autre.

Nous assumons que les agents ne sont placés sur leurs nœuds de départ qu'au commencement de leur exécution de l'algorithme. De cette manière, nous éliminons le cas où l'un des agents peut rencontrer l'autre avant que ce dernier ne commence sa première ronde.

3.4 L'accomplissement du rendez-vous et son coût

Avant de discuter du coût du rendez-vous, il faut en premier lieu parler de son accomplissement. Nous avons parlé dans ce qui précède et à plusieurs reprises des positions initiales des agents. Ces positions jouent un rôle crucial dans la réalisation du rendez-vous déterministe. Si les positions initiales sont symétriques et le départ des agents est simultané ($\theta = 0$), alors le rendez-vous déterministe est impossible pour n'importe quelle paire d'agents. Par contre, si ces positions ne sont pas symétriques ou si le départ n'est pas simultané, alors il existe une paire d'agents qui en exécutant un algorithme déterministe, peut se rencontrer dans un arbre quelconque.

Définition 3.4.1 *On dit qu'une paire d'agents résout le problème du rendez-vous en utilisant un algorithme déterministe pour une classe d'arbres, si pour tout arbre appartenant à cette classe, les deux agents se rencontrent dans le même nœud sauf dans le cas spécial où les positions initiales sont symétriques et le départ des agents est simultané.*

Dans le cas aléatoire, on dit qu'une paire d'agents résout le problème du rendez-vous en exécutant un algorithme aléatoire pour une classe d'arbres d'ordre n, si la probabilité

pour que les deux agents accomplissent le rendez-vous dans le même nœud durant la même ronde est au moins $1 - 1/n$ *indépendamment de leurs positions initiales.*

Nous nous intéressons au pire cas pour évaluer le coût de rendez-vous, c'est à dire qu'il faut toujours penser à une situation qui va rendre la rencontre la plus compliquée. Le deuxième agent est celui qui commence l'exécution de l'algorithme plus tard. Dans le cas du départ simultané, le deuxième agent est n'importe quel des deux.

Définition 3.4.2 *Pour des positions initiales des agents données et pour un délai* θ *donné, le coût d'un algorithme de rendez-vous est le nombre de rondes écoulées entre le démarrage du deuxième agent et la rencontre des agents. Ce coût en pire cas est le suprémum des coûts pour toutes les positions initiales et tous les délais.*

3.5 Terminologie

Dans tout ce mémoire, nous utilisons la terminologie suivante :

- Nous nommerons le cas où les agents commencent leurs déplacements en même temps par le cas de *départ simultané* et par le cas de *départ arbitraire* lorsque l'un des deux commence son déplacement avant l'autre.
- A_1 et A_2 font références à l'agent 1 et l'agent 2 respectivement ; dans le cas du *départ arbitraire*, A_1 désigne l'agent qui commence le déplacement en premier (aucun agent ne sait si c'est lui le premier ou le dernier à commencer l'exécution de l'algorithme). Dans ce cas, on note par θ l'écart de temps entre les départs des agents, ce qui veut dire que A_2 commencent sa première action θ rondes après A_1.
- n, Δ et D représentent respectivement le nombre de nœuds d'un arbre, le degré maximum d'un nœud et la distance entre les positions initiales des agents.
- Les nœuds de l'arbre ne sont pas étiquetés mais les ports portent des numéros allant de 0 jusqu'à $d - 1$ pour chaque nœud de degré d, où $d \leq \Delta$.

3.6 Énoncé du problème

Étant donné une paire d'agents situés dans deux positions différentes d'un arbre :

1. Est ce que le rendez-vous déterministe est toujours possible dans l'arbre malgré que les agents sont identiques ? Qu'en est-il du cas aléatoire ?

2. Quel est l'impact de la symétrie de l'arbre et celle des positions initiales du départ des agents sur l'accomplissement du rendez-vous ?
3. Quel est le coût optimal pour faire le rendez-vous déterministe et le rendez-vous aléatoire ?
4. Que rapportera l'étude des algorithmes aléatoires et presque certains à l'amélioration du coût et à la réalisation du rendez-vous, comparé au scénario déterministe ?

Chapitre 4

Les résultats de la recherche et la méthodologie

4.1 Les résultats de la recherche

Nous abordons en premier lieu le problème du rendez-vous dans la ligne. Nous construisons un algorithme déterministe qui permet aux agents, dans tous les cas où c'est possible, de faire le rendez-vous en temps $O(n)$, où $n-1$ est la longueur de la ligne. Nous construisons ensuite un algorithme déterministe qui accomplit le rendez-vous en temps $O(n)$ dans les arbres à n nœuds, chaque fois que le rendez-vous est possible, et nous montrons que ce temps ne peut pas être améliorée en général, même si les agents se situent initialement à une distance égale à 1 dans les arbres de degré borné.

Nous présentons aussi deux algorithmes aléatoires presque certains qui permettent respectivement de faire le rendez-vous en temps $O(n)$ dans une ligne de longueur $n-1$ et dans un arbre d'ordre n, pour des positions initiales arbitraires des agents. Nous verrons ensuite qu'on peut améliorer notre algorithme aléatoire pour qu'il devient plus rapide en temps d'exécution. Nous construisons un algorithme presque certain qui permet d'accomplir le rendez-vous en temps $O(\log n)$ lorsque les agents savent à l'avance que la distance qui sépare leurs positions initiales est constante, que le degré maximum de l'arbre est borné et ils connaissent ces deux constantes. En revanche, nous montrons que pour certains arbres, il n'existe aucun algorithme aléatoire presque certain qui peut accomplir le rendez-vous en un temps meilleur que $\Omega(n)$ même si les agents se trouvent initialement à une distance égale à 1.

Toutes les bornes supérieures présentés dans ce mémoire sont valides même si les agents commencent l'exécution de l'algorithme du rendez-vous à des moments différents t_1 et t_2 et toutes nos bornes inférieures sont valides même lorsque les agents démarrent simultanément ($t_1 = t_2$).

L'article S. Elouasbi, A. Pelc, *Time of anonymous rendezvous in trees : Determinism vs. randomization*, contenant les résultats de ce mémoire a été accepté à la *19th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2012), June 2012, Reykjavik, Iceland.*

4.2 Méthodologie

Tous les résultats présentés sont démontrés par des preuves rigoureuses qui utilisent des méthodes combinatoires et des éléments de la théorie des graphes. Pour le cas des algorithmes aléatoires, nous avons utilisé des méthodes probabilistes. Les bornes théoriques dans le cas probabiliste seront testées à l'aide des simulations.

Chapitre 5

Rendez-vous déterministe dans la ligne

5.1 Introduction et définitions

Dans ce chapitre nous considérons le problème du rendez-vous dans une ligne de longueur $n-1$, où n représente le nombre de nœuds appartenant à cette ligne. Voici la situation : deux agents mobiles A_1 et A_2 sont situés sur deux nœuds différents d'une ligne de longueur $n-1$. Chaque agent sait que le rendez-vous va se faire dans une ligne, mais aucun d'eux n'a une idée ni de la longueur de la ligne ni des directions (droite ou gauche) ni de quel côté se trouve l'autre agent. À part les nœuds des deux extrémités qui ont chacun un port qui porte le numéro 0, le reste des $n-2$ nœuds ont chacun deux ports qui portent les numéros 0 et 1.

Les agents possèdent une mémoire assez grande pour stocker toutes les informations dont ils ont besoin. Ils vont parcourir la ligne afin de construire une carte de cette dernière, ce qui va leur permettre de trouver un « focal point » [26] pour réaliser leur rendez-vous quand c'est possible.

La ligne possède une particularité très importante que les agents peuvent exploiter pour se rencontrer : son *centre*. Si la ligne a un nombre paire d'arêtes, alors son centre est le noeud qui se trouve dans son milieu. Si elle possède un nombre impaire d'arêtes, dans ce cas son centre va être l'arête qui se trouve au milieu de la ligne. Dans le premier cas, on fait référence à un *noeud central*, et dans le deuxième cas on parle d'une *arête centrale*.

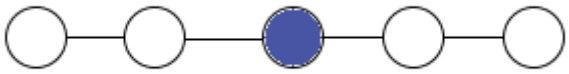

FIGURE 5.1 – Une ligne avec un nœud central

FIGURE 5.2 – Une ligne avec une arête centrale

Définition 5.1.1 *Un code de ligne est la suite de longueur* $2(n-1)$ *dont les termes sont les numéros de ports consécutifs rencontrés pendant le parcours de la ligne débutant dans une extrémité et se terminant à l'autre.*

Selon cette définition, il est évident que chaque ligne possède deux codes de ligne commençant dans ses deux extrémités. Dans tout le reste de ce chapitre, nous allons utiliser la notation suivante : C_1 est le code de ligne commençant dans l'extrémité E_1 et C_2 est le code de ligne commençant dans l'extrémité E_2. Ces deux codes peuvent être comparés en utilisant l'ordre lexicographique. Si $C_1 = C_2$ alors la ligne est *symétrique* ; sinon, elle ne l'est pas. L'utilisation de ce code va nous aider à casser la symétrie de la ligne pour permettre aux agents de faire le rendez-vous.

Proposition 5.1.1 *Si la ligne est symétrique et les positions initiales des agents sont symétriques, alors le rendez-vous est impossible dans le cas du départ simultané.*

Preuve : Considérons deux agents qui partent simultanément des positions symétriques. Puisque les agents exécutent le même algorithme, ils quittent leurs positions initiales en utilisant le même numéro de port. Par induction sur le numéro de la ronde, on montre que dans chaque ronde leurs positions restent toujours symétriques ; donc ils ne seront jamais sur le même nœud durant la même ronde. □

Nous allons maintenant introduire une procédure qui va nous aider à annoncer notre algorithme du rendez-vous déterministe dans la ligne à n nœuds. Cette procédure permet

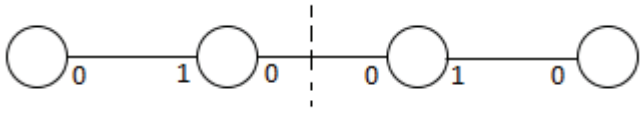

FIGURE 5.3 – Exemple d'une ligne symétrique

aux agents de connaître la longueur de la ligne et produire ses codes.

Procédure parcourir-ligne

Quitter sa position initiale par le port 0 ;

Aller jusqu'à l'extrémité en enregistrant les numéros de ports ;

Aller jusqu'à l'autre extrémité en enregistrant les numéros de ports ;

Retourner à sa position initiale ;

Produire le code de ligne débutant dans chaque extrémité.

5.2 Énoncé de l'algorithme

Avant d'énoncer l'algorithme **RV-déterministe-ligne**, nous allons présenter une idée générale de son fonctionnement. Chaque agent fait le parcours de la ligne à partir de sa position initiale afin de construire une ligne semblable grâce à sa mémoire. Dans cette copie qu'il vient de créer, l'agent doit identifier sa position initiale. Si la ligne n'est pas symétrique, alors les agents vont se rendre à un nœud unique et y rester pour accomplir la rencontre.

Si la ligne est symétrique, alors elle ne peut être que de longueur impaire et dans ce cas elle possède une arête centrale. Dans cette situation, chaque agent va générer une étiquette x qui va dépendre de sa position initiale pour aller ensuite se positionner sur l'extrémité la plus proche de cette arête. Une fois arrivé, chacun des deux agents va rester immobile à la ronde suivante. Ensuite chaque agent va répéter jusqu'au rendez-vous la boucle « rester inactif pendant x rondes et parcourir l'arête centrale x fois ».

Cet algorithme garantit le rendez-vous sauf dans le cas où la ligne est symétrique, les agents se trouvent sur deux positions initiales symétriques et le départ est simultané.

Tel que démontré dans la proposition 5.1.1, dans ce cas exceptionnel le rendez-vous est impossible et il n'existe aucun moyen pour briser cette symétrie.

Algorithme RV-déterministe-ligne

Exécuter la **Procédure parcourir-ligne**.

Si la ligne est de longueur paire **alors** rejoindre son nœud central et y rester.

Si la ligne est de longueur impaire mais elle n'est pas symétrique **alors**

se placer sur l'extrémité de la ligne pour laquelle le code de ligne est le plus grand selon l'ordre lexicographique.

Si la ligne est symétrique **alors**

$x :=$ la plus petite des deux distances qui séparent la position initiale de l'agent des deux extrémités de la ligne ;

rejoindre l'extrémité la plus proche de l'arête centrale ;

rester immobile à la ronde suivante ;

répéter jusqu'à la réalisation du rendez-vous

rester immobile durant x rondes ;

traverser l'arête centrale x fois.

5.3 Temps du rendez-vous

Théorème 5.3.1 *L'algorithme RV-déterministe-ligne résout le problème du rendez-vous dans n'importe quelle ligne de longueur* $n-1$ *en temps* $O(n)$.

Preuve : Nous allons tout d'abord montrer que l'algorithme est correct. Après l'exécution de la procédure **parcourir-ligne**, chaque agent doit connaître dans quelle situation il se trouve parmi celles listées dans cet algorithme. Si la longueur de la ligne est paire, ça entraîne forcément qu'elle possède un unique nœud central et les deux agents vont le rejoindre pour y accomplir le rendez-vous. Si la longueur de la ligne est impaire mais cette dernière n'est pas symétrique, alors $C_1 \neq C_2$. Dans ce cas, les agents achèvent le rendez-vous dans l'extrémité de la ligne pour laquelle le code de ligne est le plus grand lexicographiquement.

Il nous reste maintenant à étudier le cas où la ligne est symétrique. Si elle l'est, forcément elle est de longueur impaire et dans ce cas elle possède une arête centrale. L'exécution de la procédure **parcourir-ligne** permet à chaque agent de calculer les distances qui le séparent des deux extrémités de la ligne. Peu importe sa position initiale, l'agent va constater que ces distances sont différentes ; ceci est dû à la parité de la longueur de la ligne parce que s'il existe une position pour laquelle ces deux distances sont égales, alors la ligne possède un noeud central et ceci contredit le fait que la longueur de la ligne est impaire. Chaque agent choisit la distance la plus petite, notée x, et se dirige vers l'extrémité de l'arête centrale la plus proche de sa position initiale. Supposons d'abord que les positions initiales des deux agents ne sont pas symétriques. Alors les entiers x associés à chaque agent ne peuvent pas être égaux. On les note par x_1 et x_2 et on suppose, sans perte de généralité, que $x_1 > x_2$. Soit A l'agent qui a commencé plus tard l'exécution de la boucle « répéter jusqu'à la réalisation du rendez-vous » à l'instant t (Si les deux agents ont commencé cette exécution à la même ronde, A est choisi arbitrairement). À l'instant au plus $t+2x_1$, il va y avoir une ronde durant laquelle l'un des agents traverse l'arête centrale et l'autre reste passif à l'une des extrémités de cette dernière. Le rendez-vous aura donc lieu au plus tard durant cette ronde. Supposons maintenant que la ligne est symétrique et les agents se trouvent initialement sur des positions symétriques mais le délai est $\theta > 0$; dans ce cas $x_1 = x_2$. Soit A défini comme auparavant. Une fois arrivé à l'extrémité de l'arête centrale, A va rester inactif pendant $x_1 + 1$ rondes et donc la rencontre va se faire au plus tard à la ronde $t + 1 + x_1$.

Calculons maintenant le temps d'exécution de l'algorithme RV-déterministe-ligne. Selon la définition 3.4.2, nous devons commencer le calcul de ce temps à partir du moment où l'agent qui débute plus tard a lancé sa première action. L'exécution de la procédure **parcourir-ligne** prend un temps $2(n-1)$. Le temps pour rejoindre le nœud central ou l'une des extrémités de l'arête centrale est au plus $\lfloor \frac{n-1}{2} \rfloor$. Enfin, le temps d'exécution de la boucle « répéter jusqu'à la réalisation du rendez-vous » est au plus $2x_1 \leq 2n$. D'où le temps pour accomplir le rendez-vous appartient à $O(n)$. □

5.4 Conclusion

Dans ce chapitre, nous avons montré que le rendez-vous est faisable dans n'importe quelle ligne de longueur $n-1$. Nous avons présenté un algorithme déterministe qui

fonctionne en temps $O(n)$. Nous avons utilisé deux approches : la première est de trouver le nœud pour faire le rendez-vous dans le cas d'une ligne asymétrique, et la deuxième est d'attribuer des étiquettes différentes aux agents afin de les distinguer et d'accomplir le rendez-vous dans le cas de la ligne symétrique. Nous avons montré que le rendez-vous déterministe est impossible quand les positions initiales sont symétriques et le départ est simultané et qu'il est réalisable dans tous les autres cas. Dans le chapitre qui suit, nous allons généraliser ce résultat des arbres arbitraires.

Chapitre 6

Rendez-vous déterministe dans l'arbre

6.1 Introduction et définitions

Dans ce chapitre, nous assumons que les agents savent qu'ils doivent essayer de faire le rendez-vous dans un arbre, mais ils n'ont aucune information ni sur la topologie de l'arbre ni sur sa taille. Une des particularités que possède un arbre est son *centre* ; pour le trouver, il suffit d'appliquer l'algorithme suivant pour n'importe quel arbre T :

1. Enlever toutes les feuilles de l'arbre T et soit T_1 l'arbre qui reste ;
2. Répéter : Enlever toutes les feuilles de l'arbre T_i et soit T_{i+1} l'arbre qui reste jusqu'à l'obtention d'un arbre avec au plus deux nœuds ;
3. Si on obtient à la fin un arbre avec un seul nœud, on dit alors que l'arbre T possède un *nœud central* ; c'est le noeud qui reste.
4. Si on obtient à la fin un arbre avec deux nœuds, on dit alors que l'arbre T possède une *arête centrale* ; c'est l'arête qui uni les deux noeuds qui restent.

Présentons maintenant quelques définitions qui vont nous permettre de construire notre algorithme.

Définition 6.1.1 *Le tour de base est une stratégie de parcours d'arbre qui consiste en ce qui suit :*

1. *L'agent quitte son nœud de départ v par le port numéro 0 pour aller visiter un nœud adjacent w de degré d ;*
2. *Si le port par lequel l'agent est entré dans le nœud w est i, alors il le quitte par le port numéro $(i+1) \mod d$;*

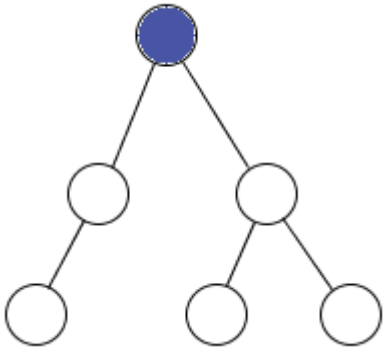

FIGURE 6.1 – Exemple d'un arbre avec un nœud central

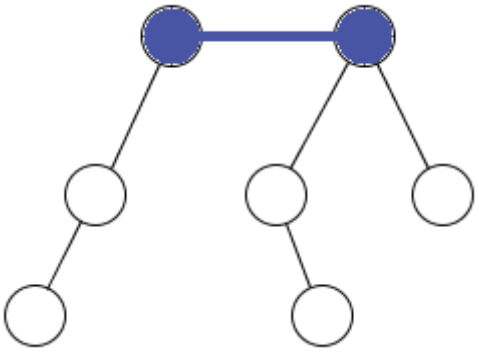

FIGURE 6.2 – Exemple d'un arbre avec une arête centrale

3. *Les étapes 2 et 3 sont répétées jusqu'à ce que l'agent traverse toutes les arêtes et revienne au nœud de départ.*

Définition 6.1.2 *On définit le code de l'arbre enraciné avec m nœuds comme la suite de longueur $2(m-1)$ dont les termes sont les numéros de ports consécutifs rencontrés pendant le tour de base débutant dans la racine. Le port* 0 *à chaque feuille est noté deux fois : à l'arrivée et à la sortie.*

6.2 La symétrie dans l'arbre

Définition 6.2.1 *Un arbre T qui possède des ports numérotés est dit symétrique s'il existe un automorphisme f de l'arbre différent de l'identité* ($pour un certain u, on a $f(u) \neq u$).

Tout arbre T qui possède une arête centrale peut être partitionné en deux sous-arbres T_1 et T_2 enracinés dans les deux extrémités de cette arête. Notons par C_1 et C_2 les codes des arbres enracinés T_1 et T_2 respectivement. Ces codes peuvent être comparés en utilisant l'ordre lexicographique. Ces derniers vont permettre aux agents de briser la symétrie de l'arbre afin d'accomplir le rendez-vous.

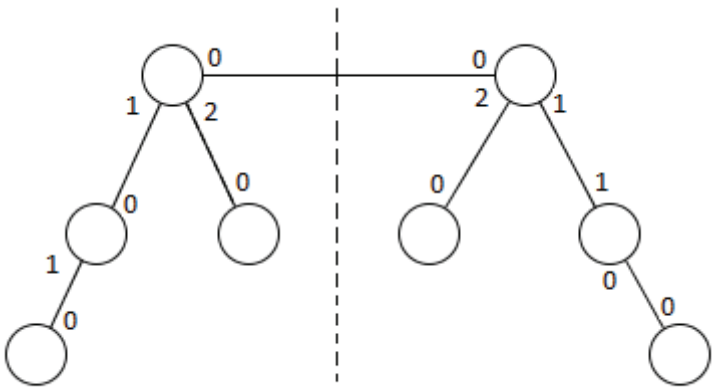

FIGURE 6.3 – Exemple d'un arbre qui n'est pas symétrique

Théorème 6.2.1 *L'arbre avec l'arête centrale $\{v, w\}$ est symétrique si et seulement si les deux ports de l'arête centrale sont identiques et $C_1 = C_2$.*

Preuve :
($\Rightarrow$) Supposons que l'arbre est symétrique. Considérons l'automorphisme f de l'arbre autre que l'identité. L'image par f de l'arête centrale est elle-même, donc $f(u) = v$ te les deux ports de l'arête centrale sont identiques.
Supposons que $C_1 \neq C_2$. Soient $C_1 = (p_1, p_2, \ldots, p_s)$ et $C_1 = (p'_1, p'_2, \ldots, p'_t)$. Soit j le premier indice tel que $p_j \neq p'_j$. Il y a des arêtes $e = \{w_1, w_2\}$ et $e = \{w'_1, w'_2\}$ telles que $f(w_1) = w'_1$, $f(w_2) = w'_2$ et le port correspondant à e au nœud w_1 et le port correspondant à e' au nœud w'_1 est p_{j-1} ; mais, le port correspondant à e au nœud w_2 est p_j et le

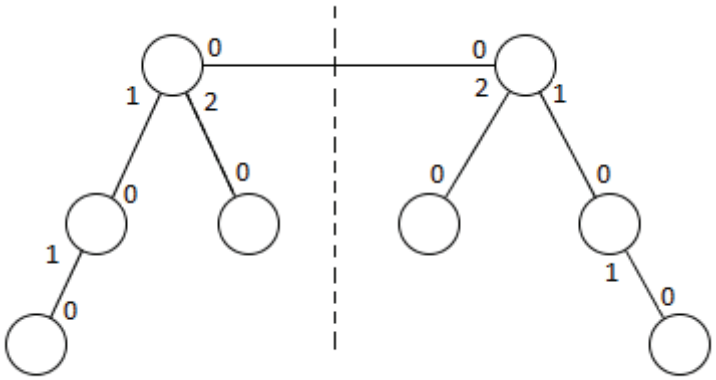

FIGURE 6.4 – Exemple d'un arbre symétrique

port correspondant à e' au nœud w'_2 est p'_j. Or $p_j \neq p'_j$, ce qui contredit le fait que f est un automorphisme. Donc $C_1 = C_2$.

($\Leftarrow$) Supposons que les deux ports de l'arête centrale de l'arbre T sont identiques et que $C_1 = C_2$. À l'état initial, deux agents A_1 et A_2 sont situés sur les extrémités de l'arête centrale qui lie les racines de T_1 et de T_2.

Les agents effectuent le tour de base de T_1 et de T_2 respectivement.Soient v_k et v'_k les noeuds des arbres T_1 et de T_2 visités par A_1 et A_2 dans la k-ème ronde. La fonction $f : V_1 \to V_2$ donnée par $f(v_k) = v'_k$ est un isomorphisme tel que $f(u) = v$. Donc l'arbre f est différent de l'identité. □

Proposition 6.2.1 *Si l'arbre est symétrique et les positions initiales des agents sont symétriques, alors le rendez-vous est impossible dans le cas du départ simultané.*

Preuve : Considérons deux agents qui partent simultanément des positions symétriques. Puisque les agents exécutent le même algorithme, ils quittent leurs positions initiales en utilisant le même numéro de port. Par induction sur le numéro de la ronde, on prouve que dans chaque ronde leurs positions sont symétriques ; donc ils ne seront jamais sur le même nœud durant la même ronde. □

6.3 Énoncé de l'algorithme

Avant d'énoncer l'algorithme **RV-déterministe**, nous allons présenter une introduction sur la motivation de son construction. Chaque agent effectue le tour de base à partir de sa position initiale. Une fois terminé, l'agent va avoir dans sa mémoire une carte isomorphe de l'arbre qu'il vient de parcourir. Dans cette carte, sa position initiale est bien identifiée. Si l'arbre n'est pas symétrique, alors l'agent se rend à un nœud particulier (le même pour les deux agents) et s'arrête.

Si l'arbre a une arête centrale et est symétrique, chaque agent calcule une étiquette x qui est liée à sa position initiale et rejoint par la suite l'extrémité la plus proche de cette arête. Une fois arrivé, chaque agent doit rester immobile à la prochaine ronde. Ensuite, chacun d'eux répète jusqu'au rendez-vous la boucle : « rester inactif durant x rondes et parcourir l'arête centrale x fois », ce qui permet de casser la symétrie de l'arbre pour effectuer le rendez-vous. Cet algorithme fonctionne toujours, sauf dans le cas spécial où les agents se trouvent dans des positions symétriques dans un arbre symétrique et le départ est simultané. Tel que ça était prouvé dans la proposition 6.2.1, le rendez-vous est impossible dans ce cas spécial. Voici le pseudo-code de l'algorithme.

Algorithme RV-déterministe

Effectuer le tour de base à partir de la position initiale.

Si l'arbre a un nœud central **alors** le rejoindre et y rester.

Si l'arbre a une arête centrale dont les numéros de port sont différents **alors**

se placer sur l'extrémité qui possède le plus grand numéro de port.

Si l'arbre a une arête centrale dont les numéros de port sont égaux, mais l'arbre n'est pas symétrique **alors**

se placer sur l'extrémité qui est la racine de l'arborescence dont le code est le plus grand selon l'ordre lexicographique.

Si l'arbre est symétrique **alors**

$x :=$ le nombre de pas du tour de base effectué à partir de la position initiale jusqu'à la première traversée de l'arête centrale ;

rejoindre l'extrémité la plus proche de l'arête centrale ;

rester inactif à la prochaine ronde ;

répéter jusqu'à la réalisation du rendez-vous

rester immobile durant x rondes ;

traverser l'arête centrale x fois.

6.4 Temps du rendez-vous

Théorème 6.4.1 *L'algorithme RV-déterministe résout le problème du rendez-vous dans n'importe quel arbre d'ordre n en temps $O(n)$.*

Preuve : Nous allons tout d'abord montrer que l'algorithme est correct. Après avoir effectué le tour de base, chaque agent sait dans quelle situation, parmi les quatres listées dans cet algorithme, il se situe. Si le nœud central existe, alors il est unique et les deux agents vont le rejoindre pour y accomplir le rendez-vous. Si l'arbre possède une arête centrale telle que les numéros de port à ses deux extrémités sont différents, alors les agents choisissent le nœud avec le plus grand numéro de port pour faire la rencontre. D'autre part, si l'arbre a une arête centrale avec des numéros de port égaux mais l'arbre n'est pas symétrique, alors les deux sous-arbres enracinés aux extrémités de l'arête centrale ne sont pas isomorphes et par conséquent leurs codes sont différents.

Dans ce cas, les agents achèvent le rendez-vous dans la racine de l'arborescence dont le code est lexicographiquement le plus grand.

Supposons maintenant que l'arbre possède une arête centrale et qu'il est symétrique, c'est à dire que les deux numéros de port de l'arête centrale sont égaux et que les deux sous-arbres enracinés à ses deux extrémités sont isomorphes. Supposons d'abord que les positions initiales ne sont pas symétriques. Alors les nombres de pas du tour de base effectué à partir de chacune de ces positions jusqu'à la première traversée de l'arête centrale doivent être différents. Ceci veut dire que les entiers x calculés par chaque agent sont différents. Notons-les par x_1 et x_2 et supposons, sans perte de généralité, que $x_1 > x_2$. Soit A l'agent qui a commencé plus tard l'exécution de la boucle « répéter jusqu'à la réalisation du rendez-vous » à l'instant t (si le départ est simultané, l'agent peut être choisi arbitrairement). À l'instant au plus $t + 2x_1$, il va y avoir une ronde durant laquelle l'un des agents traverse l'arête centrale et l'autre reste passif à l'une de ses extrémités. Le rendez-vous aura donc lieu au plus tard durant cette ronde. Supposons maintenant que l'arbre est symétrique et les agents se trouvent initialement sur des positions symétriques, mais le délai est $\theta > 0$; dans ce cas $x_1 = x_2$. Soit A définit comme auparavant. Une fois arrivé à l'extrémité de l'arête centrale, A va rester inactif pendant $x_1 + 1$ rondes et donc la rencontre va se faire au plus tard à la ronde $t + 1 + x_1$.

Calculons maintenant le temps d'exécution de l'algorithme RV-déterministe. Selon la définition 3.4.2, nous devons commencer le calcul de ce temps à partir du moment où A_2 a lancé sa première action. Le tour de base prend un temps $2(n-1)$. L'entier x calculé par chaque agent est au plus n car il ne dépasse pas la longueur du tour de base dans le sous-arbre où l'agent commence l'exécution de l'algorithme, enracinée dans l'une des extrémités de l'arête centrale. Le temps pour rejoindre le nœud central ou l'une des extrémités de l'arête centrale est au plus $n/2$. Enfin, le temps d'exécution de la boucle « répéter jusqu'à la réalisation du rendez-vous » est au plus $2x_1 \leq 2n$. D'où le temps total pour accomplir le rendez-vous est au plus $2(n-1) + n + \frac{n}{2} + 2n \leq \frac{11n}{2}$. □

Remarque : L'algorithme RV-déterministe est conçu pour résoudre le problème du rendez-vous dans un cadre déterministe, c'est à dire, sous l'hypothèse que soit les positions initiales des agents ne sont pas symétriques, soit le départ n'est pas simultané.
Dans le cas spécial du départ simultané, cet algorithme peut être renforcé pour retourner le résultat suivant : si les positions initiales ne sont pas symétriques, alors le rendez-vous est accompli, et si les positions initiales sont symétriques, alors les deux agents s'arrêtent

et annoncent que le rendez-vous est impossible.
Il est possible d'obtenir ce résultat en arrêtant l'algorithme après $\frac{11n}{2}$ rondes si le rendez-vous n'est pas achevé, où n est le nombre de nœuds de l'arbre que chaque agent apprend après avoir effectué le tour de base. Ceci résulte de la preuve du théorème 6.4.1 dans laquelle nous avons montré que si les positions initiales ne sont pas symétriques, alors le rendez-vous est achevé au plus tard après $\frac{11n}{2}$ rondes de l'exécution de l'algorithme par l'agent qui le commence en dernier (dans le cas du départ simultané, au plus $\frac{11n}{2}$ rondes après ce départ). D'où l'algorithme RV-déterministe fonctionnera toujours en temps $O(n)$ même après cette modification dans le cas d'un départ simultané.
Par contre, il serait impossible de renforcer cet algorithme de cette façon dans le cas d'un départ arbitraire parce que l'agent ne peut jamais savoir si le rendez-vous n'est pas accompli à cause que c'est impossible de le faire ou bien parce que l'autre agent devra arriver en retard.

6.5 Borne inférieure

Une fois le résultat ci-dessus est prouvé, une question évidente s'impose : peut-on améliorer le temps du rendez-vous $O(n)$? Autrement dit, existe-t-il un algorithme déterministe qui peut résoudre le problème du rendez-vous en un temps meilleur que $O(n)$? La réponse est clairement « non » si les positions initiales des agents se trouvent à une distance appartenant à $\Omega(n)$. Cependant, le résultat suivant montre que la réponse est aussi « non » même si la distance initiale entre les agents est égale à 1, même dans le cas d'un départ simultané et même dans la classe des arbres de degré borné.

Proposition 6.5.1 *Chaque algorithme déterministe qui résout le problème du rendez-vous prend un temps $\Omega(n)$ pour certains cas où les positions initiales des agents sont à une distance égale à 1 même pour un départ simultané. Ceci est valable même pour la classe d'arbres de degré borné.*

Preuve : Considérons une ligne de longueur $n-1$ telle que sur toute arête $e = \{u, v\}$ appartenant à cette ligne, le numéro de port associé à u est différent de celui associé à v (les numéros de port appartiennent à {0,1}).

Soit A un algorithme qui permet de résoudre le problème du rendez-vous et soient deux agents situés au milieu de cette ligne à distance 1. Assumons que les agents exé-

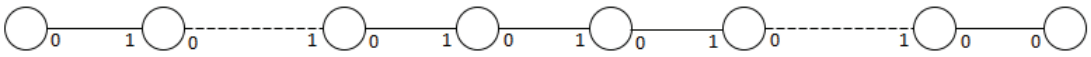

FIGURE 6.5 – Exemple de la ligne pour prouver la borne inférieure

cutent cet algorithme avec un départ simultané. Durant les premières $\lceil \frac{n}{2} \rceil - 1$ rondes, l'historique des deux agents est identique puisqu'aucun d'eux n'a atteint l'extrémité de la ligne. D'autre part, puisqu'ils exécutent le même algorithme et par induction sur le nombre de rondes, les agents seront toujours à distance 1 durant ces rondes. Donc il faut au moins $\lceil \frac{n}{2} \rceil$ rondes pour faire la rencontre, d'où un temps dans $\Omega(n)$. □

6.6 Conclusion

Dans ce chapitre, nous avons montré que le rendez-vous déterministe est faisable dans n'importe quel arbre à n noeuds. Nous avons présenté un algorithme déterministe qui fonctionne en temps $O(n)$ et nous avons montré que ce temps est optimal pour le rendez-vous déterministe. Nous avons utilisé deux approches : la première est de trouver un nœud particulier pour faire le rendez-vous , et la deuxième est d'attribuer des étiquettes différentes aux agents afin de les distinguer. Nous avons aussi introduit la notion de l'état actif ou passif d'un agent afin de résoudre le problème que peut causer un départ arbitraire dans la réalisation du rendez-vous. Nous avons montré que le rendez-vous déterministe est impossible quand les positions initiales sont symétriques et le départ est simultané. Dans tous les autres cas, notre algorithme garantit le rendez-vous. La question qu'on peut poser maintenant est la suivante : peut-on trouver une solution aléatoire pour accomplir le rendez-vous aussi dans le cas d'un arbre symétrique avec des agents situés initialement sur des positions symétriques ? Si oui, à quel prix peut-on le faire ?

Chapitre 7

Rendez-vous aléatoire dans l'arbre

Dans ce chapitre, nous allons présenter des algorithmes aléatoires dits « presque certain »qui accomplissent le rendez-vous avec une probabilité de succès d'au moins $1 - \frac{1}{n}$ dans les arbres à n nœuds, avec un n suffisamment grand. Dans ces algorithmes, les agents utilisent le lancé de la monnaie pour définir leurs déplacements. par exemple, si l'agent lance la pièce de monnaie et reçoit comme résultat « pile », alors il peut se déplacer vers un nœud voisin ; si par contre il reçoit « face », alors il doit rester inactif sur sa position actuelle. Cet ingrédient aléatoire va permettre aux agents de briser la symétrie de l'arbre et de réaliser le rendez-vous avec une très grande probabilité de succès.

7.1 Le rendez-vous aléatoire dans un arbre arbitraire

Nous allons traiter le cas du rendez-vous dans un arbre arbitraire dans lequel sont situés initialement deux agents dans des positions arbitraires. Nous proposons un algorithme presque certain qui va permettre aux agents d'accomplir leur rencontre. Le parcours de l'arête centrale se fait toujours par un aller-retour.

Algorithme RV-aléatoire
Effectuer le tour de base à partir de la position initiale.
Calculer le nombre n de nœuds dans l'arbre.
Si l'arbre a un nœud central **alors** aller au nœud central et y rester
sinon
rejoindre l'extrémité la plus proche de l'arête centrale ;
répétez $\lceil \log n \rceil$ fois ou jusqu'à la réalisation du rendez-vous, selon ce qui qui arrive d'abord :
rester inactif avec une probabilité de $\frac{1}{2}$
parcourir l'arête centrale avec une probabilité de $\frac{1}{2}$.

Théorème 7.1.1 *L'algorithme RV-aléatoire est presque certain et permet à deux agents situés initialement sur deux positions différentes d'accomplir le rendez-vous en temps $O(n)$ même dans le cas d'un départ arbitraire, et ceci dans n'importe quel arbre d'ordre n.*

Preuve : Nous allons d'abord montrer que l'algorithme est presque certain. Si l'arbre a un nœud central, alors le rendez-vous va se faire dans ce nœud. Supposons maintenant que l'arbre a une arête centrale. Soient A_1 le premier agent à rejoindre l'une des extrémités de l'arête centrale et A_2 celui qui le fait en deuxième lieu. Soit t la ronde à laquelle A_2 arrive à l'extrémité de l'arête centrale la plus proche, donc il va commencer à lancer la monnaie à la ronde $t+1$. À la ronde t, l'agent A_1 se trouve déjà sur l'une des extrémités de l'arête centrale. (Si les deux agents commencent à lancer la monnaie au même moment, alors nous les appelons A_1 et A_2 arbitrairement). Nous assumons que les extrémités sur lesquelles se trouvent les deux agents sont différentes ; sinon le rendez-vous sera accompli.

Dans chacune des rondes $t+1, \ldots, t+\lceil \log n \rceil$, ou jusqu'à la réalisation du rendez-vous, A_2 lance la pièce de monnaie et l'agent A_1 doit se trouver dans l'une des deux situations suivantes : soit il va lui aussi lancer la pièce ou il a terminé les lancers et il est inactif. Considérons maintenant l'événement E qui présume que le rendez-vous ne sera pas accompli durant une de ces rondes, c'est à dire que les agents se trouvent sur des extrémités différentes de l'arête centrale durant chacune de ces rondes. Prenons par exemple la ronde r. Au début et à la fin de cette ronde, les agents se trouvaient sur deux positions différentes sur l'arête centrale. On peut distinguer deux cas possibles : Le premier cas est celui où les deux agents ont lancé la pièce de monnaie et ils ont tous

les deux parcouru l'arête centrale, ou tous les deux sont restés inactif. La probabilité de cet événement est de $\frac{1}{2}$. Le deuxième cas est celui où l'agent A_2 a lancé la pièce et il est resté inactif, tandis que l'agent A_1 a fini ces lancers et il reste inactif à l'autre extrémité. La probabilité de cet événement est aussi $\frac{1}{2}$. D'où la probabilité de l'événement E est $(1/2)^{\lceil \log n \rceil} \leq 1/n$, et donc l'algorithme est presque certain.

Calculons maintenant le temps estimé de l'exécution de l'algorithme RV-aléatoire. On rappelle que le calcul du temps se fait à partir du moment où le deuxième agent, dans le cas d'un départ arbitraire, commence l'exécution de l'algorithme. Le tour de base se fait en $2(n-1)$. Le temps pour rejoindre le nœud central ou l'extrémité la plus proche de l'arête centrale est au plus n. La partie aléatoire de l'algorithme se fait en $\lceil \log n \rceil$ rondes. Ainsi le temps total estimé est au plus $3n + \lceil \log n \rceil$. □

7.2 Le rendez-vous aléatoire dans l'arbre de degré borné

Dans cette partie, nous allons présenter un algorithme presque certain qui permet de réaliser le rendez-vous en temps logarithmique dans n'importe quel arbre de degré maximum constant, si la distance initiale des agents est constante. La situation est la suivante : Les agents savent qu'il se trouvent dans un arbre de degré maximum constant Δ et à une distance initiale constante D. L'idée de notre algorithme presque certain pour faire le rendez-vous est la suivante.

En connaissant D et Δ, un agent peut trouver la borne supérieure $\alpha := \Delta^{D+1}$ sur la grandeur de chaque sous-arbre enraciné de profondeur D de l'arbre donné.Ensuite, l'agent divise toutes les rondes depuis son départ en segments consécutifs de longueur 4α. Pendant la première ronde de chaque segment, il lance la monnaie. Si le résultat est pile, le segment est actif, sinon il est passif. Dans un segment actif, l'agent fait le tour de base du sous-arbre de profondeur D enraciné en sa position initiale et ensuite reste immobile jusqu'à la fin du segment. Dans un segment passif, l'agent reste immobile. Après avoir complété $3\lceil \log n \rceil$ segments, l'agent s'arrête. Voici le pseudo-code de l'algorithme.

Algorithme RV-aléatoire-rapide
$\alpha := \Delta^{D+1}$.
Selon la première situation qui se présente, répéter $3\lceil \log n \rceil$ fois ou jusqu'à la réalisation du rendez-vous :

Rester sur sa position actuelle durant 4α rondes avec une probabilité $\frac{1}{2}$

et avec une probabilité $\frac{1}{2}$, faire :

Durant z rondes, effectuer le tour de base dans l'arbre de profondeur D enraciné dans sa position initiale et y rester durant les prochaines $4\alpha - z$ rondes.

Théorème 7.2.1 *L'algorithme RV-aléatoire-rapide est presque certain et permet d'accomplir le rendez-vous dans n'importe quel arbre d'ordre n et de degré maximum borné Δ, pour deux agents positionnés à une distance initiale constante D, si les agents connaissent n, D et Δ. Le temps d'exécution de cet algorithme se fait en $O(\log n)$ dans les arbres à n nœuds.*

Preuve : Supposons que $t_1 \leq t_2$ soient les temps de départ des deux agents ; notons par $S_1, S_2, ...$ les segments qui représentent les rondes consécutives de longueur 4α associées au premier agent et $T_1, T_2, ...$ ceux associées au deuxième agent. Assumons que t_2 se trouve dans le segment S_i. Puisque tous les segments ont la même longueur 4α, au moins une des propriétés suivantes doit être satisfaite. Soit pour tout entier q, les segments S_{i+q} et T_{1+q} vont avoir une intersection de longueur au moins 2α, soit pour tout entier q, les segments S_{i+1+q} et T_{1+q} vont l'avoir. Nous allons analyser seulement le premier cas puisque l'analyse du deuxième cas va être pareille. Les paires des segments S_{i+q} et T_{1+q}, pour $q = 0, 1, \ldots$ sont appelées associées.

Supposons que S_{i+q} est passif et que T_{1+q} est actif ; donc durant les premières $z \leq 2\alpha$ rondes de T_{1+q}, le deuxième agent fait le tour de base dans l'arborescence enracinée en sa position initiale et de profondeur D pendant que le premier agent reste immobile sur sa position initiale. Puisque la distance entre les positions initiales des deux agents est D, le deuxième agent va rejoindre la position du premier agent durant son tour de base et le rendez-vous va être accompli.

Considérons maintenant les segments associés S_{i+q} et T_{1+q}, pour $q = 0, 1, ..., 3\lceil \log n \rceil - 1$. Deux cas se présentent. Dans le premier cas, l'activité ou la passivité de ces deux segments sont décidées par le lancé de la monnaie ; donc la probabilité pour que S_{i+q} soit

passif et que T_{1+q} soit actif est $\frac{1}{4}$. Dans le deuxième cas, le premier agent a terminé l'exécution de l'algorithme (i.e., S_{i+q} est passif) et T_{1+q} est actif avec une probabilité de $\frac{1}{2}$. Par conséquent, la probabilité de faire le rendez-vous durant T_{1+q} est au moins $\frac{1}{4}$ dans les deux cas. Donc, la probabilité pour que le rendez-vous ne soit pas réalisable après $3\lceil \log n \rceil$ exécutions de la boucle de l'algorithme RV-aléatoire-rapide est au plus $(\frac{3}{4})^{3\lceil \log n \rceil} = (\frac{27}{64})^{\lceil \log n \rceil} < (\frac{1}{2})^{\log n} = \frac{1}{n}$; ce qui prouve que cet algorithme est presque certain. il accomplit le rendez-vous en temps $O(\log n)$, car la durée 4α de chaque segment est constante. □

7.3 Le rendez-vous aléatoire dans l'arbre de degré non borné

Nous allons voir maintenant que le coût du rendez-vous aléatoire change quand le degré maximal d'un arbre n'est pas constant. Considérons la classe des arbres C composée de deux arbres en étoile, chacun de degré $k-1$. Les centres de ces derniers sont liés par une arête. Plus précisément, on suppose que $n = 2k-1$ (la construction pour n pair est semblable). Un arbre $T(n,i)$ de la classe C est défini comme suit : il a deux noeuds adjacents v et w avec le numéro de port i aux noeuds v et w, correspondant à l'arête $\{v,w\}$. En plus, v et w sont adjacents à $k-1$ feuilles.

Supposons maintenant que deux agents sont situés sur les noeuds v et w (distance initiale égale à 1).

Question : En pire cas, dans combien de temps un agent peut trouver le bon port qui va lui permettre de faire le rendez-vous presque certain ?

Pour répondre à cette question, nous allons utiliser un lemme très connu dans la littérature qui est dû à Andrew Yao et qui est connu sous le nom du « principe minimax de Yao ». En voici l'énoncé :

Lemme 7.3.1 (Principe minimax de Yao [29]) *Soit $0 < \epsilon < 1/2$. Pour toute distribution de probabilité $\mathcal{P}$ sur l'ensemble des entrées, soit $\mathcal{A}(\mathcal{P})$ l'ensemble des algorithmes déterministes qui fonctionnent avec une probabilité d'erreur d'au plus 2ϵ sur $\mathcal{P}$. Pour $A \in \mathcal{A}(\mathcal{P})$, soit $C(A,\mathcal{P})$ le temps espéré d'exécution de A sur $\mathcal{P}$. Soit $\mathcal{R}$ l'ensemble*

des algorithmes aléatoires qui fonctionnent avec une probabilité d'erreur au plus ϵ pour n'importe quelle entrée et soit $T(R, I)$ le temps espéré de fonctionnement de $R \in \mathcal{R}$ sur l'entrée I. Alors pour toute distribution de probabilité $\mathcal{P}$ et pour tout $R \in \mathcal{R}$,

$$\min_{A \in \mathcal{A}(\mathcal{P})} C(A, \mathcal{P}) \leq 2 \max_{I} T(R, I).$$

L'application standard du lemme ci-dessus pour prouver la borne inférieure est la suivante. Nous construisons une distribution de probabilité sur l'ensemble des entrées d'un problème donné pour laquelle tout algorithme déterministe qui fonctionne avec une probabilité d'erreur d'au plus 2ϵ a un grand temps espéré de fonctionnement sur cette distribution de probabilité. Selon le lemme de Yao, ceci implique que tous les algorithmes aléatoires qui fonctionnent avec une probabilité d'erreur d'au plus ϵ pour n'importe quelle entrée doit avoir un grand temps espéré de fonctionnement pour une certaine entrée.

Nous considérons maintenant la tâche auxiliaire suivante appelée « Trouvez-trésor » et qui est définie comme suit : on a k boîtes numérotées $0, 1, \ldots k-1$ et dont une seule contient un trésor. Les boîtes peuvent être ouvertes une par une, dans un ordre arbitraire, jusqu'à ce que le trésor soit trouvé. Le temps d'exécution d'un algorithme qui peut résoudre le problème « Trouvez-trésor » se calcule en nombre de boîtes ouvertes. En utilisant le lemme de Yao, le lemme suivant peut être prouvé.

Lemme 7.3.2 (Trouver le trésor) *Chaque algorithme aléatoire pour le problème* Trouver-trésor *avec k boites qui fonctionne avec une erreur au plus $\frac{1}{\sqrt{k}}$ a le temps espéré de travail d'au moins $\frac{k}{16}$ pour une certaine donnée et pour un k suffisamment grand.*

Preuve : Une entrée du problème « Trouvez-trésor » avec k boîtes est le numéro de la boîte qui contient le trésor. Nous utilisons le lemme 7.3.1 avec $\epsilon = \frac{1}{\sqrt{k}}$ et avec la distribution de probabilité uniforme $\mathcal{P}$. Considérons maintenant n'importe quel algorithme déterministe qui peut résoudre ce problème avec une probabilité d'erreur d'au plus $\frac{2}{\sqrt{k}}$ pour $\mathcal{P}$. Pour k suffisamment grand, un tel algorithme doit ouvrir au moins $\lceil \frac{k}{2} \rceil$ boîtes.

Soit $a_1, a_2, \ldots, a_{\lceil \frac{k}{2} \rceil}$ la permutation des numéros de boîtes dans l'ordre d'ouverture par l'algorithme. Alors le temps espéré d'exécution de l'algorithme pour la distribution uniforme des entrées est au moins $\frac{1}{k}(1 + 2 + \cdots + \lceil \frac{k}{2} \rceil) \geq \frac{k}{8}$. En utilisant le lemme 7.3.1, on déduit que pour n'importe quel algorithme aléatoire qui fonctionne avec une probabilité d'erreur d'au plus $\frac{1}{\sqrt{k}}$, il existe une entrée pour laquelle cet algorithme résout le

problème en temps espéré au moins $\frac{k}{16}$. □

Nous sommes maintenant prêts à prouver notre résultat principal négatif. Ce résultat, en combinaison avec le théorème 7.2.1 montre clairement l'impact de la connaissance du degré maximum borné de l'arbre sur le temps de rendez-vous déterministe versus le rendez-vous aléatoire. Pour le rendez-vous déterministe, nous avons vu que la borne inférieure est linéaire même dans la ligne et même lorsque les deux agents savent qu'ils sont situés à une distance égale à 1 l'un par rapport à l'autre. Ainsi la connaissance du degré maximum borné du graphe n'apporte aucune aide en général.

En revanche, pour le rendez-vous aléatoire, cette restriction est cruciale. Nous avons montré dans le théorème 7.2.1 que pour tous les arbres de degré borné, les agents peuvent réaliser le rendez-vous en temps logarithmique, si ils savent les paramètres D, Δ et n ; donc plus particulièrement si ces agents se trouvent initialement à une distance égale à 1.

Nous verrons maintenant que ça ne va pas être le cas pour les arbres de degré non borné. En fait pour ces arbres, le coût du rendez-vous aléatoire est exponentiellement plus grand. Voici le résultat qui le montre.

Théorème 7.3.1 *Il existe une classe d'arbres d'ordre n et de degré maximum $\lceil \frac{n}{2} \rceil$, pour tous les entiers $n \geq 2$, telle que n'importe quel algorithme aléatoire et presque certain qui peut résoudre le problème du rendez-vous pour cette classe d'arbres, ne peut le faire en un temps espéré meilleur que $\Omega(n)$ pour un certains arbres à n noeuds de cette classe pour tout entier n.*

Preuve : Soit $n = 2k$; la preuve pour le cas où n est impair reste similaire. Considérons la classe C des arbres $T(n, i)$ définie au début de cette section. Les positions initiales des agents sont aux nœuds v et w.

Considérons un algorithme aléatoire A qui résout le problème du rendez-vous dans l'arbre $T(n, i)$ avec une probabilité d'erreur d'au plus $\frac{1}{n}$. Si le rendez-vous est accompli, alors au moins un des agents a parcouru l'arête $\{v, w\}$. D'où la probabilité qu'un des agents traverse cet arête est au moins $1 - \frac{1}{\sqrt{n}}$. Ceci est équivalent à résoudre le problème « Trouvez-trésor » avec k boîtes et avec une probabilité d'échec d'au plus $\frac{1}{\sqrt{n}} \leq \frac{1}{\sqrt{k}}$. En utilisant le lemme 7.3.2, le temps espéré pour que l'algorithme A résout le problème du

rendez-vous pour cette classe d'arbres est au moins $\frac{k}{16}$ pour certaines entrées et pour k suffisamment grand.

Donc pour n suffisamment grand, le temps espéré pour que l'algorithme A résout le problème du rendez-vous dans l'arbre $T(n, i)$ est au moins $\frac{k}{16} \in \Omega(n)$, pour un certain nombre i. □

Remarque : Le théorème 7.3.1 est vrai même si les positions initiales des agents sont à une distance de 1, même lorsque les agents connaissent la taille de l'arbre dans lequel ils se trouvent, et même quand ils commencent simultanément l'exécution de l'algorithme.

7.4 Conclusion

Dans ce chapitre, nous avons présenté un algorithme aléatoire et presque certain qui résout le problème du rendez-vous en temps $O(n)$ pour toutes les positions initiales des agents. Nous avons montré que le coût de cet algorithme peut être amélioré au temps $O(\log n)$ si les agents connaissent à l'avance l'ordre de l'arbre n, la distance constante D séparant les positions initiales des agents et le degré maximum constant Δ de l'arbre. Dans le cas où le degré maximum de l'arbre n'est pas borné, nous avons prouvé qu'il n'existe aucun algorithme aléatoire qui peut résoudre le problème du rendez-vous en un temps espéré meilleur que $\Omega(n)$.

Chapitre 8

Simulation

Les simulations touchent l'algorithme **RV-aléatoire-rapide**. Nous l'avons testé pour 15 arbres différents. Pour chaque arbre, l'exécution de l'algorithme est faite 1000 fois. Cet algorithme utilise les 3 paramètres suivants :

- n : le nombre de nœuds de l'arbre
- D : la distance initiale qui sépare les agents
- Δ : le degré maximal de l'arbre

Dans la construction de ces 15 arbres, nous avons essayé de garder tous les degrés aussi proche du degré maximal que possible. Notre objectif est de voir l'impact de chacun des paramètres cités ci-dessus sur le temps d'exécution de l'algorithme et sur la réussite ou l'échec du rendez-vous.

Afin de comprendre les tableaux ci-dessous qui présentent les résultats des simulations, voici quelques définitions :

- Le *temps moyen du succès* représente la moyenne du temps de tous les cas qui se sont terminés par la réussite du rendez-vous.
- Le *coût théorique* est le temps maximal qu'utilise chaque agent avant l'arrêt final de l'exécution de l'algorithme **RV-aléatoire-rapide**. Il est égal à $12\lceil \log n \rceil \Delta^{D+1}$. Dans les cas où le rendez-vous n'a pas eu lieu jusqu'à ce temps, l'algorithme était interrompu et l'échec était déclaré.
- Le *taux des succès* représente le pourcentage des cas de succès du rendez-vous durant les 1000 essais.

		n				
		8	20	40	60	100
$\Delta = 2$	temps moyen du succès	16.77	18.89	16.47	15.82	16.84
	coût théorique	144	240	288	288	336
	taux des succès	99.5%	99.9%	99.9%	99.9%	99.9%
$\Delta = 5$	temps moyen du succès	95.89	42.45	38.36	38.43	40.28
	coût théorique	1200	1500	1800	1800	2100
	taux des succès	99.8%	99.9%	99.9%	99.9%	99.9%
$\Delta = 8$	temps moyen du succès	267.82	236.5	263.63	255.15	256.94
	coût théorique	2304	3840	4608	4608	5376
	taux des succès	99.4%	99.9%	99.9%	99.9%	99.9%

TABLE 8.1 – Simulation de l'algorithme RV-aléatoire-rapide dans le cas où $D = 1$

		n				
		8	20	40	60	100
$\Delta = 2$	temps moyen du succès	66.55	76.01	70.82	66.49	68.92
	coût théorique	576	960	1152	1152	1344
	taux des succès	99.7%	99.9%	99.9%	99.9%	99.9%
$\Delta = 5$	temps moyen du succès	2535.06	2489.46	2661.5	2580.42	2535.34
	coût théorique	30000	37500	45000	45000	52500
	taux des succès	99.8%	99.9%	99.9%	99.9%	99.9%
$\Delta = 8$	temps moyen du succès	-	16819.97	17539.52	15325.62	16749.88
	coût théorique	-	245760	294912	294912	344064
	taux des succès	-	99.9%	99.9%	99.9%	99.9%

TABLE 8.2 – Simulation de l'algorithme RV-aléatoire-rapide dans le cas où $D = 3$

		n				
		8	20	40	60	100
$\Delta = 2$	temps moyen du succès	241.25	271.15	265.37	263.58	268.18
	coût théorique	2304	3840	4608	4608	5376
	taux des succès	99.8%	99.9%	99.9%	99.9%	99.9%
$\Delta = 5$	temps moyen du succès	62571.66	62206.08	63097.49	59641.98	64079.47
	coût théorique	750000	937500	1125000	1125000	1312500
	taux des succès	99.7%	99.9%	99.9%	99.9%	99.9%
$\Delta = 8$	temps moyen du succès	-	1055935.06	1190172.87	1075874.26	1079048.02
	coût théorique	-	15728640	18874368	18874368	22020096
	taux des succès	-	99.9%	99.9%	99.9%	99.9%

TABLE 8.3 – Simulation de l'algorithme RV-aléatoire-rapide dans le cas où $D = 5$

La conclusion de ces simulations est la suivante :

1. Le succès du rendez-vous est toujours supérieur à 99%.
2. Le temps moyen est toujours très inférieur au coût théorique de l'algorithme. Ce dernier a été calculé comme une borne qui assure le succès du rendez-vous avec forte probabilité pour tous les arbres avec les paramètres donnés.
3. Le temps moyen pour un n donné est plus élevé lorsque D ou Δ sont grands.
4. L'impact de la distance D sur le temps moyen est très fort par rapport à celui du degré maximum Δ.
5. Dans tous les essais, le temps moyen du succès représente moins de 10% de la borne qui constitue le coût théorique de l'algorithme.
6. Le nombre de nœuds de l'arbre n'a pas une grande influence sur le temps moyen du succès ; ceci est dû à la fonction logarithmique qui atténue l'impact de la grandeur.

Ces simulations ont été exécutées sur un PC double core avec 4 gig de RAM. Le code est écrit en langage Java.

Chapitre 9

Conclusion

Ce mémoire consiste de deux parties. La première c'est une description détaillée du problème du rendez-vous dans le domaine du calcul distribué tel qu'il est décrit dans la littérature. Nous avons présenté les modèles utilisés par les chercheurs et les derniers résultats publiés jusqu'à date. La deuxième partie, celle qui est le but principal de ce mémoire, c'est notre contribution à la résolution du problème du rendez-vous.

- Notre modèle se distingue des autres cités dans la littérature [10, 15, 28] par le fait que nos agents mobiles ne possèdent pas d'étiquettes qui permettent de les différencier.
- Nous avons présenté un algorithme déterministe qui permet à des agents de faire un rendez-vous, lorsque c'est possible, dans n'importe quel arbre en temps $O(n)$ où n représente le nombre de nœuds de cet arbre.
- Nous avons prouvé que ce rendez-vous s'avère impossible si et seulement si les agents se trouvent sur des positions initiales symétriques et que leur départ est simultané.
- Nous avons montré que notre algorithme est optimal. En effet, il n'existe aucun algorithme déterministe qui peut résoudre ce problème en un temps meilleur que $\Omega(n)$ dans tous les arbres et pour toutes les positions initiales, même si les agents se trouvent initialement à une distance très proche.

Nous avons ensuite présenté une solution utilisant un ingrédient aléatoire qui est le lancer de la monnaie pour résoudre le cas spécial qui ne permet pas la rencontre des agents de façon déterministe.

- Nous avons réalisé un algorithme presque certain qui permet aux agents d'accomplir le rendez-vous en temps espéré $O(n)$ dans n'importe quel arbre d'ordre n quelque soit la situation dans laquelle se trouvent les agents.
- Nous avons amélioré ce coût espéré à $O(\log n)$ dans le cas où les agents connaissent la distance initiale D constante qui les séparent, Δ le degré maximum constant de l'arbre et le nombre de noeuds n de l'arbre.
- Nous avons démontré qu'il n'existe aucun algorithme aléatoire qui peut résoudre le problème du rendez-vous en un temps meilleur que $\Omega(n)$ dans la classe de tous les arbres, même si la distance initiale des agents est 1.

Les problèmes ouverts laissés pour la recherche future sont :

1. Est-il possible d'appliquer cette approche aux anneaux ?
2. Peut-on la généraliser pour n'importe quel graphe d'ordre n ?
3. Comment le problème du rendez-vous change si le nombre d'agents est plus grand et s'il est inconnu aux agents ?

Annexe A

Code de la simulation

```
package rv;
import java.util.*;

/**
 *
 * @author Samir Elouasbi
 */
public class Rv {

    public static void main(String args[]) {

        //Création de l'arbre
        createTree28 tree = new createTree28();
        HashMap<Object, HashMap> myTree = tree.createTree28();

        //Infos nécessaires pour les agents
        Object initNodeAgent1 = "H"; //Position initiale du premier agent
        Object initNodeAgent2 = "G"; //Position initiale du deuxième agent
        int distance = 1; //Distance initiale entre les deux agents
        int delta = 2;   //Degré maximum de l'arbre
        Long alpha = (long)(Math.pow(delta, distance+1)); //Calcul du Alpha

        statRv stat = new statRv();
        long loop = stat.plafond(statRv.log2(myTree.size()));

        //Coût théorique
```

```
        Long maxTime = 4*alpha*3*loop;

        //Pour les statistiques
        ArrayList result = new ArrayList();
        ArrayList time = new ArrayList();
        ArrayList succes = new ArrayList();

        int e = 0;
        int exec = 1000; // Le nombre d'execution de l'algorithme

        while(e < exec){
            Boolean rdv = false;
            long temps = 0;
            int x = 0;
            while (x < 3*loop && !rdv){
                int i =0; //Compteur de rondes
                //Mettre les agents à leurs états initiaux
                Agent agent1 = new Agent();
                Agent agent2 = new Agent();


                ArrayList<String> bwA1 = new ArrayList<String>();
                ArrayList<String> bwA2 = new ArrayList<String>();

                Boolean agentMove1 = agent1.agentMove();

                Boolean agentMove2 = agent2.agentMove();


                /* Premier cas */
                if (agentMove1==false && agentMove2 == true){
                    bwA2 = agent2.basicWalk(myTree, initNodeAgent2,
initNodeAgent2, 0, distance);
                    bwA2.add(0, initNodeAgent2.toString());
                    while(!rdv){
                        if (initNodeAgent1.toString().equals(bwA2.get(i))){
                            rdv = true;
                            temps = temps + i;
                        }
                        i++;
                    }
                /* Deuxième cas */
```

```
                }else if (agentMove1==true && agentMove2 == false){
                    bwA1 = agent1.basicWalk(myTree, initNodeAgent1,
initNodeAgent1, 0, distance);
                    bwA1.add(0, initNodeAgent1.toString());
                    while(!rdv){
                        if (initNodeAgent2.toString().equals(bwA1.get(i))){
                            rdv = true;
                            temps = temps + i;
                        }
                        i++;
                    }
                /* Troisième cas */
                }else if (agentMove1==true && agentMove2 == true){
                    bwA1 = agent1.basicWalk(myTree, initNodeAgent1,
initNodeAgent1, 0, distance);
                    bwA1.add(0, initNodeAgent1.toString());
                    bwA2 = agent2.basicWalk(myTree, initNodeAgent2,
initNodeAgent2, 0, distance);
                    bwA2.add(0, initNodeAgent2.toString());
                    int z1 = bwA1.size();
                    int z2 = bwA2.size();

                    //DÉBUT DU TRAITEMENT z1<z2
                        if(z1 <= z2){
                            while(i<z1 && !rdv){
                                if (bwA1.get(i).equals(bwA2.get(i))){
                                    rdv = true;
                                }
                                i++;
                            }
                            if (!rdv){
                                while(i<z2 && !rdv){
                                    if (bwA1.get(0).equals(bwA2.get(i))){
                                            rdv = true;
                                    }
                                    i++;
                                }
                            }
                         }else if(z2 <= z1){
                            while(i<z2 && !rdv){
                                if (bwA1.get(i).equals(bwA2.get(i))){
                                    rdv = true;
```

```
                                }
                                i++;
                            }
                            if (!rdv){
                                while(i<z1 && !rdv){
                                    if (bwA1.get(i).equals(bwA2.get(0))){
                                            rdv = true;
                                    }
                                    i++;
                                }
                            }
                         }

                          if(rdv){
                              temps = temps + i-1;
                          }else{
                              temps = temps + (4*alpha);
                          }
                        /* Quatrième cas */
                        }else if (agentMove1==false && agentMove2 == false){
                            temps = temps + (4*alpha);
                        }
                        x++;
                    }
                // Stocker le temps après chaque exécution pour les statistiques
                time.add(temps);
                result.add(rdv);
                if(rdv){
                    succes.add(temps);
                }

                e++;
            }

            //Calcul du taux des succès
            Double sucTaux = (double)succes.size() / (double)e;



        } //Fin de la méthode main

    }//Fin dela classe Rv
```

```
*************************************************************************
Construction de l'arbre
*************************************************************************
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package rv;
import java.util.*;

/**
 *
 * @author selouasb
 */
public class createTree28 {


    public HashMap<Object, HashMap> createTree28(){


        /**************************************/
        /* Définition du noeud D              */
        /**************************************/
        // Create a node
        HashMap nodeD = new HashMap();
        // Put elements to the map
        nodeD.put("label", "D");
        nodeD.put("E", new Integer(0));


        /**************************************/
        /* Définition du noeud E              */
        /**************************************/
        // Create a node
        HashMap nodeE = new HashMap();
        // Put elements to the map
        nodeE.put("label", "E");
        nodeE.put("F", new Integer(0));
        nodeE.put("D", new Integer(1));
```

```
/*************************************/
/* Définition du noeud F             */
/*************************************/
// Create a node
HashMap nodeF = new HashMap();
// Put elements to the map
nodeF.put("label", "F");
nodeF.put("G", new Integer(0));
nodeF.put("E", new Integer(1));

/*************************************/
/* Définition du noeud G             */
/*************************************/
// Create a node
HashMap nodeG = new HashMap();
// Put elements to the map
nodeG.put("label", "G");
nodeG.put("H", new Integer(0));
nodeG.put("F", new Integer(1));

/*************************************/
/* Définition du noeud H             */
/*************************************/
// Create a node
HashMap nodeH = new HashMap();
// Put elements to the map
nodeH.put("label", "H");
nodeH.put("G", new Integer(0));
nodeH.put("I", new Integer(1));

/*************************************/
/* Définition du noeud I             */
/*************************************/
// Create a node
HashMap nodeI = new HashMap();
// Put elements to the map
nodeI.put("label", "I");
nodeI.put("H", new Integer(0));
nodeI.put("J", new Integer(1));

/*************************************/
/* Définition du noeud J             */
```

```
 /*************************************/
 // Create a node
HashMap nodeJ = new HashMap();
 // Put elements to the map
 nodeJ.put("label", "J");
 nodeJ.put("I", new Integer(0));
 nodeJ.put("K", new Integer(1));

 /*************************************/
 /* Définition du noeud K               */
 /*************************************/
 // Create a node
HashMap nodeK = new HashMap();
 // Put elements to the map
 nodeK.put("label", "K");
 nodeK.put("J", new Integer(0));




 /*****************************************/
 /* Création de l'arbre                      */
 /*****************************************/

  ArrayList<HashMap> tree = new ArrayList<HashMap>();

 tree.add(nodeK);
 tree.add(nodeI);
 tree.add(nodeJ);
 tree.add(nodeF);
 tree.add(nodeE);
 tree.add(nodeH);
 tree.add(nodeG);
 tree.add(nodeD);



 HashMap<Object, HashMap> hm = new HashMap<Object, HashMap>();
 for(int i=0; i < tree.size(); i++){
    HashMap hms = new HashMap();
```

```
            Iterator iteratorKey = tree.get(i).keySet().iterator();
                        while(iteratorKey.hasNext()){
                            Object test = iteratorKey.next();
                             if (test != "label"){
                                 hms.put(test, tree.get(i).get(test));
                             }
                        }

                        hm.put(tree.get(i).get("label"), hms);
        }

        return hm;
         }

}

**************************************************************************
Construction d'un agent
**************************************************************************

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package rv;
import java.util.*;

/**
 *
 * @author selouasb
 */
public class Agent {

        private Integer time = new Integer(0);
        //private String target = new String();
        private ArrayList<String> target = new ArrayList<String>();
        private Integer compteur = new Integer(0);
        private Integer distance = new Integer(0);

    public Boolean agentMove(){
        Random randomGenerator = new Random();
        int randomInt = randomGenerator.nextInt(2);
```

```
        if (randomInt == 0){
            return false;
        } else {
            return true;
        }
    }

    //Le Basic Walk
    public ArrayList<String> basicWalk(HashMap<Object, HashMap> arbre,
Object initiale, Object positionNext,
int port, int distanceD){

         Boolean find = false;

        int enterPort = 0;                    // Numéro du port d'entrée
        int exitPort = port;                  // Numéro du port de sortie
        Object actualNode = positionNext;     // Position actuelle
        Object nextNode = new Object ();  // Prochaine position
        Object keyNode = new Object ();

        if (initiale == positionNext){
            compteur++;
        }

        //Obtenir les voisins de la position actuelle
        HashMap nodesVoisins = arbre.get(actualNode);
        //if(time < (2*arbre.size() - 2)){
        if(compteur <= arbre.get(initiale).size()){
            Iterator iteratorKey = nodesVoisins.keySet().iterator();
            while(iteratorKey.hasNext()&& !find){
                keyNode = iteratorKey.next();
                if (Integer.toString(exitPort).equals(
nodesVoisins.get(keyNode).toString())){
                    nextNode = keyNode;
                    //nodesVoisins.remove(keyNode);
                    find = true;
                }
            }
            if (target.contains(nextNode.toString())
|| initiale.equals(nextNode)){
                distance = distance - 1;
            }
```

```
            else {
                distance++;
            }

            time++;
            target.add(nextNode.toString());
            enterPort =
Integer.parseInt(arbre.get(nextNode).get(actualNode).toString());
            if (distance == distanceD){
                exitPort = enterPort;
            }
            else {
                exitPort = (enterPort + 1) % arbre.get(nextNode).size();
            }

            // Appel reccursif pour faire le Basic Walk
            basicWalk(arbre, initiale, nextNode, exitPort, distanceD);

        }

        return target;

    }



}
```

Bibliographie

[1] Alpern, S. The rendezvous search problem. *SIAM J. Control Optim. 33* (May 1995), 673–683.

[2] Alpern, S., Baston, V. J., and Essegaier, S. Rendezvous search on a graph. *Journal of Applied Probability 36*, 1 (1999), pp. 223–231.

[3] Alpern, S., and Gal, S. *The theory of search games and rendezvous*. Kluwer Academic Publishers, 2003.

[4] Anderson, E. J., and Weber, R. R. The rendezvous problem on discrete locations. *Journal of Applied Probability 27*, 4 (1990), pp. 839–851.

[5] Bampas, E., Czyzowicz, J., Gasieniec, L., Ilcinkas, D., and Labourel, A. Almost optimal asynchronous rendezvous in infinite multidimensional grids. In *Proceedings of the 24th international conference on Distributed computing* (Berlin, Heidelberg, 2010), DISC'10, Springer-Verlag, pp. 297–311.

[6] Baston, V., and Gal, S. Rendezvous search when marks are left at the starting points. *Naval Research Logistics (NRL) 48*, 8 (2001), 722–731.

[7] Bshouty, N. H., Higham, L., and Warpechowska-gruca, J. Meeting times of random walks on graphs. *Information Processing Letters 69* (1999), 265.

[8] Czyzowicz, J., Kosowski, A., and Pelc, A. How to meet when you forget : log-space rendezvous in arbitrary graphs. In *Proceeding of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing* (New York, NY, USA, 2010), PODC '10, ACM, pp. 450–459.

[9] Czyzowicz, J., Labourel, A., and Pelc, A. How to meet asynchronously (almost) everywhere. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms* (Philadelphia, PA, USA, 2010), SODA '10, Society for Industrial and Applied Mathematics, pp. 22–30.

[10] Dessmark, A., Fraigniaud, P., Kowalski, D. R., and Pelc, A. Deterministic rendezvous in graphs. *Algorithmica 46* (September 2006), 69–96.

[11] Flocchini, P., Kranakis, E., Krizanc, D., Luccio, F., Santoro, N., and Sawchuk, C. Mobile agents rendezvous when tokens fail. In *Structural Information and Communication Complexity*, vol. 3104 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2004, pp. 161–172.

[12] Fraigniaud, P., and Pelc, A. Deterministic rendezvous in trees with little memory. In *Proceedings of the 22nd international symposium on Distributed Computing* (Berlin, Heidelberg, 2008), DISC '08, Springer-Verlag, pp. 242–256.

[13] Fraigniaud, P., and Pelc, A. Delays induce an exponential memory gap for rendezvous in trees. In *Proceedings of the 22nd ACM symposium on Parallelism in algorithms and architectures* (New York, NY, USA, 2010), SPAA '10, ACM, pp. 224–232.

[14] Kouckỳ, M. Universal traversal sequences with backtracking. *J. Comput. Syst. Sci. 65* (December 2002), 717–726.

[15] Kowalski, D., and Malinowski, A. How to meet in an anonymous network. In *In Proc. 13th Sirocco* (2006), pp. 44–58.

[16] Kowalski, D., and Pelc, A. Polynomial deterministic rendezvous in arbitrary graphs. In *Algorithms and Computation*, R. Fleischer and G. Trippen, Eds., vol. 3341 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2005, pp. 269–277.

[17] Kranakis, E., and Krizanc, D. An algorithmic theory of mobile agents. In *Proceedings of the 2nd international conference on Trustworthy global computing* (Berlin, Heidelberg, 2007), TGC'06, Springer-Verlag, pp. 86–97.

[18] Kranakis, E., Krizanc, D., and Markou, E. Deterministic symmetric rendezvous with tokens in a synchronous torus. *Discrete Applied Mathematics*.

[19] Kranakis, E., Krizanc, D., and Markou, E. Mobile agent rendezvous in a synchronous torus. In *LATIN 2006 : Theoretical Informatics*, J. Correa, A. Hevia, and M. Kiwi, Eds., vol. 3887 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2006, pp. 653–664.

[20] Kranakis, E., Krizanc, D., and Morin, P. Randomized rendez-vous with limited memory. In *Proceedings of the 8th Latin American conference on Theoretical informatics* (Berlin, Heidelberg, 2008), LATIN'08, Springer-Verlag, pp. 605–616.

[21] Kranakis, E., Krizanc, D., Santoro, N., and Sawchuk, C. Mobile agent rendezvous in a ring. *Distributed Computing Systems, International Conference on 0* (2003), 592.

[22] Marco, G. D., Gargano, L., Kranakis, E., Krizanc, D., Pelc, A., and Vaccaro, U. Asynchronous deterministic rendezvous in graphs. In *Mathematical Foundations of Computer Science 2005*, vol. 3618 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2005, pp. 271–282.

[23] Pelc, A. Disc 2011 invited lecture : Deterministic rendezvous in networks : Survey of models and results. In *Distributed Computing - 25th International Symposium, DISC 2011, Rome, Italy, September 20-22, 2011. Proceedings* (2011), pp. 1–15.

[24] Reingold, O. Undirected connectivity in log-space. *J. ACM 55* (September 2008), 17 :1–17 :24.

[25] Sawchuk, C. *Mobile Agent Rendezvous in the Ring*. PhD thesis, Carleton University, School of Computer Science, Ottawa, Canada, 2004.

[26] Schelling, T. *The strategy of conflict*. Harvard University Press, 1981.

[27] Stachowiak, G. Asynchronous deterministic rendezvous on the line. In *Proceedings of the 35th Conference on Current Trends in Theory and Practice of Computer Science* (Berlin, Heidelberg, 2009), SOFSEM '09, Springer-Verlag, pp. 497–508.

[28] Ta-Shma, A., and Zwick, U. Deterministic rendezvous, treasure hunts and strongly universal exploration sequences. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 2007), SODA '07, Society for Industrial and Applied Mathematics, pp. 599–608.

[29] Yao, A. C.-C. Probabilistic computations : Toward a unified measure of complexity. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science* (Washington, DC, USA, 1977), IEEE Computer Society, pp. 222–227.

[30] Yu, X., and Yung, M. Agent rendezvous : A dynamic symmetry-breaking problem. In *Automata, Languages and Programming*, F. Meyer and B. Monien, Eds., vol. 1099 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 1996, pp. 610–621.