UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

RENDEZ-VOUS SYNCHRONE DANS LES GRAPHES AVEC TRACES DISTINCTES

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR

ISSAM BOUTAHAR

SOUS LA SUPERVISION DU PROFESSEUR ANDRZEJ PELC

JUIN 2018

UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

Département d'informatique et d'ingénierie

Ce mémoire intitulé :

RENDEZ-VOUS SYNCHRONE DANS LES GRAPHES AVEC TRACES DISTINCTES

présenté par

Issam Boutahar

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 Mohand Said Allili ..... Président du jury
Dr Jurek Czyzowicz ..... Membre du jury

À

ALLAH

Le Très-Haut, le très grand, le tout puissant, le très miséricordieux
d'avoir permis à ce travail d'aboutir à son terme.
Au PROPHÈTE MOHAMED paix et salut sur lui.

À la mémoire de mon père Lhaj Mohammed qui m'a enseigné toujours l'importance de l'éducation.

À ma mère Lhaja Chrifa Elhilali qui m'a encouragé durant toutes mes études et qui ma accompagné par ses prières, son amour et sa bénédiction.

À ma femme Ghizlane qui a fait beaucoup de sacrifices et qui m'a donné du soutien pour que je puisse avancer dans mes études.

À mes enfants Iyad, Walid et Yassir.

À mes soeurs Soukaina et Amina ainsi que mes frères Alaa, Oussama, et Abdelhak.

À mon grand frère Samir Elouasbi qui m'a soutenu durant toutes mes études.

Remerciements

Je tiens à remercier tout d'abord mon directeur de recherche, Professeur Andrzej Pelc, pour sa patience, et surtout pour sa confiance, ses remarques et ses conseils, sa disponibilité et sa bienveillance.

Je voudrais également remercier les membres de mon jury pour leurs temps et pour l'honneur qu'ils m'ont fait en portant leur attention sur ce travail.

Je tiens aussi à remercier Samir Elouasbi, François Lépine, Antoine Robertson, ma soeur Soukaina Bouatahar, Jalal Amarof et Othmane Himadi pour leur contribution, leur aide et leur collaboration.

Finalement, je remercie tous mes amis qui, de près ou de loin, ont contribué à ce travail par leur encouragement et leur conseil.

Merci à vous tous

Table des matières

Résumé

Ce mémoire aborde le sujet du rendez-vous synchrone de deux agents mobiles avec des traces distinctes (jetons). Les deux agents, identifiés par des étiquettes différentes, sont placés sur des nœuds différents dans un graphe anonyme. L'exploration de ce dernier est nécessaire pour avoir une carte complète. Dans une ronde, l'agent mobile peut aller dans un des nœuds adjacents à son nœud actuel ou rester immobile dans ce dernier. Nous présentons des algorithmes déterministes qui permettent de faire le rendez-vous dans un temps optimal pour les trois scénarios suivants :

- Chaque agent mobile laisse un jeton d'une couleur différente sur son nœud initial. Les agents résolvent le rendez-vous dans un temps optimal $O(Exp + DlogL)$, où Exp est le temps d'exploration, D est la distance entre les positions initiales des agents et L est la grandeur de l'espace des étiquettes.

- Chaque agent mobile laisse deux jetons sur son nœud initial et au plus un jeton sur les autres nœuds. Les agents résolvent le rendez-vous dans un temps optimal $O(e + DLogL)$, où e est le nombre d'arêtes, D est la distance entre les positions initiales des agents et L est la grandeur de l'espace des étiquettes.

- Chaque agent mobile laisse un nombre illimité de jetons sur son nœud initial et au plus un jeton sur les autres nœuds. Les agents résolvent le rendez-vous dans un temps optimal $O(e)$ où e est le nombre d'arêtes.

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

Abstract

This thesis addresses the subject of the synchronous rendezvous of two mobile agents with distinct traces (tokens). The two agents, identified by different labels, are placed on different nodes in an anonymous graph. Exploration of the latter is necessary to have a complete map. In a round, the mobile agent can go to one of the nodes adjacent to his current node or remain motionless in the current node. We present deterministic algorithms that allow us to complete the rendezvous in an optimal time for the following three scenarios:

- Each mobile agent leaves a token of a different color in the starting node. Agents solve the rendezvous in optimal time $O(Exp + D \log L)$ where Exp is the exploration time, D is the distance between the tokens and L is the size of the space of the labels.

- Each mobile agent leaves two tokens in their home node and at most one token in the other nodes. The agents solve the rendezvous in optimal time $O(e + D \log L)$ where e is the number of edges, D is the distance between the tokens and L is the size of the space of the labels.

- Each mobile agent leaves an unlimited number of tokens in their home node and at most one token in the other nodes. The agents solve the rendezvous in optimal time $O(e)$ where e is the number of edges.

Keywords: mobile agent, rendezvous, distinct traces, deterministic algorithm.

Chapitre 1

Introduction

Sommaire

Le rendez-vous des agents mobiles était largement abordé dans le milieu de la recherche. Le problème du rendez-vous peut être expliqué comme suit : deux personnes arrivent à un grand centre commercial pour se rencontrer, mais sans aucun moyen de communiquer. Par la suite, chaque personne doit choisir entre attendre à un endroit dans l'espoir que l'autre commence à la chercher ou chercher activement l'autre personne. Cependant si les deux personnes choisissent d'attendre, elles ne se rencontreront jamais. La stratégie consiste donc à construire des trajets pour les deux personnes qui garantissent la rencontre dans un même lieu et temps.

Le problème du rendez-vous est utilisé dans plusieurs domaines d'applications informatiques, par exemple :

- Chercher des informations différentes par des agents logiciels dans plusieurs bases de données, situées dans différents coins du monde, et ensuite les échanger afin d'effectuer certains traitements.

- Collecter des échantillons d'un terrain contaminé par des robots et ensuite partager ces échantillons.

- Explorer des réseaux de communication pour diagnostiquer des pannes et ensuite effectuer la rencontre pour planifier les réparations.

Si on veut formaliser l'explication ci-dessus dans le domaine de l'informatique et précisément dans le cas de ce projet, on aura la situation suivante : il y a des entités nommées agents mobiles qui doivent se rencontrer dans un environnement modélisé par un graphe. On considère que le graphe a des étiquettes différentes sur les ports de chaque nœud, mais les nœuds eux-mêmes sont anonymes.

Les agents mobiles doivent explorer le graphe pour connaître l'environnement avant de choisir la stratégie pour effectuer le rendez-vous. Plusieurs recherches ont considéré différentes suppositions pour réaliser le rendez-vous comme : les étiquettes des agents ou agents anonymes, l'utilisation des jetons ou l'absence de jetons, le déplacement synchrone ou asynchrone, etc. [25][9]. Dans ce travail, les agents ont des étiquettes différentes et ils laissent des traces sur les nœuds en utilisant des jetons afin d'améliorer le temps pris pour arriver au rendez-vous.

1.1 Le modèle

Dans ce mémoire, nous étudions le problème du rendez-vous des agents mobiles en utilisant des traces distinctes. Ces traces sont des jetons que les agents peuvent poser sur les nœuds.

Le réseau est modélisé par un graphe connexe anonyme non orienté $G = (V, E)$. Le graphe a des ports à chaque nœud correspondant à des arêtes incidentes et numérotées de 0 à $d-1$ où d est le degré d'un nœud. Ces ports sont perçus par les agents mobiles. Ces

derniers ont des étiquettes différentes afin de briser la symétrie. Les étiquettes des agents sont des entiers positifs. Le besoin de briser la symétrie peut être expliqué comme suit : Supposons que deux agents mobiles identiques (anonymes), c'est-à-dire sans étiquettes, sont situés sur deux nœuds différents d'un réseau modélisé, comme un anneau avec numérotation des ports 0,1 dans le sens horaire, commencent simultanément et exécutent le même algorithme déterministe. Les agents auront le même comportement, soit garder la même distance entre eux après chaque pas, ce qui rend le rendez-vous impossible.

Les traces distinctes sont des jetons disponibles pour les agents afin qu'ils les placent sur les nœuds visités. Chacun des agents utilise des jetons de couleur différente.

Les deux agents sont placés au début sur deux nœuds différents. Les agents ont des horloges locales qui tic-tacquent à la même cadence, une fois par ronde. Chaque agent est réveillé pendant une ronde, possiblement différente de celle de l'autre agent. L'horloge de l'agent est initialisée à son réveil.

Les agents se déplacent dans le graphe d'une manière synchrone. Pendant chaque ronde, chaque agent peut soit rester sur le nœud courant, soit se déplacer sur un nœud adjacent. Il n'y a aucune limite dans la mémoire des agents. Le rendez-vous se réalise lorsque les agents mobiles se retrouvent dans le même nœud dans une ronde. Les agents ne se rencontrent pas à l'intérieur des arêtes et ne s'aperçoivent pas du fait de croisement sur une arête. Le temps du rendez-vous est comptabilisé par le nombre de rondes entre le réveil du premier agent et le rendez-vous.

Dans ce mémoire, nous présentons trois scénarios qui diffèrent par la disponibilité des jetons pour les agents :

- Le premier scénario permet de laisser un jeton sur le nœud initial de chaque agent mobile. L'exploration du graphe avec un jeton stationnaire est considérée comme une procédure **boîte noire**, cf. [3].

- Le deuxième scénario permet de laisser deux jetons sur le nœud initial de chaque agent mobile et au plus un jeton de chaque agent sur les autres nœuds. L'exploration du graphe se fait par l'algorithme de parcours en profondeur (DFS).

- Le troisième scénario permet de laisser un nombre illimité de jetons sur le nœud initial de chaque agent mobile et au plus un jeton de chaque agent sur les autres nœuds. L'exploration du graphe se fait par DFS.

1.2 Les résultats

Ce mémoire présente trois algorithmes déterministes concernant le problème de rendez-vous avec des traces distinctes :

Le premier algorithme fonctionne dans le scénario où chacun des agents mobiles peut laisser un seul jeton au point de départ. Il permet de faire le rendez-vous dans un temps $O(Exp + D \log L)$, où L est la grandeur de l'espace des étiquettes, D est la distance entre les positions initiales des agents et Exp est le temps de l'exploration avec jeton stationnaire, qui est une boîte noire. Nous avons aussi trouvé une borne inférieure $\Omega(Exp + D \log L)$ sur le temps de rendez-vous dans ce scénario.

Nous pouvons donc conclure que le temps $O(Exp + D \log L)$ utilisé par notre algorithme de rendez-vous est optimal pour la classe des graphes arbitraires. Le meilleur algorithme d'exploration des graphes arbitraires connu avec un jeton stationnaire est $O(n^4)$ pour les graphes avec n nœuds [3].

Le deuxième algorithme fonctionne dans le scénario où chacun des agents mobiles peut laisser deux jetons sur le nœud initial et au plus un jeton sur les autres nœuds. Il permet de faire le rendez-vous dans un temps $O(e + D \log L)$, où L est la grandeur de

l'espace des étiquettes, D est la distance entre les positions initiales des agents et e est le nombre d'arêtes. Nous avons aussi prouvé que ce temps est optimal dans ce scénario.

Le troisième algorithme fonctionne dans le scénario où chacun des agents mobiles peut laisser un nombre illimité de jetons sur le nœud initial et au plus un jeton sur les autres nœuds. Il permet de faire le rendez-vous dans temps $O(e)$, où e est le nombre d'arêtes. Ce temps est optimal dans ce scénario.

Chapitre 2

Revue de la littérature

Sommaire

2.1 Introduction

Utilisé dans plusieurs domaines, le concept du rendez-vous des agents mobiles a été largement étudié dans le milieu de la recherche. Imaginons deux astronautes déposés dans deux endroits distancés sur un corps sphérique avec le but de se rencontrer ou deux parachutistes qui doivent se rencontrer après avoir été parachutés séparément d'un avion [25]. Pour analyser ces cas, plusieurs articles ont traité le problème du rendez-vous dans des scénarios divers. Dans cette revue de la littérature, nous allons voir ce concept dans différents types d'environnements (graphe ou terrain géométrique) ainsi qu'avec deux scénarios de déplacement des agents, soit déterministe ou aléatoire.

Aussi, cette revue de la littérature aborde le problème de l'exploration par des agents mobiles où ces derniers doivent parcourir un environnement. Dans le but de créer une carte du réseau ou de trouver un objet dans un labyrinthe, l'agent mobile doit visiter tous les nœuds et traverser toutes les arêtes. Les auteurs utilisent différentes techniques pour explorer l'environnement, que nous allons expliquer dans ce chapitre.

En faisant le lien avec notre mémoire, cette revue de la littérature commence par les articles qui ont traité de l'exploration. Par la suite, elle aborde les articles qui parlent du problème de rendez-vous. Cette dernière partie va être subdivisée en trois volets : rendez-vous déterministe, rendez-vous aléatoire et rendez-vous en présence de pannes.

2.2 Exploration d'un environnement par les agents mobiles

L'article d'Albers et Henzinger [1] est parmi les premiers qui ont traité le sujet de l'exploration dans le contexte des graphes orientés. Un agent mobile doit visiter tous les nœuds et traverser toutes les arêtes d'un graphe fortement connexe. Les auteurs utilisent la technique *divide-and-conquer* (diviser pour régner) où l'agent explore des sous-graphes en visitant les arêtes qui ne sont pas encore visitées jusqu'à ce qu'il soit bloqué sur un nœud visité. Par la suite, il y a une relocalisation vers un autre sous-graphe.

Le temps de fonctionnement de l'algorithme a une borne supérieure $d^{O(logd)}m$ où m est le nombre d'arêtes dans le graphe et d est le nombre minimum d'arêtes qui doivent être ajoutées pour rendre le graphe Eulérien. Les auteurs montrent également que la borne inférieure sur le temps de l'exploration est de $d^{\Omega(logd)}m$.

En 2001, Duncan et al. [15] ont ajouté des contraintes au problème de l'exploration d'un graphe inconnu. La première contrainte correspond à l'attachement du robot par

une corde au nœud de départ, et le robot ne peut pas aller plus loin que la longueur de la corde. La deuxième contrainte consiste à imposer au robot un réservoir de carburant de capacité limitée C, ce qui force le robot à retourner périodiquement au nœud de départ. Les auteurs prouvent que le robot peut explorer le graphe dans un temps de $O(|E|)$ dans les deux scénarios. Ils ont aussi prouvé que leur algorithme est optimal puisque la borne inférieure est de $\Omega(|E|)$.

En 2004, Dessmark et Pelc [10] ont abordé le problème de l'exploration des graphes non orientés où les agents ont une connaissance limitée de l'environnement. Les auteurs considèrent trois scénarios étudiés dans trois classes de graphes : le premier scénario correspond à l'agent qui n'a aucune connaissance du graphe (sans carte) ; le deuxième correspond à l'agent qui a une copie de la carte sans marquage du nœud de départ de l'agent (carte non ancrée) ; le dernier scénario correspond à l'agent qui a une copie de la carte avec un jeton sur le nœud de départ (carte ancrée). Les trois classes de graphes considérées sont : les lignes, les arbres et les graphes arbitraires. L'article traite de la performance des algorithmes en calculant leur ratio compétitif. Ce dernier est la borne supérieure sur le ratio entre son coût (nombre d'arêtes traversées) et le coût d'un algorithme optimal ayant une connaissance complète du graphe, prise sur tous les graphes de la classe et tous les nœuds de départ.

Dans la classe des lignes, les auteurs montrent que l'utilisation de l'algorithme de parcours en profondeur dans le scénario sans carte est optimale avec le ratio compétitif 2, contrairement aux scénarios de la carte non ancrée et celle ancrée où l'utilisation du DFS n'est pas optimale. Les ratios compétitifs optimaux pour le deuxième et le troisième scénarios sont respectivement $\sqrt{3}$ et $\frac{7}{5}$.

Pour la classe des arbres, les auteurs établissent également que le DFS est optimal uniquement dans le scénario sans carte avec le ratio compétitif 2. Pour le deuxième et le troisième scénarios, ces ratios optimaux sont respectivement strictement inférieurs à

2 et $\frac{3}{2}$. De plus, pour le scénario de la carte non ancrée, une borne inférieure de $\sqrt{3}$ est valide.

Finalement, concernant les graphes arbitraires, le DFS est optimal pour tous les scénarios et le ratio compétitif optimal est 2.

En 2010, Chalopin et al. [3] ont abordé le problème de la construction d'une carte d'un graphe inconnu par un agent mobile. Les auteurs utilisent la technique Séquences d'exploration universelle (UXS) pour résoudre le problème d'exploration. Une suite UXS est une suite de numéros de ports dont l'utilisation permet de visiter toutes les arêtes d'un graphe à partir de n'importe quel nœud. Le coût de l'exploration est le nombre de traversées d'arêtes.

L'article traite deux scénarios : le premier scénario suppose que l'agent connaît le nombre de nœuds du graphe n et le degré maximal d, mais son nœud de départ n'est pas identifié. Les auteurs proposent un algorithme qui résout le problème de l'exploration dans un temps de $O(n^6 d^2 \log n)$ avec $O(n^6 d^2 \log n)$ bits de mémoire. Ils donnent aussi un autre algorithme avec un temps de $O(n^6 d^3 \log n)$ en utilisant $O(n^6 d^2 \log n \log d)$ bits de mémoire.

Le deuxième scénario correspond à un graphe où le nœud de départ de l'agent est identifié par un jeton. Dans ce cas, il est possible de produire une carte fidèle du graphe (c'est-à-dire sa copie isomorphe). Les auteurs présentent un algorithme utilisant une mémoire de taille optimale $\Theta(\log n)$ et qui fonctionne en temps polynomial. Ils ont aussi proposé un algorithme qui nécessite plus de mémoire, soit $O(dn \log n)$, mais qui est plus rapide avec un temps d'exploration de $O(n^3 d)$.

2.3 Rendez-vous déterministe

Un rendez-vous est dit déterministe lorsque le déplacement des agents mobiles est basé uniquement sur leur historique. Ce dernier se résume à leur identité et à tout ce qu'ils ont vu depuis leur départ. La réalisation du rendez-vous déterministe est conditionnelle au principe de briser la symétrie (ex. : différentes identités des agents ou marquage des nœuds).

2.3.1 Rendez-vous synchrone déterministe

Dessmark et al. [9] ont abordé le problème du rendez-vous déterministe en 2006 dans le mode synchrone. Les agents se déplacent selon des rondes synchrones et la rencontre des agents se fait dans un nœud. Les auteurs considèrent deux agents mobiles qui se déplacent dans un graphe anonyme connexe non orienté, et qui ont des étiquettes différentes. Les auteurs étudient deux scénarios : simultané (départ des agents mobiles en même temps) et arbitraire (départ arbitraire).

Dans la classe des arbres, les auteurs donnent un algorithme pour résoudre le problème du rendez-vous avec un départ arbitraire dans un temps de $O(n + \log l)$, où n est le nombre de nœuds et l est l'étiquette la plus petite. Ils montrent aussi que ce résultat est optimal, car ils ont prouvé la borne inférieure $\Omega(n + \log l)$, même pour le départ simultané.

Dans la classe des anneaux, les auteurs présentent un algorithme valide pour un départ simultané. Cet algorithme travaille dans un temps de $O(D \log l)$, où D est la distance entre les nœuds de départ. Ils ont établi aussi l'optimalité de ce temps de rendez-vous pour le départ simultané. Les auteurs ont prouvent qu'il existe une borne

inférieure pour réaliser le rendez-vous dans un temps $\Omega(n + D \log l)$ en ce qui concerne le départ arbitraire, et ils ont soumis deux algorithmes :

- Le premier algorithme résout le problème du rendez-vous dans un temps de $O(n \log l)$ quand les agents connaissent n.

- Le deuxième algorithme résout le problème du rendez-vous dans un temps de $O(l\tau + ln^2)$ quand les agents ne connaissent pas n, où τ est la différence entre les temps de départ.

Pour la classe des graphes arbitraires, les auteurs donnent un algorithme qui résout le problème de rendez-vous dans un temps $O(n^5\sqrt{\tau \log l} \log n + n^{10} \log^2 n \log l)$.

Cependant, ils ont posé la question suivante : Existe-t-il un algorithme qui résout le problème du rendez-vous en temps polynomial indépendamment de τ ?

La réponse à cette question a été donnée par Ta-Shma et Zwick [26] en 2007. Les auteurs soumettent un algorithme déterministe qui résout le problème du rendez-vous dans un temps de $\tilde{\mathrm{O}}(n^5 l)$ ($\tilde{\mathrm{O}}$ est utilisé comme une variante de la notation O, qui ignore les facteurs logarithmiques). Également, ils prouvent que les agents peuvent se rencontrer dans un temps de $\tilde{\mathrm{O}}(d^2 n^3 l)$ s'ils sont placés dans un graphe de degré borné d.

Fraigniaud et Pelc [17] ont étudié la grandeur minimum de la mémoire requise pour que deux agents anonymes puissent faire un rendez-vous dans la classe des arbres. Les auteurs indiquent qu'une borne inférieure $\Omega(\log n)$ bits de mémoire était requise pour assurer le rendez-vous, même pour la classe de lignes de longueur n, dans le cas d'un départ arbitraire. Les auteurs prouvent aussi que pour un départ simultané, les agents ont besoin de seulement $O(\log l + \log \log n)$ bits de mémoire pour réaliser le rendez-vous et que c'est optimal.

Dans le même contexte, Czyzowicz et al. [7] ont étudié la grandeur minimum de la mémoire requise pour faire un rendez-vous dans les graphes arbitraires. Les agents sont anonymes et se trouvent dans un graphe inconnu, connexe et non orienté. Les auteurs

déterminent que les agents peuvent se rencontrer dans tous les graphes de n nœuds avec $O(\log n)$ bits de mémoire, quel que soit le délai entre les temps de début. Ils ont prouvé aussi que ce résultat est optimal : il correspond à la borne inférieure $\Omega(\log n)$ sur la grandeur de la mémoire, même pour un départ simultané.

En 2015, Miller et Pelc [23] ont cherché un compromis entre la taille minimale de l'information requise fournie aux agents et le coût pour résoudre le problème de rendez-vous et le problème de la chasse au trésor. Les agents sont anonymes et se trouvent dans un graphe connexe non orienté dont les nœuds ont des étiquettes distinctes. Cette information, possiblement différente pour les deux agents, est fournie aux agents sous la forme de suites binaires. Pour la classe des graphes arbitraires, les auteurs donnent la borne supérieure $O(D \log (D.\frac{e}{C}))$ et la borne inférieure $\Omega(D \log \frac{e}{C})$ sur la taille de l'information où C est le coût de résolution du problème de rendez-vous ainsi que du problème de chasse au trésor et e est le nombre d'arêtes. Concernant la classe des arbres, les auteurs donnent respectivement une borne supérieure et une borne inférieure serrées, soit $O(D \log (\frac{e}{C}))$ et $\Omega(D \log (\frac{e}{C}))$.

Dans [22], les mêmes auteurs focalisent l'attention sur le minimum d'information requise pour faire le rendez-vous dans un temps optimal de $\theta(D)$. Cette fois-ci, l'information donnée aux deux agents est identique. Les auteurs considèrent des agents mobiles avec des étiquettes de l'ensemble $\{1, ..., L\}$ dans un graphe anonyme. Ils ont montré que pour réaliser le rendez-vous dans des réseaux de n nœuds, les agents ont besoin d'une information contenant la taille minimale $\Omega(D \log \frac{n}{D} + \log \log L)$, où D est la distance entre les nœuds de départ. Ils ont prouvé aussi une borne supérieure serrée, au moyen d'un algorithme de rendez-vous qui utilise une information de taille $O(D \log \frac{n}{D} + \log \log L)$.

En 2016, Dieudonné et Pelc [11] ont abordé le problème du rassemblement de plusieurs agents mobiles anonymes dans des graphes arbitraires, connexes et non orientés. Les auteurs traitent deux scénarios :

- Le premier scénario correspond à la supposition que les agents connaissent la borne supérieure N sur la taille du graphe. L'algorithme des auteurs garantit le rassemblement avec détection (les agents mobiles deviennent conscients du rendez-vous) dans un temps polynomial en N.

- Le deuxième scénario correspond à la supposition que les agents ne connaissent pas la borne supérieure N sur la taille du graphe. Les auteurs ont construit un autre algorithme qui garantit le rassemblement sans détection (les agents mobiles ne sont pas conscients du rendez-vous) dans un temps polynomial et dont la taille du graphe est inconnue. Dans les deux cas, les algorithmes fonctionnent pour toutes les configurations initiales des agents pour lesquelles le rassemblement est faisable.

2.3.2 Rendez-vous asynchrone déterministe

De Marco et al. [21] ont été parmi les premiers à avoir traité le problème du rendez-vous asynchrone déterministe dans les réseaux. En 2006, ces auteurs considèrent le modèle suivant : deux agents ont des identificateurs différents dans un graphe connexe non orienté et n'ont aucune connaissance de sa topologie. Les agents se déplacent de façon asynchrone. Les nœuds sont anonymes, les ports ont des étiquettes et le rendez-vous est permis à l'intérieur d'une arête. La mesure d'efficacité du rendez-vous est le nombre de traversées d'arêtes, appelé le coût.

Dans la classe des lignes infinies, les auteurs donnent un algorithme qui résout le problème du rendez-vous. Le résultat est un coût de $O(D|L_{min}|^2)$ si D est connu et un coût de $O(D+|L_{max}|^3|)$ si D est inconnu, où D est la distance entre les points de départ, L_{min} et L_{max} sont respectivement l'étiquette la plus courte et l'étiquette la plus longue des agents. Les auteurs prouvent aussi que ces résultats sont valables pour l'anneau.

Dans la classe des anneaux, les auteurs proposent un algorithme ayant un coût de résolution optimal de $O(n|L_{min}|)$ si la taille n de l'anneau est connue, et de $O(n|L_{max}|)$ si n est inconnu.

Dans la classe des graphes arbitraires, les auteurs ont dessiné un algorithme conditionnel à la connaissance de la borne supérieure sur la taille du graphe, la topologie et les points de départ des agents. Ces derniers se rencontrent dans un coût optimal de $O(D|L_{min}|)$. Dans le cas où une seule borne supérieure sur la taille du graphe est connue, les auteurs proposent un algorithme à coût exponentiel.

Pour conclure l'article, les auteurs se sont questionnés sur la possibilité d'avoir un rendez-vous asynchrone déterministe dans n'importe quel graphe fini connexe sans connaître aucune borne supérieure sur la taille.

En 2012, Czyzowicz et al. [8] répondent à la question en soumettant un algorithme asynchrone où le rendez-vous est faisable dans n'importe quel graphe dénombrable (fini ou infini), sans connaître la borne supérieure de la taille du graphe dans le cas des graphes finis. Par contre, les auteurs n'ont pas indiqué le coût de l'algorithme, mais ont mentionné qu'il était exponentiel. Pour finir, ils ont posé la question suivante : Existe-t-il un algorithme de rendez-vous asynchrone déterministe qui fonctionne pour tous les graphes finis connexes inconnus, et ayant un coût polynomial en fonction des étiquettes des agents et de la taille du graphe ?

Dieudonné et al. [13] ont répondu à cette question. Ils ont donné un algorithme qui résout le problème du rendez-vous asynchrone avec un coût polynomial en fonction de la taille du graphe et de la longueur de l'étiquette la plus petite de l'agent. De plus, ils ont résolu les problèmes suivants (en utilisant le même coût) :

- Taille de l'équipe : chaque agent doit trouver le nombre total d'agents.

- L'élection du chef : tous les agents doivent produire l'étiquette d'un agent unique.

- Le renommage parfait : tous les agents doivent adopter de nouvelles étiquettes différentes de l'ensemble $\{1, ..., k\}$ où k est le nombre d'agents.

- Le bavardage : chaque agent a initialement une information et tous les agents doivent afficher toutes les informations.

Guilbault et Pelc [18] ont effectué un changement dans le modèle en considérant des graphes dénombrables, possiblement infinis, et des agents anonymes. Les auteurs utilisent la notion de vue de l'agent et la notion de chemin palindrome. La vue d'un agent à partir d'un nœud v, c'est l'arbre enraciné infini avec des ports étiquetés représentant les chemins débutant dans v. Un chemin r est dit palindrome lorsqu' il a la même séquence de ports que son chemin inverse. Les auteurs garantissent le rendez-vous des agents si les vues à partir des positions initiales des agents sont différentes ou si ces positions sont connectées par un chemin palindrome. Par contre, le coût de l'algorithme de rendez-vous est inefficace, car il est exponentiel en taille du graphe.

Contrairement aux articles précédents, Cieliebak et al. [6] ont étudié le problème du rassemblement déterministe asynchrone dans le plan. La configuration initiale est composée de n robots dont chacun possède son propre système de coordonnées local. Un robot est doté de capacités sensorielles et observe le monde en activant ses capteurs, ce qui renvoie une image de la position de tous les autres robots par rapport à son système de coordonnées local. Il est également est doté de capacités motrices et il peut se déplacer librement dans le plan. Les auteurs ont considéré une faible faculté appelée détection de multiplicité (la capacité de détecter si, à un moment donné, il y a un ou plus d'un robot, ou aucun) dans un point du plan. Les robots sont inconscients (ils ne se souviennent pas des résultats et des observations des cycles précédents), désorientés (ils n'ont pas de système de coordonnées commun) et asynchrones.

Les auteurs utilisent le principe du cycle regarder-calculer-bouger. Pendant un cycle, un robot obtient une image de l'environnement (regarder), fait des calculs en se basant

sur cette image (calculer) et se dirige vers la destination calculée (bouger). Ils ont prouvé que le problème du rassemblement peut être résolu pour tout $n > 2$ et toute configuration initiale, même dans les conditions restrictives de l'asynchronisme, de l'oubli et de la désorientation.

2.4 Rendez-vous aléatoire

Après avoir traité du scénario déterministe, nous allons présenter dans cette section un autre type de rendez-vous. Le rendez-vous aléatoire est un scénario où les agents mobiles utilisent un ingrédient aléatoire, comme le lancer de la monnaie, pour déterminer leurs déplacements. Pelc [25] mentionne que l'analyse du rendez-vous aléatoire nécessite souvent des outils analytiques. Aussi, il faut mentionner que les algorithmes aléatoires sont en général presque certains, c'est-à-dire que la probabilité de succès est d'au moins $1 - 1/n$ où n est le nombre de nœuds, et leur temps de fonctionnement soit au moins cette probabilité, soit sa valeur espérée est estimée.

Premièrement, dans le cas de la classe des arbres, nous supposons deux agents anonymes dans un réseau anonyme avec des ports étiquetés. Elouasbi et Pelc [16] ont présenté deux algorithmes presque certains dans un mode synchrone. Le premier algorithme résout le problème du rendez-vous dans un temps de $O(n)$ pour des positions initiales arbitraires des agents. Le deuxième algorithme résout le problème dans un temps de $O(\log n)$ dans le scénario où les agents connaissent la distance séparant leurs positions initiales ainsi que le degré maximal de l'arbre.

Deuxièmement, dans le cas de la classe des anneaux, nous allons parler de deux articles qui traitent deux différentes hypothèses. Kranakis et al. [20] ont cherché un compromis entre le temps espéré du rendez-vous et la mémoire nécessaire. L'anneau est

synchrone, anonyme et orienté avec n nœuds. Les auteurs prouvent que $O(\log \log n)$ bits de mémoire est suffisant et nécessaire pour résoudre le problème du rendez-vous dans un temps de $O(n)$.

Kranakis et al. [19] ont repris le modèle précédent en rendant l'anneau bidirectionnel et en supposant deux agents ayant des vitesses respectives différentes u,1. Les auteurs présentent un algorithme qui ne nécessite aucune connaissance et qui résout le problème du rendez-vous dans un temps espéré de $\frac{un}{2(u^2-1)}$ avec un seul bit aléatoire que l'agent utilise une seule fois pour choisir sa direction. Les auteurs démontrent que cet algorithme est optimal parmi tous les algorithmes qui utilisent un seul bit. Par la suite, les auteurs ont dessiné un autre algorithme où les agents ont besoin de connaître un entier donné K ainsi que le temps de déplacement $\frac{n}{u+1}$ après chaque choix de la direction. En utilisant $K+1$ bits aléatoires, le rendez-vous est réalisable dans un temps de $\frac{(u+2)n}{2u(u+1)} - \frac{n(u-2)}{2u(u^2-1)(u+1)^k}$. Les auteurs établissent aussi que cet algorithme est optimal dans le cas de $u \leq 2$.

Enfin, dans le cas de l'environnement des graphes arbitraires, l'article de Guilbault et Pelc [18] présente un algorithme de rendez-vous pour un graphe inconnu et connexe. Les deux agents sont anonymes, asynchrones, et l'adversaire contrôle leurs vitesses, qui peuvent varier. Les auteurs prouvent que le rendez-vous est faisable pour toutes les positions initiales avec la probabilité 1.

Il existe un autre type de rendez-vous, qui porte le nom de rassemblement, où plusieurs agents doivent se réunir dans un nœud. Ooshita et al. [24] ont considéré un modèle dans un anneau anonyme unidirectionnel. Les agents ne connaissent ni leur nombre ni le nombre de nœuds, mais ces derniers sont équipés d'un tableau de bord disponible en lecture et en écriture. Pour une probabilité constante donnée p, les auteurs prouvent qu'il n'existe aucun algorithme aléatoire qui résout le problème de rassemblement pour $0 < p \leq 1$ avec détection. Afin d'alléger le modèle, les auteurs considèrent le scénario sans détection. Ils ont trouvé un algorithme qui résout le problème pour chaque

constante $0 < p < 1$ où chaque nœud a besoin de $O(\log n)$ bits de mémoire pour le tableau de bord, et chaque agent a besoin de $O(n)$ bits de mémoire pour se déplacer $O(n)$ fois. Finalement, les auteurs prouvent qu'il n'existe aucun algorithme qui résout le problème de rassemblement avec probabilité 1 sans détection. Le tableau suivant montre la faisabilité du rassemblement selon les différents scénarios.

	avec détection	sans détection
prob. p ($0 < p < 1$)	impossible	possible
prob. 1	impossible	impossible

TABLE 2.1 – Solutions de rassemblement dans [24]

2.5 Rendez-vous en présence de pannes

Après avoir traité les articles sur le rendez-vous dans un environnement sain, cette dernière section de la revue de la littérature aborde le problème du rendez-vous dans un environnement avec pannes. Divers types de pannes ont été considérés. Plusieurs auteurs se sont penchés sur le problème des trous noirs (un processus stationnaire situé à un nœud qui détruit tout agent entrant sans laisser de traces), des arêtes défectueuses (un processus stationnaire situé à une arête qui détruit tout agent traversant sans laisser de traces), des agents byzantins (des agents qui se comportent différemment de ce qui est prévu dans l'algorithme) ou des agents qui sont soumis à des défauts de retard par un adversaire.

Dobrev et al. [14] ont envisagé le problème du rendez-vous et le problème du rassemblement proche (qui consiste à avoir au moins k agents à distance d) avec la présence d'un trou noir dans un anneau anonyme de n nœuds et k agents mobiles. Les nœuds sont équipés d'un tableau de bord d'une mémoire limitée. Les auteurs ont cherché à

estimer le nombre maximal d'agents qui peuvent se rencontrer. Ils ont considéré quatre scénarios :

- Si le nombre n est inconnu et que l'anneau est orienté, le rendez-vous de $k-1$ agents mobiles est résolu.

- Si le nombre n est connu et que l'anneau est orienté, le rendez-vous de $k-2$ agents mobiles est résolu.

- Si le nombre n est inconnu et que l'anneau est non orienté, le rendez-vous de $k-2$ agents est résolu si k est impair ainsi que pour $\frac{k-2}{2}$ agents si k est pair.

- Si le nombre n est connu et que l'anneau est non orienté, le rendez-vous de $k-2$ agents est résolu si k est impair ou n est pair ainsi que pour $\frac{k-2}{2}$ agents si k est pair et n est impair.

Ces auteurs ont résolu aussi le problème du rassemblement proche de $k-2$ agents avec une distance de 1 pour les deux derniers scénarios.

Chalopin et al. [4] ont fait une généralisation des études sur les réseaux de topologie arbitraire qui contiennent des pannes. Les auteurs abordent le problème du rendez-vous d'un ensemble de k agents mobiles dans un graphe connexe non orienté contenant certaines arêtes défectueuses. Les nœuds sont équipés d'un tableau de bord pour lecture et écriture. Les auteurs considèrent le principe d'exclusion mutuelle pour l'accès aux tableaux de bord (au plus un agent peut accéder au tableau de bord d'un nœud en même temps). Les auteurs montrent que s'il y a c liens défectueux, le rendez-vous des agents sera impossible si $k' > k-c$. Ils montrent un algorithme de rendez-vous de $k-c$ agents pour une certaine classe de réseaux. Le nombre total de mouvements des agents est de $O(m(m+k))$ où m est le nombre d'arêtes. Les auteurs prouvent que chaque algorithme de rendez-vous nécessite au moins $\Omega(m(m+k))$ mouvements où m est le nombre d'arêtes, même lorsque la topologie du graphe est connue a priori.

Dieudonné et al. [12] ont été les premiers auteurs à étudier le problème de rassemblement en présence d'un nombre f d'agents byzantins. Ils considèrent le modèle suivant : plusieurs agents placés sur différents nœuds dans un graphe connexe non orienté se déplacent d'une façon synchrone. Chaque agent connaît uniquement son étiquette et il ne peut marquer ni ports ni arêtes. Au plus f agents sont byzantins et le but est le rassemblement de tous les bons agents dans un nœud. Les auteurs ont résolu complètement le problème du rassemblement en présence d'agents byzantins faibles (les agents ne peuvent pas changer leurs étiquettes) et ils ont donné des résultats approximatifs pour les agents byzantins forts (les agents peuvent changer leurs étiquettes). L'article traite deux scénarios en cherchant le nombre minimal x de bons agents qui garantit leur rassemblement :

- Si les agents connaissent la taille n du graphe, les auteurs montrent que $x = 1$ en présence d'agents byzantins faibles. Aussi, ils donnent une borne supérieure $2f + 1$ et une borne inférieure $f + 1$ sur x en présence d'agents byzantins forts.

- Si les agents ne connaissent pas la taille n du graphe, les auteurs montrent que $x = f+2$ en présence d'agents byzantins faibles. Aussi, ils donnent une borne supérieure $4f + 2$ et une borne inférieure $f + 2$ sur x en présence d'agents byzantins forts.

Bouchard et al. [2] reprennent le même modèle pour résoudre complètement le problème du rassemblement en présence d'agents byzantins forts. Les auteurs prouvent que $f + 1$ bons agents peuvent se rencontrer s'ils connaissent la taille du graphe et que $f + 2$ bons agents peuvent se rencontrer s'ils ne connaissent pas la taille du graphe.

Chalopin et al. [5] observent le problème du rendez-vous de deux agents mobiles qui se déplacent en mode synchrone et qui sont soumis à des pannes de retard dans un réseau connexe non orienté. Les agents connaissent uniquement leurs propres étiquettes et leur mémoire est illimitée. Si un agent encourt une panne, il reste au nœud courant et il est conscient de ce fait. Les auteurs cherchent un algorithme déterministe de rendez-vous en

présence de pannes de retard. Le coût de l'algorithme consiste au nombre de traversées d'arêtes par les deux agents. Les auteurs prennent en compte trois scénarios de pannes :

- Aléatoire : Le retard est indépendant pour chaque ronde et pour chaque agent, avec une probabilité constante $0 < p < 1$. Les auteurs ont construit un algorithme de rendez-vous pour les réseaux arbitraires à coût polynomial en fonction de la taille n du réseau et polylogarithmique en fonction de la plus grande étiquette L, avec une très forte probabilité (plus précisément avec la probabilité d'au moins $1 - \frac{1}{n}$ où n est la taille du réseau).

- Pannes non bornées : L'adversaire peut retarder un agent pour un nombre fini arbitraire de rondes consécutives. Les auteurs montrent que le rendez-vous n'est pas réalisable même dans la classe des anneaux. Ils ont construit un algorithme de rendez-vous avec coût $O(nl)$ où l est la plus petite étiquette dans la classe des arbres arbitraires, et ils indiquent que l est la limite inférieure du coût de rendez-vous, même pour l'arbre à deux nœuds.

- Pannes bornées : L'adversaire peut retarder un agent pour au plus c rondes consécutives, où c est inconnu des agents. Les auteurs proposent un algorithme de rendez-vous qui fonctionne pour les réseaux arbitraires avec un coût polynomial en fonction de n, logarithmique en c et en l'étiquette L la plus grande.

Chapitre 3

Rendez-vous avec un jeton aux nœuds de départ

Sommaire

3.1 Introduction

Dans ce chapitre, nous considérons le scénario où chacun des agents pose un jeton stationnaire sur son nœud de départ. Les agents n'ont pas le droit d'utiliser d'autres jetons. Les deux jetons stationnaires sont de couleurs différentes. Les agents réalisent le rendez-vous dans un temps de $O(Exp + D \log L)$ où Exp est le temps d'exploration avec un jeton stationnaire, D est la distance initiale entre les agents et L est la grandeur de l'espace des étiquettes. Nous présentons le fonctionnement de l'algorithme, la preuve de l'exactitude, l'analyse de complexité et la borne inférieure sur le temps de rendez-vous. Cette borne montre que le temps de fonctionnement de notre algorithme est optimal.

3.2 Algorithme

Tout d'abord, les étiquettes des agents mobiles vont subir une transformation en doublant chaque bit de la représentation binaire de l'étiquette et en ajoutant les bits 10 au début et à la fin de l'étiquette (voir Figure 3.1).

Une étiquette transformée n'est jamais le préfixe d'une autre étiquette transformée, ce qui sera important pour le fonctionnement de notre algorithme.

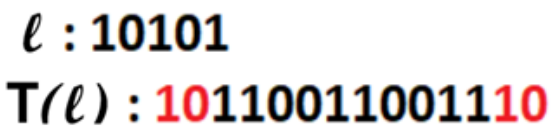

FIGURE 3.1 – Exemple de transformation d'une étiquette

Ensuite, chaque agent mobile va faire l'exploration du graphe avec un jeton stationnaire en appliquant l'algorithme BFS-Tree-Construction [3]. Après une première exploration, chaque agent mobile va obtenir une carte complète du graphe. La fin de cette étape se résume donc par une carte complète du graphe dans la mémoire de chaque agent avec la position de son propre jeton ainsi que du jeton de l'autre agent.

Puis chaque agent calcule la distance D entre les deux jetons et choisit le chemin p lexicographiquement le plus petit parmi tous les chemins de longueur D entre son jeton et celui de l'autre agent. Chaque chemin, d'un nœud A au nœud B, est codé par la suite par des numéros de ports utilisés dans le parcours du chemin.

Chaque agent mobile va faire des allers-retours sur le chemin choisi jusqu'à la réalisation du rendez-vous, où tous les agents seront sur le même nœud à la même ronde.

Supposons que l'agent avec étiquette l a trouvé le chemin p. Le chemin rebours au chemin p est noté par $\bar{p}$. L'algorithme de l'agent fonctionne en phases de longueur de 2D rondes correspondant aux bits de l'étiquette transformée $T(l)$. Si ce bit est 0, l'agent

reste sur sa position initiale pendant toute la phase. Si ce bit est 1, l'agent traverse le chemin p et revient par le chemin $\bar{p}$ à son nœud de départ.

On peut modéliser ce trajet d'aller-retour comme un mouvement dans un anneau composé de chemins traversés par les deux agents tout en sachant que ces chemins peuvent être égaux (voir Figure 3.2).

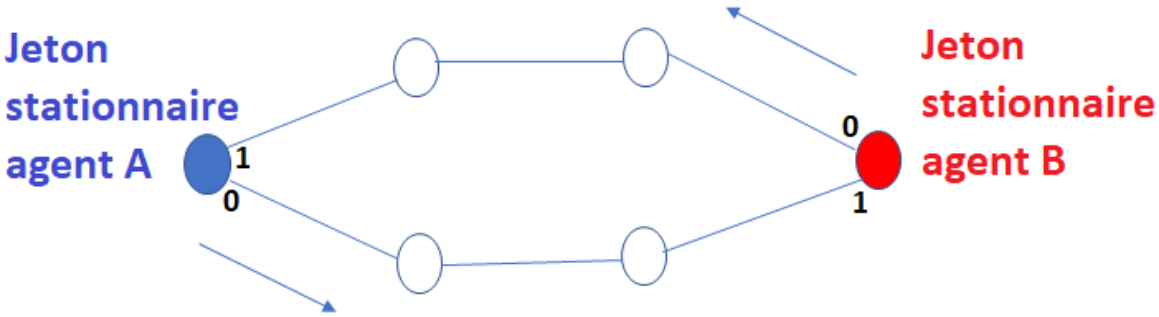

FIGURE 3.2 – Exemple du trajet Aller-retour

La procédure pour transformer l'étiquette est la suivante :

Procédure transformer-étiquette

Lire $l \leftarrow c_1 \dots c_k$

Retourner $T(l) \leftarrow 10c_1c_1 \dots c_kc_k10$

En utilisant cette procédure, l'algorithme Jeton-position-initiale qui permet de réaliser le rendez-vous des agents mobiles peut être formulé comme suit (l'algorithme est exécuté par un agent avec étiquette l et il est interrompu au moment du rendez-vous) :

Algorithme Jeton-position-initiale

Exécuter procédure transformer-étiquette

$T(l) \leftarrow d_1 \dots d_{2k+4}$

Exécuter BFS-Tree-Construction

Trouver la distance D du jeton de l'agent à l'autre jeton

Parmi les chemins de longueur D entre le jeton de l'agent et l'autre jeton, identifier le chemin p lexicographiquement le plus petit

Soit $i \leftarrow 1$

Répéter tant que $i <= 2k + 4$

Si $d_i = 1$ ***alors*** parcourir le chemin p et revenir à la position initiale par le chemin $\bar{p}$

Sinon rester immobile durant $2D$ rondes

$i \leftarrow i + 1$

3.3 Preuve de l'exactitude et analyse de complexité

Le résultat suivant justifie l'exactitude de notre algorithme et donne l'analyse de sa complexité.

Théorème 1. *L'algorithme Jeton-position-initiale résout le problème du rendez-vous dans un temps de $O(Exp + D \log L)$, où Exp est le temps d'exécution de la procédure BFS-Tree-Construction [3].*

Preuve. Considérons les cas suivants :

Blocs alignés : On parle de blocs alignés lorsque les phases de longueur $2D$ du premier agent mobile sont alignées avec les phases du deuxième agent, c'est-à-dire qu'un

agent commence sa première phase au début d'une phase de l'autre agent (voir Figure 3.3).

FIGURE 3.3 – Blocs alignés

On considère les cas suivants :

- Cas 1 :

Les dernières phases des agents finissent au même moment et leurs étiquettes ont la même longueur :

En sachant que les étiquettes transformées sont différentes, les agents mobiles vont se rencontrer pendant la première phase qui correspond aux bits différents, car un agent restera inerte sur sa position initiale pendant cette phase et l'autre visitera cette position initiale (voir Figure 3.4).

1 0 0 0 1 1 **0** 0 1 0

1 0 0 0 1 1 **1** 1 1 0

FIGURE 3.4 – Étiquettes de même longueur

- Cas 2 :

Les dernières phases des agents finissent au même moment et leurs étiquettes ont des longueurs différentes :

L'agent A avec l'étiquette plus courte commence plus tard, au début d'une phase de numéro impair du premier agent B. Les deux premières phases de l'agent

A correspondent aux bits 10 et les phases correspondantes de l'agent B correspondent soit aux bits 00, soit aux bits 11. Donc pendant une des deux premières phases de l'agent A, un des agents restera inerte sur sa position initiale et l'autre visitera cette position (voir Figure 3.5).

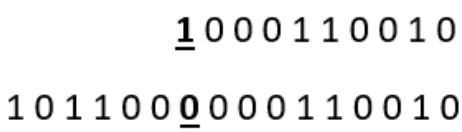

FIGURE 3.5 – Étiquettes de longueur différente

- Cas 3 :

Les agents ne finissent pas au même moment (départ décalé) et la fin de la dernière phase d'un agent a lieu au moment du début de la dernière phase de l'autre agent :

Les agents se rencontrent au plus tard pendant la dernière phase de l'agent qui finit plus tôt, car cette phase correspond au bit 0 de cet agent et au bit 1 de l'autre agent (voir Figure 3.6).

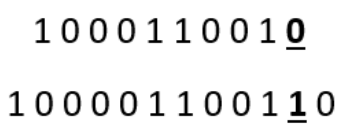

FIGURE 3.6 – Décalage d'un seul bit

- Cas 4 :

Les agents ne finissent pas au même moment (départ décalé) et après la fin de la dernière phase d'un des agents il y a au moins deux phases de l'autre agent :

La rencontre aura lieu au plus tard pendant l'avant-dernière phase de l'agent qui termine ses phases plus tard, car pendant cette phase un des agents restera inerte sur sa position initiale et l'autre visitera cette position (voir Figure 3.7).

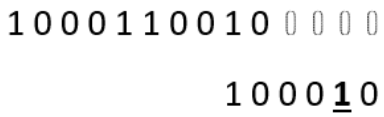

FIGURE 3.7 – Décalage de plusieurs bits

Blocs non alignés : On parle de blocs non alignés lorsque le début des phases d'un agent se trouve à l'intérieur des phases de l'autre agent.

Dans ce cas, pour chaque phase B d'un agent, il y a une phase B' pour l'autre agent, comme B et B' ont une intersection de longueur d'au moins D. Dans le cas où il y a deux possibilités, on prend en tant que B' la phase qui commence plus tôt. Appelons ces phases des phases jumelées. Il s'ensuit que pour un des agents les premières moitiés de ces phases sont entièrement incluses dans les phases jumelées de l'autre agent. Suivant le même raisonnement que pour les blocs alignés, il y a toujours une période de temps de longueur D pendant laquelle un agent reste sur sa position initiale et où l'autre visite cette position (voir Figure 3.8).

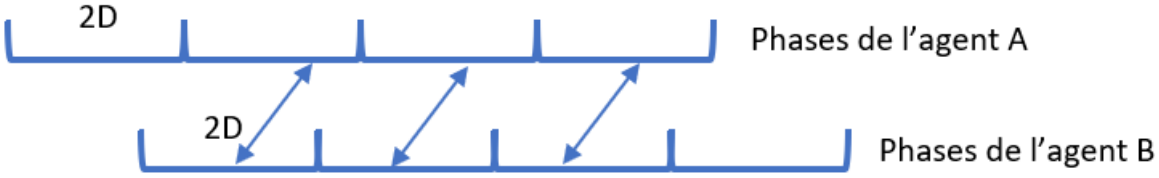

FIGURE 3.8 – Blocs non alignés. Les flèches montrent les blocs jumelés

Concernant le temps d'exécution de l'algorithme, nous avons mentionné auparavant que les agents mobiles font au début l'exploration avec l'algorithme BFS-Tree-

Construction. Les auteurs de [3] montrent que cet algorithme conçu pour l'exploration afin de construire la carte fonctionne dans un temps de $Exp = O(n^4)$.

Après l'exploration, les agents exécutent des phases des mouvements aller-retour entre leurs jetons stationnaires, ou restent immobiles selon le bit de leurs étiquettes transformées, en commençant par le premier bit jusqu'à la réalisation du rendez-vous. Chaque phase se fait dans un temps de $2D$ rondes et le nombre de phases est au plus la longueur de l'étiquette transformée. Puisque cette longueur est de $O(\log L)$, le temps d'exécution des phases est de $O(D \log L)$.

Le temps d'exécution de l'algorithme pour faire le rendez-vous est donc de $O(Exp + D \log L) = O(n^4 + D \log L)$. □

3.4 La borne inférieure

Après avoir prouvé que notre algorithme permet aux agents mobiles de réaliser le rendez-vous dans un temps de $O(Exp + D \log L)$, la question qu'il faut poser est si ce temps est optimal. Est-ce que le temps de $\Omega(D \log L)$ est nécessaire ? La réponse est oui, mais pas pour tous les graphes. Prenons l'exemple d'un graphe qui contient un sommet de degré unique. Après l'exploration, les agents mobiles peuvent se rencontrer à ce sommet. Pour certains graphes, on peut donc effectuer le rendez-vous dans un temps de $O(Exp)$.

Néanmoins, nous allons démontrer que le temps de rendez-vous de $\Theta(Opt - Explo + DlogL)$, où $Opt - Explo$ est le meilleur temps d'exploration dans le pire cas et qui ne peut pas être amélioré pour la classe de tous les graphes. Observons d'abord que le temps de $\Omega(Opt - Explo)$ est nécessaire. Pour le faire, il suffit de montrer que le rendez-vous ne peut pas être accompli dans un temps meilleur que l'exploration. Supposons que

oui, et soit OPT-EXPLO l'algorithme optimal d'exploration. Cet algorithme travaille en temps $Opt - Explo$ pour un certain graphe. Soit A_1 l'agent qui commence l'algorithme de rendez-vous le premier. Il existe un nœud v qu'il va visiter seulement après le temps $Opt - Explo$. L'adversaire peut choisir v comme position initiale de l'agent A_2 et retarder son démarrage jusqu'au moment de la visite de v par A_1. Ainsi le rendez-vous prendra le temps d'au moins $Opt - Explo$.

La proposition suivante montre que le temps de $\Omega(DLogL)$ est aussi nécessaire pour certains graphes.

Proposition 2. *Le temps de n'importe quel algorithme déterministe qui permet le rendez-vous à deux agents mobiles avec jetons stationnaires dans leurs positions initiales est de* $\Omega(DlogL)$ *pour la classe des anneaux.*

Preuve. Considérons deux agents mobiles placés à une distance initiale D divisible par 6 dans deux nœuds différents d'un anneau orienté de taille $2D$ où les ports dans chaque nœud sont numérotés $0, 1$ dans le sens horaire. Divisons l'anneau en segments de taille $\frac{D}{6}$ chacun. Soit A un algorithme déterministe qui permet de faire le rendez-vous dans un temps T d'au plus $\frac{D}{6 \log 3} \log L$. Les agents exécutent A avec un départ simultané. Nous partitionnons le temps en périodes consécutives de longueur $\frac{D}{6}$. On considère le *code de comportement* [9] d'un agent mobile. Ce code représente une suite d'éléments de l'ensemble $\{-1, 0, 1\}$ défini de la manière suivante : le i-ème terme du code de comportement est -1 si l'agent termine la période i dans le segment précédant (dans le sens horaire) celui où il a commencé la période, 1 s'il termine la période i dans le segment suivant celui où il a commencé la ronde, et 0 s'il termine la période i dans le même segment où il a commencé la période. Notons que ce sont les seules possibilités, l'agent ne peut pas terminer une période dans un segment S' séparé par un autre segment du segment S où il l'a commencée. Soit $\{1, ..., L\}$ l'ensemble des étiquettes et soit Z la

taille du code de comportement généré par un agent. Alors $Z < \frac{T}{\frac{D}{6}} < \frac{DlogL}{6log3.\frac{D}{6}} = \frac{logL}{log3}$, d'où $3^Z < L$. Il existe donc au moins deux étiquettes différentes l_1 et l_2 qui génèrent le même code de comportement.

Au début, l'adversaire place les agents à distance D dans des segments antipodaux. Il leur donne des étiquettes l_1 et l_2. Puisque les agents ont le même code de comportement, ils seront dans des segments antipodaux au début de chaque période. Pendant une période, un agent peut seulement être dans le même segment ou dans le segment voisin de celui où il a commencé la période. Les agents ne vont donc jamais se rencontrer après avoir terminé l'exécution de A. Ça entre en contradiction avec le fait que A permet de faire le rendez-vous. Donc $T \geq \frac{1}{6 \log 3} D \log L$, ce qui implique que le temps du rendez-vous est de $\Omega(DlogL)$. □

Dans l'algorithme Jeton-position-initiale, nous pourrions remplacer la procédure BFS-Tree-Construction par une procédure d'exploration optimale $OPT-EXPLO$ travaillant dans un temps optimal $Opt-Explo$. La complexité de notre algorithme deviendrait alors $O(Opt - Explo + D \log L)$. La proposition 2 ainsi que la remarque qui la précède impliquent le corollaire suivant :

Corollaire 1. *Le temps optimal du rendez-vous avec jetons stationnaires sur les positions initiales des agents est de* $\Theta(Opt - Explo + D \log L)$*, où* $Opt - Explo$ *est le temps optimal d'exploration.*

Chapitre 4

Rendez-vous avec deux jetons aux nœuds de départ et au plus un dans les autres nœuds

Sommaire

4.1 Introduction

Dans ce chapitre, nous considérons le scénario où chacun des agents pose deux jetons stationnaires sur son nœud de départ et au plus un jeton sur les autres nœuds. Les agents possèdent des jetons de couleurs différentes. Ils réalisent le rendez-vous dans un temps de $O(e + D \log L)$, où e est le nombre d'arêtes du graphe, D est la distance initiale entre les agents et L est la grandeur de l'espace des étiquettes. Ce temps de rendez-vous est optimal. Nous présentons le fonctionnement de l'algorithme ainsi que la preuve de l'exactitude et l'analyse de complexité.

4.2 Algorithme

Tout d'abord, les étiquettes des agents mobiles vont subir une transformation au moyen de la Procédure **transformer-étiquette**.

Ensuite, chaque agent mobile va faire l'exploration du graphe en utilisant le DFS, contrairement au cas précédent, car ils peuvent poser au plus un jeton par nœud. Après une première exploration, chaque agent mobile va obtenir une carte complète du graphe avec les positions initiales des agents.

Puis, chaque agent va suivre les étapes de l'algorithme **Jeton-position-initiale** à partir de la ligne 4 jusqu'à la fin.

En utilisant les jetons, nous formulons la procédure suivante pour le parcours en profondeur DFS :

Procédure DFS (z)

Soit d le degré du nœud z

Mettre un jeton sur le nœud z

Pour i de 0 à $d-1$ faire

    Prendre le port i pour aller au nœud v

    Si le nœud v ne contient pas de jeton de ma couleur

        Faire DFS(v)

Revenir au nœud z

Pendant l'exécution de la procédure DFS, l'agent va avoir dans sa mémoire la carte du graphe en traçant les arêtes qu'il a parcourues.

En utilisant cette procédure, l'algorithme **Deux-jetons-position-initiale** qui permet de réaliser le rendez-vous des agents mobiles peut être formulé comme suit. L'algorithme est exécuté par un agent avec étiquette l et il est interrompu au moment du rendez-vous.

Algorithme Deux-jetons-position-initiale

Exécuter procédure transformer-étiquette

$T(l) \leftarrow d_1 \dots d_{2k+4}$

Mettre un jeton sur la position initiale w

Exécuter DFS(w)

Identifier sur la carte deux jetons de l'autre couleur sur un nœud

Soit A le nœud initial de l'agent et B le nœud avec deux jetons de l'autre couleur

Trouver la distance D de A à B

Parmi les chemins de longueur D entre A et B identifier le chemin p lexicographiquement le plus petit

Soit $i \leftarrow 1$

Répéter tant que $i <= 2k + 4$

Si $d_i = 1$ ***alors*** parcourir le chemin p et revenir à la position initiale par le chemin $\bar{p}$

Sinon Rester immobile durant $2D$ rondes

$i \leftarrow i + 1$

4.3 Preuve de l'exactitude et analyse de complexité

Le résultat suivant justifie l'exactitude de notre algorithme et donne l'analyse de sa complexité.

Théorème 3. *L'algorithme Deux-jetons-position-initiale résout le problème du rendez-vous dans un temps de $O(e + D \log L)$, où e est le nombre d'arêtes du graphe.*

Preuve. Cette preuve est semblable à celle du théorème 1. La seule chose qui change est le temps de l'exploration. L'algorithme **Deux-jetons-position-initiale** utilise le DFS pour explorer le graphe. Cette exploration se fait en temps $O(e)$.

Le temps d'éxécution de cet algorithme pour faire le rendez-vous est donc de $O(e + D \log L)$. □

Comme dans le scénario du chapitre 3, le rendez-vous ne peut pas être accompli plus rapidement que l'exploration. D'autre part, nous faisons l'observation suivante :

Proposition 4. *L'exploration d'un graphe inconnu prend un temps d'au moins e, où e est le nombre d'arêtes.*

Preuve. Considérons un algorithme d'exploration A. Considérons son éxécution dans le graphe G. Il suffit de montrer que chaque arête de G doit être traversée pendant l'exploration. Supposons que l'arête a n'est pas traversée. Soit $G*$ le graphe qui résulte de G par l'ajout d'un nouveau sommet v du degré 2 au milieu de l'arête a. L'algorithme A aurait la même éxécution dans $G*$ que dans G, ce qui implique que l'agent ne visiterait jamais le sommet v dans $G*$, donc A ne serait pas un algorithme correct d'exploration. □

Le corollaire suivant découle du théorème 3, de la proposition 2 du chapitre précédent et de la proposition 4.

Corollaire 2. *Le temps optimal du rendez-vous avec deux jetons stationnaires sur les positions initiales des agents et au plus un jeton de chaque agent sur les autres nœuds est de* $\Theta(e + D \log L)$.

Chapitre 5

Rendez-vous avec un nombre illimité de jetons aux nœuds de départ et au plus un jeton dans les autres nœuds

Sommaire

5.1 Introduction

Dans ce chapitre, nous considérons le scénario où chacun des agents pose un nombre illimité de jetons sur son nœud de départ et au plus un jeton sur les autres nœuds. Les agents possèdent des jetons de couleurs différentes. Ils réalisent le rendez-vous dans un temps de $O(e)$, où e est le nombre d'arêtes du graphe, ce qui est optimal. Nous présentons le fonctionnement de l'algorithme ainsi que la preuve de l'exactitude et l'analyse de complexité.

5.2 Algorithme

Tout d'abord, chaque agent mobile pose le nombre de jetons qui représente la valeur de son étiquette sur son nœud de départ. Ensuite, chaque agent va faire l'exploration du graphe en utilisant le DFS.

Après une première exploration, chaque agent mobile va obtenir une carte complète du graphe avec les positions initiales des agents, ce qui va lui permettre de connaître la valeur de l'étiquette de l'autre.

Puis, l'agent qui a la plus grande étiquette rejoint la position initiale de l'autre agent par le chemin le plus court lexicographiquement le plus petit pour réaliser le rendez-vous.

En utilisant la procédure DFS (voir chapitre 4), l'algorithme **Jetons-illimités-position-initiale** qui permet de réaliser le rendez-vous des agents mobiles peut être formulé comme suit. L'algorithme est exécuté par un agent avec étiquette l.

Algorithme Jetons-illimités-position-initiale

Mettre l jetons dans le nœud de départ

Exécuter DFS

Identifier sur la carte le nœud v avec plus que 1 jeton de l'autre couleur

Soit x le nombre de jetons de l'autre couleur sur le nœud v

Si $x > l + 1$ alors rester sur le nœud de départ

Sinon aller vers v par le chemin le plus court lexicographiquement le plus petit

5.3 Preuve de l'exactitude et analyse de complexité

Le résultat suivant justifie l'exactitude de notre algorithme et donne l'analyse de sa complexité.

Théorème 5. *L'algorithme Jetons-illimités-position-initiale résout le problème du rendez-vous dans un temps de $\Theta(e)$, où e est le nombre d'arêtes du graphe.*

Preuve. Soient A et B deux agents mobiles avec des étiquettes respectives L et l telles que $l < L$. Sur leurs propres positions initiales, A met L jetons rouges et B met l jetons bleus. Ensuite, A et B commencent à exécuter le DFS. Une ronde après son réveil, A va avoir $x = L + 1$ jetons rouges dans sa position initiale et B va avoir $y = l + 1$ jetons bleus dans sa position initiale (le réveil des agents peut se produire dans la même ronde ou dans des rondes différentes). À la fin de l'exécution du DFS, les agents vont obtenir la carte complète du graphe. À l'aide de cette carte, A va identifier le nœud qui contient plus qu'un jeton bleu et B va identifier le nœud qui contient plus qu'un jeton rouge. Nous avons $x = L + 1 > l + 1$ et $y = l + 1 < L + 1$, alors B reste sur sa position initiale et A rejoint la position initiale de B par le chemin le plus court lexicographiquement le plus petit.

Pour ce qui est du temps d'exécution, l'exploration se fait en temps $\Theta(e)$ et le déplacement de l'agent ayant la plus grande étiquette se fait en temps $O(n)$ qui est $O(e)$.

□

La proposition 4 du chapitre 4 implique que le temps de fonctionnement de notre algorithme est optimal.

Chapitre 6

La simulation

Sommaire

6.1 Documentation du logiciel de simulation

Dans cette section, nous expliquons le fonctionnement du logiciel de simulation pour l'algorithme Jetons-illimités-position-initiale. Ce logiciel est une application Windows réalisée avec Microsoft Visual Studio 2017 dans le langage de programmation *C#* en utilisant le framework .NET 4.6 et la librairie PRISME, qui gère l'affichage dans l'interface pour les applications WPF. Nous avons utilisé le modèle de programmation Multithreading pour simuler le déplacement synchrone des agents mobiles.

L'interface du simulateur contient les sections suivantes :

- Section pour dessiner le graphe en utilisant les actions suivantes :

  + clique droite de la souris sur la section du dessin : ajouter un nœud

  + clique gauche de la souris sur deux nœuds : ajouter une arête

  + clique droite de la souris sur un nœud : ajouter un agent

- Section Nœud pour voir les informations de chaque nœud comme le nombre de jetons sur le nœud (voir Figure 6.1)

FIGURE 6.1 – Section nœud du simulateur

- Section Agent pour saisir les informations de chaque agent comme l'étiquette et le délai de départ (voir Figure 6.2)

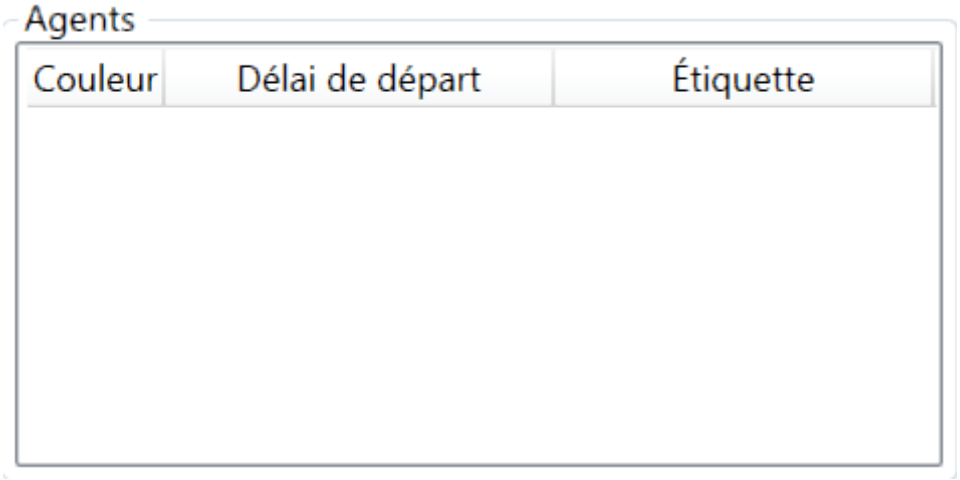

FIGURE 6.2 – Section agent du simulateur

- Section Solution qui contient une zone de texte pour voir les différentes étapes de réalisation du rendez-vous, un bouton pour effacer le graphe dessiné et un bouton pour réinitialiser les paramètres sans effacer le graphe.

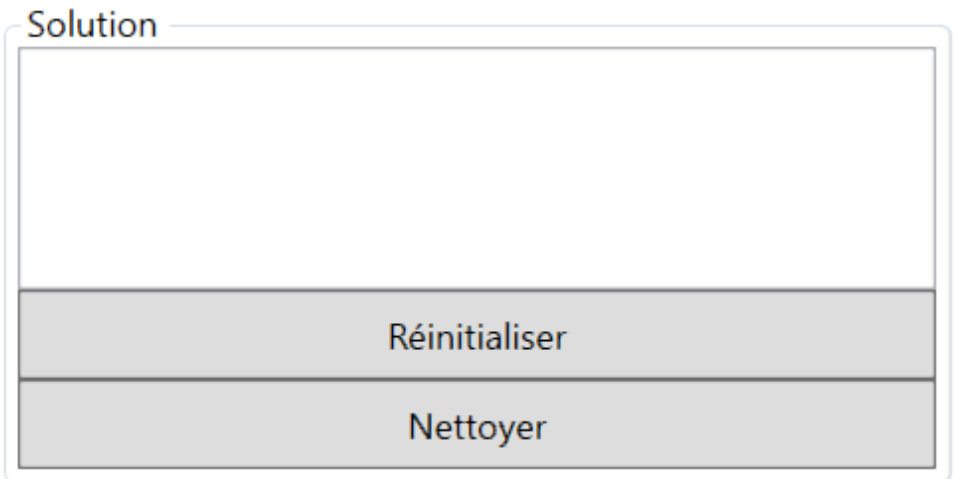

FIGURE 6.3 – Section solution du simulateur

La figure 6.4 ci-dessous montre l'interface complète du simulateur.

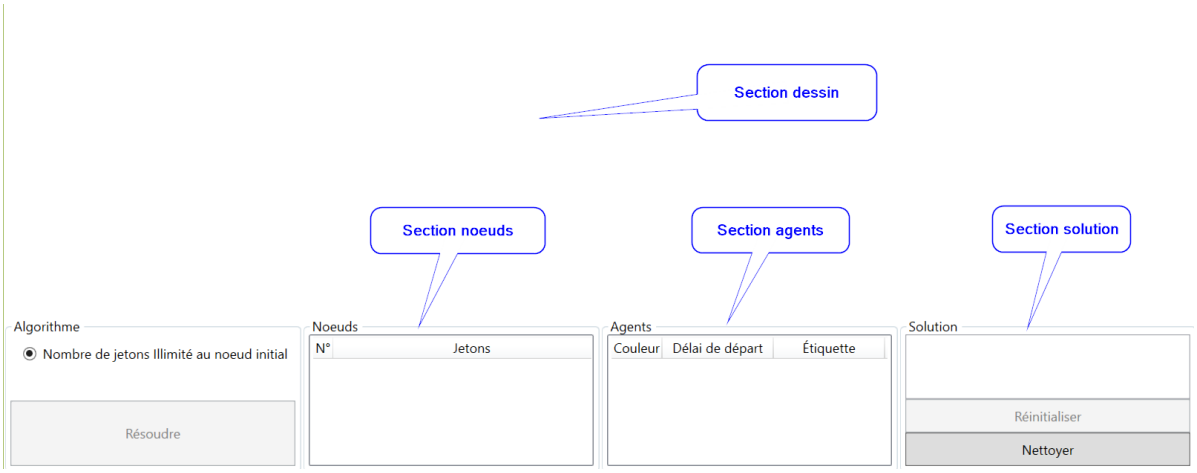

FIGURE 6.4 – L'interface du simulateur

6.2 Les étapes de la simulation

1- Dessiner le graphe et mettre les agents dans différents nœuds (voir Figure 6.7)

+ nous pouvons simuler un départ simultané (voir Figure 6.5).

+ nous pouvons simuler un départ différé (voir Figure 6.6).

2- Démarrer la simulation avec le bouton Résoudre

3- Consulter les étapes dans la section Solution (voir Figure 6.8)

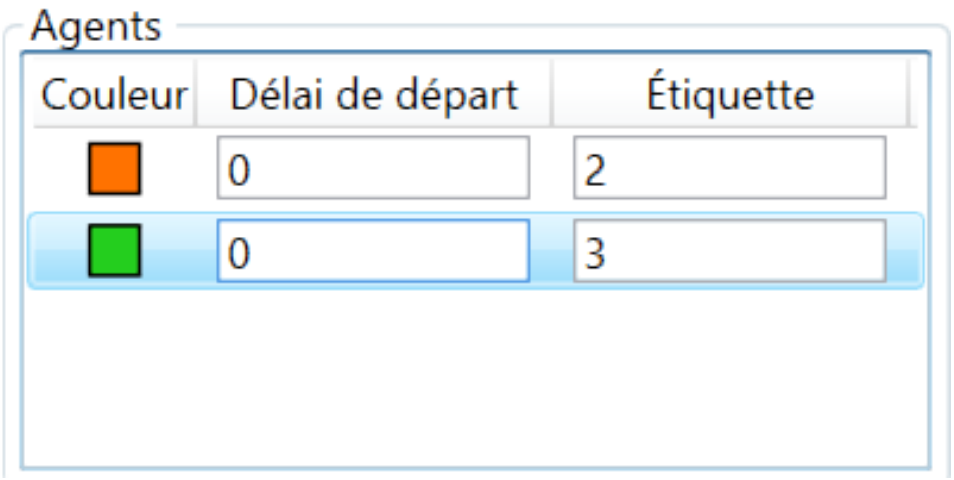

FIGURE 6.5 – Un départ simultané

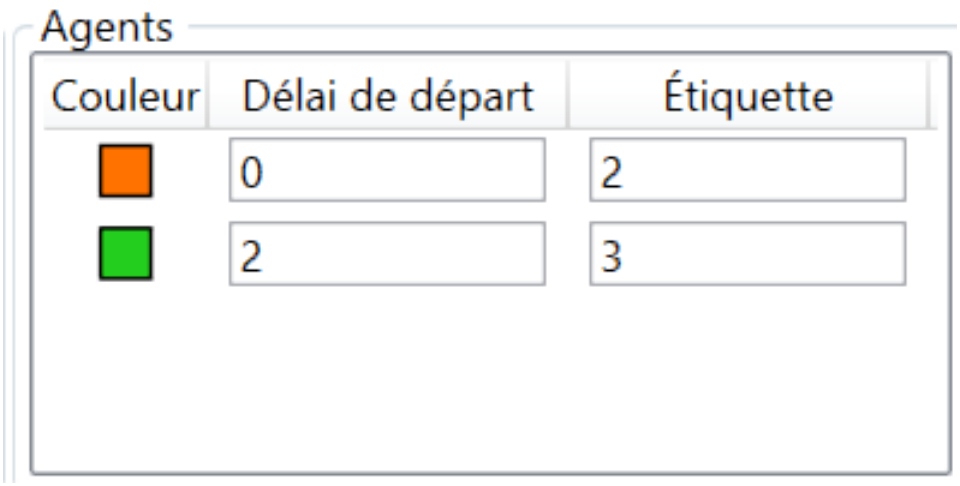

FIGURE 6.6 – Un départ différé

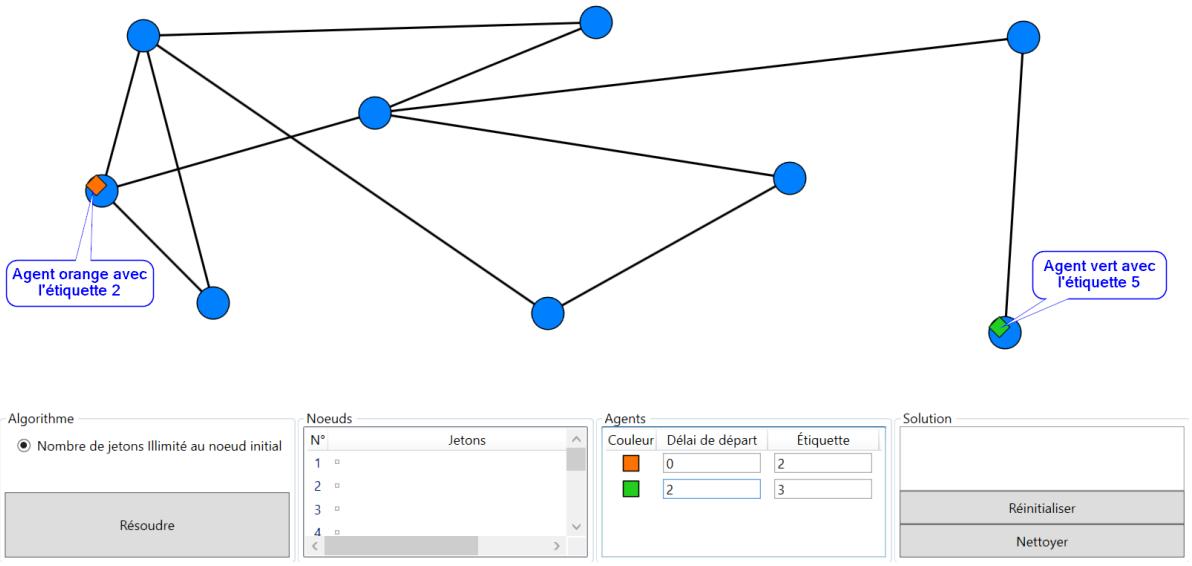

FIGURE 6.7 – Un graphe dessiné dans le simulateur

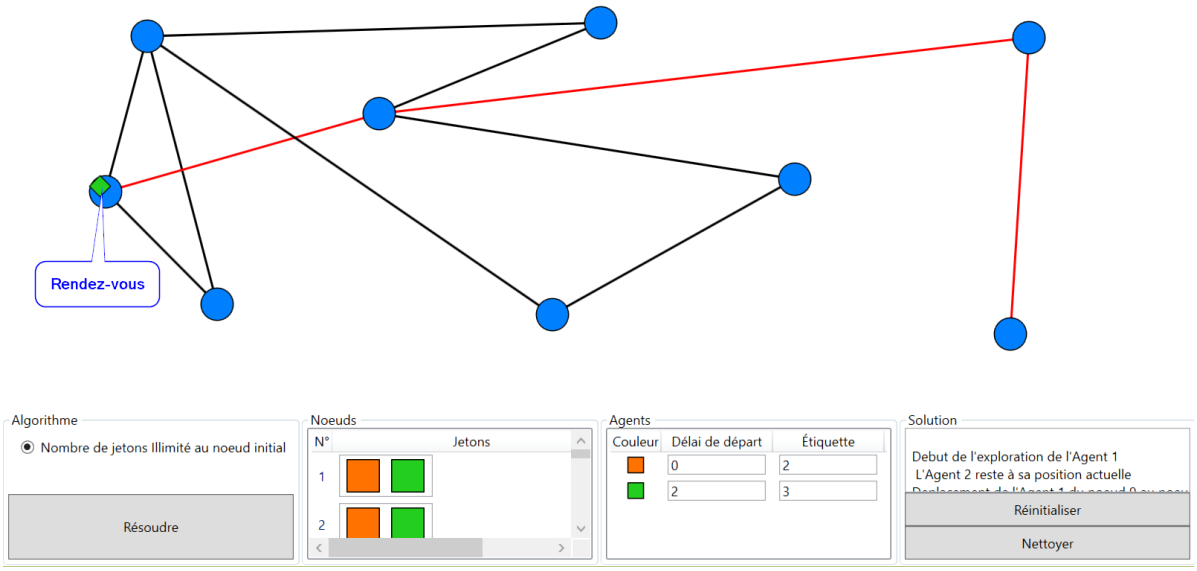

FIGURE 6.8 – Fin de la simulation

Chapitre 7

Conclusion

Nous avons conçu trois algorithmes qui résolvent le problème du rendez-vous dans les graphes en utilisant des jetons. Ces algorithmes diffèrent par le nombre de jetons utilisés par les agents ainsi qu'à la manière de les positionner dans le graphe :

- Dans le premier algorithme chaque agent met un jeton à sa position initiale.

- Dans le deuxième algorithme chaque agent met deux jetons à sa position initiale et un jeton au plus dans les autres noeuds.

- Dans le troisième algorithme chaque agent met un nombre illimité de jetons à sa position initiale et au plus un jeton dans les autres noeuds.

Nous avons prouvé l'exactitude, donné la complexité et démontré l'optimalité de chaque algorithme. Également, nous avons créé une application Windows pour simuler le troisième algorithme.

Dans ce mémoire, nous avons considéré le problème de rendez-vous avec jetons dans le modèle synchrone, où chaque agent peut traverser une seule arête dans une ronde. Il serait intéressant de généraliser le problème du rendez-vous avec jetons au modèle asynchrone, où la vitesse des agents sera arbitraire. Il nous semble que dans ce cas des techniques très différentes devraient être utilisées.

Annexe A

Code du simulateur

Listing A.1 – Class Agent

```
1  using GraphExpert.Data.Interfaces.Modeles;
2
3  namespace GraphExpert.Data.Modeles
4  {
5      /// <summary>
6      /// Class Agent
7      /// </summary>
8      public class Agent : IAgent
9      {
10         /// <summary>
11         /// Constructeur
12         /// </summary>
13         /// <param name="id"></param>
14         /// <param name="noeudId"></param>
15         /// <param name="etiquette"></param>
16         /// <param name="couleur"></param>
17         public Agent(byte id, byte noeudId, string etiquette, string
               couleur)
18         {
19             Id = id;
20             NoeudId = noeudId;
21             Etiquette = etiquette;
22             Couleur = couleur;
23         }
24
```

```
25          public byte Id { get; }
26
27          /// <summary>
28          /// L'identifiant du noeud sur laquelle l'agent se
                positionne
29          /// </summary>
30          public byte NoeudId { get; set; }
31
32          /// <summary>
33          /// L'etiquette de l'agent
34          /// </summary>
35          public string Etiquette { get; set; }
36
37          /// <summary>
38          /// La couleur de l'agent
39          /// </summary>
40          public string Couleur { get; set; }
41
42          /// <summary>
43          /// Le délai du départ de l'agent
44          /// </summary>
45          public int Delai { get; set; }
46      }
47  }
```

Listing A.2 – Class Arête

```
1
2   using GraphExpert.Data.Interfaces.Modeles;
3   using System.ComponentModel;
4
5   namespace GraphExpert.Data.Modeles
6   {
7       /// <summary>
8       /// Class Arête
9       /// </summary>
10      public class Arete : IArete, INotifyPropertyChanged
11      {
12          private byte _noeudIdDepart,
13              _portIdDepart,
14              _noeudIdArrivee,
```

```
15              _portIdArrivee;
16          private string _couleur;
17
18          /// <summary>
19          /// Constructeur
20          /// </summary>
21          /// <param name="noeudIdDepart"></param>
22          /// <param name="portIdDepart"></param>
23          /// <param name="noeudIdArrivee"></param>
24          /// <param name="portIdArrivee"></param>
25          /// <param name="couleur"></param>
26          public Arete(byte noeudIdDepart, byte portIdDepart, byte
                noeudIdArrivee, byte portIdArrivee, string couleur)
27          {
28              _noeudIdDepart = noeudIdDepart;
29              _portIdDepart = portIdDepart;
30              _noeudIdArrivee = noeudIdArrivee;
31              _portIdArrivee = portIdArrivee;
32              _couleur = couleur;
33          }
34
35          public byte PortIdDepart => _portIdDepart;
36
37          public byte PortIdArrivee => _portIdArrivee;
38
39          public byte NoeudIdDepart => _noeudIdDepart;
40
41          public byte NoeudIdArrivee => _noeudIdArrivee;
42
43          public string Couleur
44          {
45              get => _couleur;
46              set
47              {
48                  _couleur = value;
49                  PropertyChanged?.Invoke(this, new
                        PropertyChangedEventArgs("Couleur"));
50              }
51          }
52
53          public event PropertyChangedEventHandler PropertyChanged;
```

```
54      }
55  }
```

Listing A.3 – Class Exploration

```
1  using GraphExpert.Animations;
2  using GraphExpert.Data.Interfaces.Modeles;
3  using System;
4  using System.Collections.Generic;
5  using System.Linq;
6  using System.Text;
7  using System.Threading.Tasks;
8
9  namespace GraphExpert.Data.Modeles
10 {
11     /// <summary>
12     /// La class Exploration
13     /// </summary>
14     public class Exploration : IExploration
15     {
16         /// <summary>
17         /// Constructeur
18         /// </summary>
19         public Exploration()
20         {
21             ListDeplacement = new List<Deplacement>();
22             GrapheExplorer = new Graphe();
23             Commentaire = new List<string>();
24
25             RendezVous = false;
26
27         }
28         /// <summary>
29         /// Liste de déplacement de l'agent
30         /// </summary>
31         public List<Deplacement> ListDeplacement { get ; set; }
32         /// <summary>
33         /// Le graphe généré par l'agent après l'exploration
34         /// </summary>
35         public IGraphe GrapheExplorer { get; set ; }
36
```

```
37          /// <summary>
38          /// Liste de commentaire généré par l'agent
39          /// </summary>
40          public List<string> Commentaire { get; set; }
41
42          /// <summary>
43          /// L'état du rendez-vous
44          /// </summary>
45          public bool RendezVous { get; set; }
46
47          /// <summary>
48          /// Le nombre de ronde de l'exploration
49          /// </summary>
50          public int RondeExploration { get; set; }
51
52          /// <summary>
53          /// Liste des arêtes qui tracent le chemin le plus court
54          /// </summary>
55          public List<IArete> TracesRendezVous { get; set; }
56      }
57  }
```

Listing A.4 – Class Graphe

```
1  using GraphExpert.Data.Interfaces.Modeles;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace GraphExpert.Data.Modeles
9  {
10     public class Graphe : IGraphe
11     {
12         /// <summary>
13         /// Constructeur
14         /// </summary>
15         public Graphe()
16         {
17
```

```
18              noeuds = new List<INoeud>();
19              aretes = new List<IArete>();
20          }
21          /// <summary>
22          /// Liste des noeuds
23          /// </summary>
24          public List<INoeud> noeuds { get; set; }
25
26          /// <summary>
27          /// Liste des arêtes
28          /// </summary>
29          public List<IArete> aretes { get; set; }
30      }
31  }
```

Listing A.5 – Class Jeton

```
1
2  using GraphExpert.Data.Interfaces.Modeles;
3
4  namespace GraphExpert.Data.Modeles
5  {
6      public class Jeton : IJeton
7      {
8          /// <summary>
9          /// La couleur du jeton
10         /// </summary>
11         public string Couleur { get; set; }
12     }
13 }
```

Listing A.6 – Class Noeud

```
1
2  using GraphExpert.Data.Interfaces.Modeles;
3  using System.Collections.Generic;
4  using System.Collections.ObjectModel;
5
6  namespace GraphExpert.Data.Modeles
7  {
8      public class Noeud : INoeud
```

```
9       {
10          private Noeud()
11          {
12
13          }
14
15          /// <summary>
16          /// Constructeur
17          /// </summary>
18          /// <param name="id"></param>
19          /// <param name="etiquette"></param>
20          public Noeud(byte id, string etiquette)
21          {
22              Id = id;
23              Etiquette = etiquette;
24
25          }
26
27          public byte Id { get; }
28
29          /// <summary>
30          /// Étiquette du noeud
31          /// </summary>
32          public string Etiquette { get; }
33
34          /// <summary>
35          /// Le degré du noeud
36          /// </summary>
37          public int Degree { get; set; }
38
39          /// <summary>
40          /// La collection dynamique des jetons
41          /// </summary>
42          public ObservableCollection<IJeton> Jetons { get; set; } =
                new ObservableCollection<IJeton>();
43
44          /// <summary>
45          /// La liste temporaire des jetons
46          /// </summary>
47          public List<IJeton> JetonsT { get; set; } = new List<IJeton
                >();
```

```
48      }
49  }
```

Listing A.7 – Class Port

```
1
2  using GraphExpert.Data.Interfaces.Modeles;
3
4  namespace GraphExpert.Data.Modeles
5  {
6      public class Port : IPort
7      {
8          /// <summary>
9          /// Constructeur
10         /// </summary>
11         /// <param name="id"></param>
12         /// <param name="noeudId"></param>
13         public Port(byte id, byte noeudId)
14         {
15             Id = id;
16             NoeudId = noeudId;
17         }
18
19         public byte Id { get; }
20
21         /// <summary>
22         /// L'identifiant du noeud qui contient le port
23         /// </summary>
24         public byte NoeudId { get; set; }
25     }
26 }
```

Listing A.8 – Class RepoAgents

```
1
2  using GraphExpert.Data.Interfaces.Modeles;
3  using GraphExpert.Data.Interfaces.Repos;
4  using GraphExpert.Data.Modeles;
5  using System.Collections.Generic;
6  using System.Linq;
7
```

```
8  namespace GraphExpert.Data.Repos
9  {
10     /// <summary>
11     /// Repository des agents.
12     /// </summary>
13     public class RepoAgents : IRepoAgents
14     {
15         /// <summary>
16         /// Conteneur.
17         /// </summary>
18         private IList<IAgent> _agents = new List<IAgent>();
19
20         /// <summary>
21         /// Sélections de couleur.
22         /// </summary>
23         private string[] _couleurs = new string[]
24         {
25
26             "#FF7200",
27             "#24CE1E",
28             "#E9F418",
29             "#054ABA",
30             "#FF0000",
31             "#00A9FF"
32         };
33
34         /// <summary>
35         /// Sélecteur.
36         /// </summary>
37         private int _indexCouleurs = 0;
38
39         /// <summary>
40         /// Ajoute un nouvel agent.
41         /// </summary>
42         /// <param name="noeudId">Nř du noeud attaché.</param>
43         /// <param name="etiquette">Étiquette.</param>
44         /// <returns>Le nouvel agent.</returns>
45         public IAgent Ajouter(byte noeudId, string etiquette)
46         {
47             if (!((_indexCouleurs >= 0) && (_indexCouleurs <
                   _couleurs.Length)))
```

```
48              {
49                  return null;
50              }
51
52              var id = (byte)(_agents.Any() ? (_agents.Max(k => k.Id)
                    + 1) : 1);
53              var couleur = _couleurs[_indexCouleurs++];
54              var agent = new Agent(id, noeudId, etiquette, couleur);
55
56              _agents.Add(agent);
57
58              return agent;
59          }
60
61          /// <summary>
62          /// Obtenir tous les agents.
63          /// </summary>
64          /// <returns>Agents</returns>
65          public IEnumerable<IAgent> Obtenir() => _agents;
66
67          /// <summary>
68          /// Obtenir l'agent demandé.
69          /// </summary>
70          /// <param name="id">Nř de l'agent.</param>
71          /// <returns>Agent.</returns>
72          public IAgent Obtenir(byte id) => _agents.SingleOrDefault(p
                => p.Id == id);
73
74          /// <summary>
75          /// Retire l'agent en question.
76          /// </summary>
77          /// <param name="id">Identifiant de l'agent.</param>
78          public void Supprimer(byte id)
79          {
80              _agents.Remove(_agents.Single(p => p.Id == id));
81          }
82
83          /// <summary>
84          /// Vider les objets persistés.
85          /// </summary>
86          public void Vider()
```

```
87          {
88              _agents.Clear();
89              _indexCouleurs = 0;
90          }
91      }
92  }
```

Listing A.9 – Class RepoAretes

```
1
2  using GraphExpert.Data.Interfaces.Modeles;
3  using GraphExpert.Data.Interfaces.Repos;
4  using GraphExpert.Data.Modeles;
5  using System;
6  using System.Collections.Generic;
7  using System.Linq;
8
9  namespace GraphExpert.Data.Repos
10 {
11     /// <summary>
12     /// Repository des liaisons.
13     /// </summary>
14     public class RepoAretes : IRepoAretes
15     {
16         /// <summary>
17         /// Conteneur.
18         /// </summary>
19         private IList<IArete> _liaisons = new List<IArete>();
20
21         /// <summary>
22         /// Ajoute une nouvelle liaison.
23         /// </summary>
24         /// <param name="noeudIdDepart">Nř du noeud de départ.</
               param>
25         /// <param name="portIdDepart">Point de départ.</param>
26         /// <param name="noeudIdArrivee">Nř du noeud d'arrivée.</
               param>
27         /// <param name="portIdArrivee">Point d'arrivée.</param>
28         /// <returns>Nouvelle liaison.</returns>
29         public IArete Ajouter(byte noeudIdDepart, byte portIdDepart,
               byte noeudIdArrivee, byte portIdArrivee)
```

```
30          {
31              var liaison = Obtenir(noeudIdDepart, portIdDepart,
                    noeudIdArrivee, portIdArrivee);
32
33              if (null == liaison)
34              {
35                  liaison = new Arete(noeudIdDepart, portIdDepart,
                        noeudIdArrivee, portIdArrivee, "black");
36
37                  _liaisons.Add(liaison);
38              }
39
40              return liaison;
41          }
42
43          /// <summary>
44          /// Supprime la liaison.
45          /// </summary>
46          /// <param name="noeudIdDepart">Nř du noeud de départ.</
                param>
47          /// <param name="portIdDepart">Point de départ.</param>
48          /// <param name="noeudIdArrivee">Nř du noeud d'arrivée.</
                param>
49          /// <param name="portIdArrivee">Point d'arrivée.</param>
50          public void Supprimer(byte noeudIdDepart, byte portIdDepart,
                byte noeudIdArrivee, byte portIdArrivee)
51          {
52              var liaison = Obtenir(noeudIdDepart, portIdDepart,
                    noeudIdArrivee, portIdArrivee);
53
54              if (null == liaison) throw new Exception();
55
56              _liaisons.Remove(liaison);
57          }
58
59          /// <summary>
60          /// Obtenir une liaison.
61          /// </summary>
62          /// <param name="noeudIdDepart">Nř du noeud de départ.</
                param>
63          /// <param name="portIdDepart">Point de départ.</param>
```

```
64          /// <param name="noeudIdArrivee">Nř du noeud d'arrivée.</
                param>
65          /// <param name="portIdArrivee">Point d'arrivée.</param>
66          /// <returns>Liaison unique.</returns>
67          private IArete Obtenir(byte noeudIdDepart, byte portIdDepart
                , byte noeudIdArrivee, byte portIdArrivee) =>
68              _liaisons.SingleOrDefault(k => k.NoeudIdDepart ==
                    noeudIdDepart && k.PortIdDepart == portIdDepart && k.
                    NoeudIdArrivee == noeudIdArrivee && k.PortIdArrivee
                    == portIdArrivee);
69
70          /// <summary>
71          /// Vider les objets persistés.
72          /// </summary>
73          public void Vider()
74          {
75              _liaisons.Clear();
76          }
77
78          /// <summary>
79          /// Obtenir toutes les liaisons.
80          /// </summary>
81          /// <returns></returns>
82          public IEnumerable<IArete> Obtenir() => _liaisons;
83
84          /// <summary>
85          /// Obtenir une liaison.
86          /// </summary>
87          /// <param name="noeudIdDepart">Nř du noeud de départ.</
                param>
88          /// <param name="portIdDepart">Point de départ.</param>
89          /// <returns>Liaison unique.</returns>
90          public IArete Obtenir(byte noeudIdDepart, byte portIdDepart)
                => _liaisons.SingleOrDefault(p => p.NoeudIdDepart ==
                noeudIdDepart && p.PortIdDepart == portIdDepart);
91      }
92  }
```

Listing A.10 – Class RepoNoeuds

```
1
```

```
2  using GraphExpert.Data.Interfaces.Modeles;
3  using GraphExpert.Data.Interfaces.Repos;
4  using GraphExpert.Data.Modeles;
5  using System.Collections.Generic;
6  using System.Linq;
7
8  namespace GraphExpert.Data.Repos
9  {
10     /// <summary>
11     /// Repository des arrêts.
12     /// </summary>
13     public class RepoNoeuds : IRepoNoeuds
14     {
15         /// <summary>
16         /// Conteneur.
17         /// </summary>
18         private IList<INoeud> _noeuds = new List<INoeud>();
19
20         /// <summary>
21         /// Ajoute un nouvel arrêt.
22         /// </summary>
23         /// <param name="etiquette">Étiquette.</param>
24         /// <returns>Le nouvel arrêt.</returns>
25         public INoeud Ajouter(string etiquette)
26         {
27             var arret = new Noeud((_noeuds.Any() ? (byte)(_noeuds.
                   Max(k => k.Id) + 1) : (byte)1), etiquette);
28
29             _noeuds.Add(arret);
30
31             return arret;
32         }
33
34         /// <summary>
35         /// Obtenir tous les arrêts.
36         /// </summary>
37         /// <returns>Arrêts</returns>
38         public IEnumerable<INoeud> Obtenir() => _noeuds;
39
40         /// <summary>
41         /// Obtenir le noeud demandé.
```

```
42          /// </summary>
43          /// <param name="id">Nř du noeud.</param>
44          /// <returns>Noeud.</returns>
45          public INoeud Obtenir(byte id) => _noeuds.SingleOrDefault(p
                => p.Id == id);
46
47          /// <summary>
48          /// Retire l'arrêt en question.
49          /// </summary>
50          /// <param name="id">Identifiant de l'arrêt.</param>
51          public void Supprimer(byte id)
52          {
53              _noeuds.Remove(_noeuds.Single(p => p.Id == id));
54          }
55
56          /// <summary>
57          /// Vider les objets persistés.
58          /// </summary>
59          public void Vider()
60          {
61              _noeuds.Clear();
62          }
63      }
64  }
```

Listing A.11 – Class RepoPorts

```
1
2  using GraphExpert.Data.Interfaces.Modeles;
3  using GraphExpert.Data.Interfaces.Repos;
4  using GraphExpert.Data.Modeles;
5  using System.Collections.Generic;
6  using System.Linq;
7
8  namespace GraphExpert.Data.Repos
9  {
10     /// <summary>
11     /// Repository des ports.
12     /// </summary>
13     public class RepoPorts : IRepoPorts
14     {
```

```
15          /// <summary>
16          /// Conteneur.
17          /// </summary>
18          private IList<IPort> _ports = new List<IPort>();
19
20          /// <summary>
21          /// Ajoute un port au noeud.
22          /// </summary>
23          /// <param name="noeudId">Nř du noeud attaché.</param>
24          /// <returns>Port.</returns>
25          public IPort Ajouter(byte noeudId)
26          {
27              var portsMemeNoeud = _ports.Where(p => p.NoeudId ==
                    noeudId);
28              byte id = 1;
29
30              if (portsMemeNoeud.Any())
31              {
32                  id = (byte)(portsMemeNoeud.Max(k => k.Id) + 1);
33              }
34
35              var port = new Port(id, noeudId);
36
37              _ports.Add(port);
38
39              return port;
40          }
41
42          /// <summary>
43          /// Obtenir tous les ports.
44          /// </summary>
45          /// <returns>Liste des ports.</returns>
46          public IEnumerable<IPort> Obtenir() => _ports;
47
48          /// <summary>
49          /// Obtenir les ports pour le noeud en paramètre.
50          /// </summary>
51          /// <param name="noeudId">Nř du noeud.</param>
52          /// <returns>Ports disponibles.</returns>
53          public IEnumerable<IPort> ObtenirPourNoeud(byte noeudId) =>
                _ports.Where(p => p.NoeudId == noeudId).ToList();
```

```
54
55          /// <summary>
56          /// Supprimer le port.
57          /// </summary>
58          /// <param name="id">Nř du port.</param>
59          /// <param name="noeudId">Nř du noeud attaché.</param>
60          public void Supprimer(byte id, byte noeudId)
61          {
62              _ports.Remove(_ports.Single(p => p.Id == id && p.NoeudId
                    == noeudId));
63          }
64
65          /// <summary>
66          /// Vider les objets persistés.
67          /// </summary>
68          public void Vider()
69          {
70              _ports.Clear();
71          }
72      }
73  }
```

Listing A.12 – Class Deplacement

```
1
2   namespace GraphExpert.Animations
3   {
4       public class Deplacement : IDeplacement
5       {
6           /// <summary>
7           /// Constructeur
8           /// </summary>
9           /// <param name="agentId"></param>
10          /// <param name="portId"></param>
11          public Deplacement(byte agentId, byte portId)
12          {
13              AgentId = agentId;
14              PortId = portId;
15          }
16
17          /// <summary>
```

```
18          /// l'identifiant de l'agent
19          /// </summary>
20          public byte AgentId { get; }
21
22          /// <summary>
23          /// L'identifiant du port par lequel l'agent va se déplacer
24          /// </summary>
25          public byte PortId { get; }
26      }
27  }
```

Listing A.13 – Class AlgorithmeExplorationDfs

```
1
2  using GraphExpert.Algorithmes.Interfaces;
3  using GraphExpert.Animations;
4  using GraphExpert.Data.Interfaces.Modeles;
5  using GraphExpert.Data.Interfaces.Repos;
6  using GraphExpert.Data.Modeles;
7  using System;
8  using System.Collections.Generic;
9  using System.Collections.ObjectModel;
10 using System.Linq;
11 using System.Threading;
12 using System.Threading.Tasks;
13
14 namespace GraphExpert.Algorithmes
15 {
16     public class AlgorithmeExplorationDfs :
           IAlgorithmeExplorationDfs
17     {
18
19         /// <summary>
20         /// Variable de travail
21         /// </summary>
22         public string resultat = string.Empty;
23         public List<Deplacement> deplacement = new List<Deplacement
               >();
24
25         /// <summary>
26         /// Faire le DFS pour explorer le graphe
```

```
27          /// </summary>
28          /// <param name="agentID"></param>
29          /// <param name="idPort"></param>
30          /// <param name="repoAgent"></param>
31          /// <param name="repoNoeud"></param>
32          /// <param name="repoArete"></param>
33          /// <param name="g"></param>
34          /// <param name="commentaire"></param>
35          /// <param name="nombreTask"></param>
36          /// <param name="listTask"></param>
37          /// <returns></returns>
38          public Exploration DFS(byte agentID, byte idPort,
                IRepoAgents repoAgent, IRepoNoeuds repoNoeud, IRepoAretes
                 repoArete, IGraphe g, List<string> commentaire,
                Dictionary<int, int> nombreTask, Dictionary<Task, byte>
                listTask)
39          {
40
41              ///Variables
42              Exploration retour = new Exploration();
43              var agent = repoAgent.Obtenir(agentID);
44              var noeudAgentActuelle = repoNoeud.Obtenir().ToList().
                    Where(p => p.Id == agent.NoeudId).FirstOrDefault();
45              IJeton jet = new Jeton { Couleur = agent.Couleur };
46              noeudAgentActuelle.JetonsT.Add(jet);
47              INoeud noeudDepartGraphe = noeudAgentActuelle;
48              var Delai = repoAgent.Obtenir().Where(c => c.Delai != 0)
                    ;
49              var delaiAg = Delai.Count() != 0 ? Delai.Max(obj => obj.
                    Delai) : 0;
50              IExploration s;
51              int degre;
52              //Ajouter le noeud de départ de l'agent au graphe que ce
                     dernier va générer
53              if (!ValiderNoeud(noeudDepartGraphe, g))
54              {
55                  g.noeuds.Add(noeudDepartGraphe);
56
57              }
58
59              lock (repoNoeud)
```

```
60              {
61                  degre = repoNoeud.Obtenir().ToList().Where(p => p.Id
                        == agent.NoeudId).FirstOrDefault().Degree;
62              }
63
64              ///Parcourir chaque port du noeud pour l'exploration
65              for (byte i = 1; i <= degre; i++)
66              {
67
68                  if (i != idPort)
69                  {
70
71                      INoeud noeudArrive = null;
72                      IEnumerable<IArete> artes;
73                      lock (repoArete)
74                      {
75                          artes = repoArete.Obtenir().ToList().Where(n
                                => n.NoeudIdDepart == agent.NoeudId && n
                               .PortIdDepart == i);
76
77                      }
78
79                      var mesJetons = 0;
80                      var mesJeton = repoNoeud.Obtenir(artes.
                           FirstOrDefault().NoeudIdArrivee).JetonsT;
81                      lock (mesJeton)
82                      {
83                          mesJetons = mesJeton.Where(u => u.Couleur ==
                                agent.Couleur).Count();
84                      }
85                      ///Vérifier si l'arête que l'agent va traverser
                           existe
86                      if (artes.Count() != 0 && mesJetons == 0)
87                      {
88
89                          var arete = artes.FirstOrDefault();
90
91
92                          byte noeudIdDepart = new byte();
93                          noeudIdDepart = noeudAgentActuelle.Id;
94                          var portIdArrive = new byte();
```

```
95                          portIdArrive = artes.FirstOrDefault().
                                PortIdArrivee;
96                          var portIdDepart = artes.FirstOrDefault().
                                PortIdDepart;
97                          var noeudIdArrive = artes.FirstOrDefault().
                                NoeudIdArrivee;
98
99                          agent.NoeudId = noeudIdArrive;
100                         lock (repoNoeud)
101                         {
102                             noeudArrive = repoNoeud.Obtenir().ToList
                                    ().Where(p => p.Id == agent.NoeudId).
                                    FirstOrDefault();
103
104                         }
105
106                         if(!RendezVous.RV )
107                         {
108                             ///Faire un déplacement si le noeud
                                    visité ne contient pas de jeton
109                             deplacement.Add(new Deplacement(agent.Id
                                    , i));
110                             commentaire.Add(string.Format("\n
                                    Deplacement de l'Agent {0} du noeud
                                    {1} au noeud {2}", agent.Id,
                                    noeudIdDepart, agent.NoeudId));
111
112                         }
113
114
115                         nombreTask[Task.CurrentId.Value] =
                                nombreTask[Task.CurrentId.Value] + 1;
116
117                         IArete areteAllez = new Arete(arete.
                                NoeudIdDepart, arete.PortIdDepart, arete.
                                NoeudIdArrivee, arete.PortIdArrivee,
                                agent.Couleur);
118                         IArete areteRetour = new Arete(arete.
                                NoeudIdArrivee, arete.PortIdArrivee,
                                arete.NoeudIdDepart, arete.PortIdDepart,
                                agent.Couleur);
```

```
119                     ///Ajouter l'arête au graphe qui va être
                            générer par l'agent
120                     if (!ValiderArete(areteAllez, g))
121                     {
122                         g.aretes.Add(areteAllez);
123
124                     }
125                     //Ajouter l'arête au graphe qui va être
                            générer par l'agent
126                     if (!ValiderArete(areteRetour, g))
127                     {
128
129                         g.aretes.Add(areteRetour);
130
131                     }
132
133
134
135                     lock (listTask)
136                     {
137                         listTask[listTask.Where(c => c.Key.Id ==
                                Task.CurrentId).FirstOrDefault().Key
                               ] = agent.NoeudId;
138                     }
139                     bool res = false;
140                     var taskcomplete = 0;
141                     lock (listTask)
142                     {
143                         taskcomplete = listTask.Where(z => z.Key
                               .IsCompleted).ToList().Count();
144                     }
145
146                     var agentDeali = repoAgent.Obtenir().Where(o
                            => o.Delai != 0).Count() == 0 ? 0 :
                           repoAgent.Obtenir().Where(o => o.Delai !=
                            0).FirstOrDefault().Delai;
147
148                     var verfierRendezVous = nombreTask.Values.
                           Max() > delaiAg ? true : false;
149
150                     if ( !RendezVous.RV )
```

```
151                     {
152                         ///Synchroniser les rondes et vérifier
                                le Rendez-vous
153                         res = RendezVous.IsRendezVous(listTask,
                                verfierRendezVous,0);
154
155                     }
156                     ///Vérifier si le Rendez-vous est réalisé
                            pendant l'exploration
157                     if (res && RendezVous.RV)
158                     {
159
160                         agent.NoeudId = noeudAgentActuelle.Id;
161                         commentaire.Add(string.Format("\n Rendez
                                -vous des agents effectué pendant l'
                                exploration par l'agent {0} avec
                                succee", agentID));
162                         retour.GrapheExplorer = g;
163                         retour.ListDeplacement = deplacement;
164                         retour.Commentaire = commentaire;
165                         retour.RendezVous = true;
166                         return retour;
167                     }
168                     ///Vérifier si le noeud contient un jeton de
                            même couleur que l'agent
169                     if (noeudArrive.JetonsT.ToList().Where(p =>
                            p.Couleur == agent.Couleur).ToList().
                            Count() == 0)
170                     {
171                         ///Faire le DFS
172                         s = DFS(agent.Id, portIdArrive,
                                repoAgent, repoNoeud, repoArete, g,
                                commentaire, nombreTask, listTask);
173
174                     }
175
176
177                     var areteDerniere = repoArete.Obtenir().
                            ToList().Where(n => n.NoeudIdDepart ==
                            noeudIdDepart && n.NoeudIdArrivee ==
                            agent.NoeudId).FirstOrDefault();
```

```
178                     IArete areteDerniereAllez = new Arete(arete.
                            NoeudIdDepart, arete.PortIdDepart, arete.
                            NoeudIdArrivee, arete.PortIdArrivee,
                            agent.Couleur);
179                     IArete areteDerniereRetour = new Arete(arete
                            .NoeudIdArrivee, arete.PortIdArrivee,
                            arete.NoeudIdDepart, arete.PortIdDepart,
                            agent.Couleur);
180                     if (!ValiderArete(areteDerniereAllez, g))
181                     {
182                         g.aretes.Add(areteDerniereAllez);
183
184
185                     }
186
187                     if (!ValiderArete(areteDerniereRetour, g))
188                     {
189                         g.aretes.Add(areteDerniereRetour);
190
191                     }
192
193                     agent.NoeudId = noeudIdDepart;
194
195                     if(!RendezVous.RV)
196                     {
197                         ///Permet de faire le retour au point de
                                depart
198                         deplacement.Add(new Deplacement(agent.Id
                                , portIdArrive));
199                         commentaire.Add(string.Format("\n Retour
                                de l'Agent {0} du noeud {1} au noeud
                                {2}", agent.Id, noeudIdArrive,
                               noeudIdDepart));
200
201                     }
202
203                     nombreTask[Task.CurrentId.Value] =
                            nombreTask[Task.CurrentId.Value] + 1;
204                     lock (listTask)
205                     {
```

```
206                         listTask[listTask.Where(c => c.Key.Id ==
                                Task.CurrentId).FirstOrDefault().Key
                               ] = agent.NoeudId;
207                     }
208
209                     bool ress = false;
210
211                     var taskcomplete2 = 0;
212                     lock (listTask)
213                     {
214                         taskcomplete2 = listTask.Where(z => z.
                               Key.IsCompleted).ToList().Count();
215                     }
216
217                     var agentDeali2 = repoAgent.Obtenir().Where(
                        o => o.Delai != 0).Count() == 0 ? 0 :
                        repoAgent.Obtenir().Where(o => o.Delai !=
                         0).FirstOrDefault().Delai;
218                     bool verfierRendezVous2;
219                     lock (nombreTask)
220                     {
221                         ///définir si l'agent doit vérifier le
                               rendez-vous selon le différé entre
                               les agents
222                         verfierRendezVous2 = nombreTask.Values.
                               Max() > delaiAg ? true : false;
223                     }
224
225                     if (!RendezVous.RV )
226                     {
227                         ///Synchroniser les rondes et vérifier
                               le Rendez-vous
228                         ress = RendezVous.IsRendezVous(listTask,
                                verfierRendezVous2,0);
229
230                     }
231
232                     ///Vérifier si le Rendez-vous est réalisé
                        pendant l'exploration
233                     if (ress && RendezVous.RV)
234                     {
```

```
235
236                         agent.NoeudId = noeudAgentActuelle.Id;
237                         commentaire.Add(string.Format("\n Rendez
                                -vous des agents effectué pendant l'
                                exploration par l'agent {0} avec
                                succee", agentID));
238                         retour.GrapheExplorer = g;
239                         retour.ListDeplacement = deplacement;
240                         retour.Commentaire = commentaire;
241                         retour.RendezVous = true;
242                         return retour;
243                     }
244
245                 }
246                 else if(artes.Count() != 0 && mesJetons != 0)
247                 {
248                     var arete = artes.FirstOrDefault();
249                     IArete areteAllez = new Arete(arete.
                            NoeudIdDepart, arete.PortIdDepart, arete.
                            NoeudIdArrivee, arete.PortIdArrivee,
                            agent.Couleur);
250                     IArete areteRetour = new Arete(arete.
                            NoeudIdArrivee, arete.PortIdArrivee,
                            arete.NoeudIdDepart, arete.PortIdDepart,
                            agent.Couleur);
251
252                     if (!ValiderArete(areteAllez, g))
253                     {
254                         ///Ajouter l'arête traversé au graphe
                                que l'agent en train de construire
                                dans
255                         ///sa mémoire
256                         g.aretes.Add(areteAllez);
257
258                     }
259
260                     if (!ValiderArete(areteRetour, g))
261                     {
262                         ///Ajouter l'arête traversé au graphe
                                que l'agent en train de construire
                                dans
```

```
263                             ///sa mémoire
264                             g.aretes.Add(areteRetour);
265
266                         }
267
268                     }
269
270                 }
271             }
272
273
274             retour.GrapheExplorer = g;
275             retour.ListDeplacement = deplacement;
276             retour.Commentaire = commentaire;
277
278             return retour;
279         }
280
281         /// <summary>
282         /// Valider si l'arête existe
283         /// </summary>
284         /// <param name="ar"></param>
285         /// <param name="g"></param>
286         /// <returns></returns>
287         public bool ValiderArete(IArete ar, IGraphe g)
288         {
289             if (g.aretes.Where(n => n.NoeudIdDepart == ar.
                    NoeudIdDepart && n.PortIdDepart == ar.PortIdDepart).
                    Count() != 0)
290             {
291                 return true;
292
293             }
294             return false;
295
296         }
297
298         /// <summary>
299         /// Valider si le noeud existe
300         /// </summary>
301         /// <param name="noeud"></param>
```

```
302         /// <param name="g"></param>
303         /// <returns></returns>
304         public bool ValiderNoeud(INoeud noeud, IGraphe g)
305         {
306             if (g.noeuds.Where(n => n.Id == noeud.Id).Count() != 0)
307             {
308 
309                 return true;
310             }
311             return false;
312 
313         }
314 
315     }
316 }
```

Listing A.14 – Class AlgorithmeJetonIllimite

```
1 
2 using GraphExpert.Algorithmes.Interfaces;
3 using GraphExpert.Animations;
4 using GraphExpert.Data.Interfaces.Modeles;
5 using GraphExpert.Data.Interfaces.Repos;
6 using GraphExpert.Data.Modeles;
7 using System;
8 using System.Collections.Generic;
9 using System.Collections.ObjectModel;
10 using System.Linq;
11 using System.Threading;
12 using System.Threading.Tasks;
13 using System.Windows.Data;
14 
15 namespace GraphExpert.Algorithmes
16 {
17     public class AlgorithmeJetonIllimite : IAlgorithmeJetonIllimite
18     {
19 
20         /// <summary>
21         /// Variables de travail
22         /// </summary>
23         public string resultat = string.Empty;
```

```
24          public Dictionary<byte, IExploration> deplace = new
               Dictionary<byte, IExploration>();
25          SynchronizationContext uiContext = SynchronizationContext.
               Current;
26          public Dictionary<int, int> nombreTask = new Dictionary<int,
                int>();
27          public Dictionary<Task, byte> ListTask = new Dictionary<Task
               , byte>();
28          public List<Task> ListTaskRV = new List<Task>();
29
30
31
32
33          /// <summary>
34          /// Résoudre Le Rendez-vous avec des jetons illimités sur la
                position initiale et
35          /// au plus un jeton dans les autres noeuds.
36          /// </summary>
37          public void Resoudre(ObservableCollection<IDeplacement>
               deplacements, ObservableCollection<string> commentaires,
               IRepoAgents repoAgent, IRepoNoeuds repoNoeud, IRepoAretes
                repoArete)
38          {
39              ///Réinitialiser les variables
40              nombreTask.Clear();
41              ListTask.Clear();
42              ListTaskRV.Clear();
43              deplace.Clear();
44              deplacements.Clear();
45              commentaires.Clear();
46              RendezVous.RV = false;
47              RendezVous.Fin = false;
48              RendezVous.nombreFin = 0;
49              RendezVous.nombre = 0;
50              RendezVous.wh.Reset();
51
52
53              ///Pour chaque agent , on part un processus en mode
                   synchrone
54              foreach (var agent in repoAgent.Obtenir())
55              {
```

```
56                  Task RVIlimite = Task.Run(() => RendezVousagent(
                        agent.Id, 0, new List<Deplacement>(), repoAgent,
                        repoNoeud, repoArete));
57                  nombreTask.Add(RVIlimite.Id, 0);
58                  ListTask.Add(RVIlimite, 0);
59                  ListTaskRV.Add(RVIlimite);
60
61              }
62              ///Attendre que tous les processus(threads) terminent
                    leur exécution
63              Task.WaitAll(ListTaskRV.ToArray());
64
65              ///Envoyer les déplacements générés à l'interface
                    graphique
66              FaireDeplacemment(deplacements, commentaires, repoAgent,
                     repoNoeud, repoArete);
67
68
69          }
70
71
72          /// <summary>
73          /// L'algorithme qui permet de résoudre le rendez-vous avec
                des jetons illimités
74          /// </summary>
75          /// <param name="agentID"></param>
76          /// <param name="idPort"></param>
77          /// <param name="deplacement"></param>
78          /// <param name="repoAgent"></param>
79          /// <param name="repoNoeud"></param>
80          /// <param name="repoArete"></param>
81          public void RendezVousagent(byte agentID, byte idPort, List<
                Deplacement> deplacement, IRepoAgents repoAgent,
                IRepoNoeuds repoNoeud, IRepoAretes repoArete)
82          {
83              ///Variables de travail
84              AlgorithmeExplorationDfs aExDfs = new
                    AlgorithmeExplorationDfs();
85              AlgorithmeDijkstra aDijk = new AlgorithmeDijkstra();
86              var etiqueteAgent = string.Empty;
87              List<string> commentaire;
```

```
88              var nombreJeton = 0;
89              IExploration exp = new Exploration() ;
90              List<Deplacement> deplacementsDelai = new List<
                    Deplacement>();
91              List<string> commentaireDelai = new List<string>();
92              var delaiAgent = repoAgent.Obtenir(agentID).Delai;
93              var sortir = false;
94              INoeud noeudDestination = null;
95              var trouve = false;
96              var noeudDepart = repoNoeud.Obtenir(repoAgent.Obtenir(
                    agentID).NoeudId);
97              INoeud noeudStationnaire = noeudDepart;
98              etiqueteAgent = repoAgent.Obtenir(agentID).Etiquette;
99              nombreJeton = 0;
100
101             ///Rendre l'agent qui avec un délai inactif
102             if (delaiAgent != 0)
103             {
104                 while (!sortir)
105                 {
106                     ///Le nombre de ronde enregistré par l'agent
107                     int nombreTsk;
108                     lock (nombreTask)
109                     {
110                         nombreTsk = nombreTask.Values.ToList().Max()
                                ;
111                     }
112
113                     if ( delaiAgent > nombreTsk && !RendezVous.RV)
114                     {
115                         ///Ajouter un déplacement bidon
116                         deplacementsDelai.Add(new Deplacement(
                                agentID, 0));
117                         commentaireDelai.Add(string.Format("\n  L'
                                Agent {0} reste à sa position actuelle",
                                agentID));
118                         var res = RendezVous.IsRendezVous(ListTask,
                                false,0);
119
120                     }
121                     else
```

```
122                     {
123                         sortir = true;
124                         Task.Delay(100);
125                     }
126
127                 }
128
129             }
130
131             ///L'agent dépose son étiquette en nombre de jetons sur
                    sa position initiale
132             if (Int32.TryParse(etiqueteAgent, out nombreJeton))
133             {
134                 for (var i = 0; i < nombreJeton; i++)
135                 {
136
137                     lock (noeudStationnaire)
138                     {
139                         noeudStationnaire.JetonsT.Add(new Jeton {
                                Couleur = repoAgent.Obtenir(agentID).
                                Couleur });
140
141
142                     }
143
144                 }
145
146             }
147
148             ///Faire l'exploration jusqu'a ce qu'on trouve la
                    position initiale de l'autre agent
149             do
150             {
151                 trouve = false;
152                 ///initialisé les jetons utilisé pour l'exploration
153                 foreach (var nn in repoNoeud.Obtenir())
154                 {
155                     var jetons = nn.JetonsT.Where(x => x.Couleur ==
                            repoAgent.Obtenir(agentID).Couleur).ToList();
156                     if (jetons.Count != 0)
157                     {
```

```
158                         lock(repoNoeud)
159                         {
160                             repoNoeud.Obtenir(nn.Id).JetonsT.Where(x
                                    => x.Couleur == repoAgent.Obtenir(
                                   agentID).Couleur).ToList().RemoveAt
                                   (0);
161                         }
162
163                     }
164
165
166                 }
167
168                 IGraphe g = new Graphe();
169                 commentaire = new List<string>();
170                 commentaire.Clear();
171
172                 g.noeuds.Add(noeudStationnaire);
173
174                 commentaire.Add(string.Format("\n Debut de l'
                       exploration de l'Agent {0} ", agentID));
175                 ///Faire le DFS pour l'exploration
176                 exp = aExDfs.DFS(agentID, 0,  repoAgent, repoNoeud,
                       repoArete, g, commentaire, nombreTask, ListTask);
177
178                 ///Ajouter à l'objet exploration retourné par le DFS
                        les déplacement bidons faites au début
179                 ///à cause du délai
180                 var numList = deplacementsDelai.Count;
181                 if (numList != 0)
182                 {
183                     for (var j = 0; j < numList; j++)
184                     {
185                         exp.ListDeplacement.Insert(j,
                               deplacementsDelai.ElementAt(j));
186                         exp.Commentaire.Insert(j, commentaireDelai.
                               ElementAt(j));
187                     }
188                 }
189
```

```
190                 ///Vérifier si le Rendez-vous est réalisé pendant l'
                        exploration
191                 if (exp.RendezVous || RendezVous.RV)
192                 {
193                     RendezVous.wh.Set();
194 
195                     lock (deplace)
196                     {
197                         deplace.Add(agentID, exp);
198 
199                     }
200                     commentaire.Add(string.Format("\n Fin de
                            déplacement de l'Agent {0} ", agentID));
201                     RendezVous.Fin = true;
202                     RendezVous.wh.Set();
203                     return;
204 
205                 }
206                 ///vérifier si l'agent a trouvé la position initiale
                        de l'autre agent
207                 foreach (var no in exp.GrapheExplorer.noeuds)
208                 {
209                     var nombre = no.JetonsT.ToList().Where(u => u.
                            Couleur != repoAgent.Obtenir(agentID).Couleur
                            ).Count();
210                     if (nombre > 1)
211                     {
212                         noeudDestination = no;
213                         trouve = true;
214                     }
215 
216                 }
217 
218             } while (!trouve);
219 
220 
221             var result = false;
222 
223             commentaire.Add(string.Format("Agent {0} a termine DFS
                    avec le neoud destination {1}", agentID,
                    noeudDestination.Id));
```

```
224             ///Trouver le chemin le plus court lexicographiquement
                    entre les positions initiales des agents
225              var cheminCourt = aDijk.TrouverCourtChemin(repoAgent.
                    Obtenir(agentID).NoeudId, noeudDestination.Id, exp.
                    GrapheExplorer);
226              ///générer les déplacement en utilisant le chemin le
                    plus court
227              var allezRetour = AllerAuPlusPetit(cheminCourt, exp.
                    GrapheExplorer, repoAgent.Obtenir(agentID).NoeudId,
                    noeudDestination.Id, agentID,repoArete);
228              ///Vérifier les étiquette des agents pour que celui qui
                    a la plus grande va
229              ///se déplacer vers celui qui a la plus petite
230
231              var etiquetteAutre = noeudDestination.JetonsT.Count - 2;
232              if (Int32.Parse(repoAgent.Obtenir(agentID).Etiquette) >
                    etiquetteAutre)
233              {
234                  var rondeEXP = exp.ListDeplacement.Count;
235                  exp.RondeExploration = rondeEXP;
236
237                  var tracesRV = AretesCheminCourt(cheminCourt, exp.
                        GrapheExplorer, repoAgent.Obtenir(agentID).
                        NoeudId, noeudDestination.Id, agentID, repoArete)
                        ;
238
239                  exp.TracesRendezVous = tracesRV;
240
241                  commentaire.Add(string.Format("\n Debut de Rendez-
                        vous de l'Agent {0} ", agentID));
242                  var listChemin = cheminCourt.ToList();
243                  foreach (var b in allezRetour)
244                  {
245                      var positionElement = 0;
246                      lock (deplace)
247                      {
248                          exp.ListDeplacement.Add(b);
249
250
251                      }
```

```
252                     nombreTask[Task.CurrentId.Value] = nombreTask[
                            Task.CurrentId.Value] + 1;
253                     if(listChemin.Count() != 0)
254                     {
255                         lock (ListTask)
256                         {
257
258                             ListTask[ListTask.Where(c => c.Key.Id ==
                                    Task.CurrentId).FirstOrDefault().Key
                                ] = cheminCourt.ElementAt(0);
259                         }
260                         commentaire.Add(string.Format("\n
                                Deplacement Agent {0} au noeud {1}",
                                agentID, listChemin.ElementAt(0)));
261                         listChemin.RemoveAt(0);
262
263
264                         positionElement = positionElement + 1;
265                     }
266                     else
267                     {
268                         ListTask[ListTask.Where(c => c.Key.Id ==
                                Task.CurrentId).FirstOrDefault().Key] =
                                noeudDestination.Id;
269                         commentaire.Add(string.Format("\n
                                Deplacement Agent {0} au noeud {1}",
                                agentID, noeudDestination.Id));
270                     }
271
272                     if (!RendezVous.RV)
273                     {
274                         ///Synchroniser les rondes
275                         result = RendezVous.IsRendezVous(ListTask,
                                true, 0);
276                     }
277
278
279                     if(result)
280                     {
281                         break;
282                     }
```

```
283
284                 }
285                 ListTask[ListTask.Where(c => c.Key.Id == Task.
                        CurrentId).FirstOrDefault().Key] =
                        noeudDestination.Id;
286
287                 while (!RendezVous.Fin)
288                 {
289                     ///Synchroniser les rondes
290                     result = RendezVous.IsRendezVous(ListTask, false
                            , 2);
291                 }
292                 RendezVous.RV = true;
293                 RendezVous.wh.Set();
294
295
296             }
297             else
298             {
299                 while ( !RendezVous.Fin)
300                 {
301                     ///Synchroniser les rondes
302                     result = RendezVous.IsRendezVous(ListTask, false
                            , 1);
303                 }
304
305             }
306             commentaire.Add(string.Format("\n Rendez-vous réalisé
                    avec succée "));
307             RendezVous.Fin = true;
308             RendezVous.wh.Set();
309             lock (deplace)
310             {
311                 deplace.Add(agentID, exp);
312
313             }
314
315
316             commentaire.Add(string.Format("\n Fin de deplacement de
                    l'Agent {0} ", agentID));
317
```

```
318         }
319
320         /// <summary>
321         /// Trouver les arêtes qui forment le chemin le plus court
322         /// </summary>
323         /// <param name="cheminCourt"></param>
324         /// <param name="graphe"></param>
325         /// <param name="noeudIdDepart"></param>
326         /// <param name="noeudIdArrive"></param>
327         /// <param name="agentID"></param>
328         /// <param name="repoArete"></param>
329         /// <returns></returns>
330         public List<IArete> AretesCheminCourt(IEnumerable<byte>
                cheminCourt, IGraphe graphe, byte noeudIdDepart, byte
                noeudIdArrive, byte agentID, IRepoAretes repoArete)
331         {
332             List<IArete> retour = new List<IArete>();
333
334             IArete arete;
335
336
337             var noeudInitiale = noeudIdDepart;
338
339
340             for (var i = cheminCourt.Count() - 1; i >= 0; i--)
341             {
342
343                 var arete2 = graphe.aretes.Where(x => x.
                        NoeudIdDepart == noeudInitiale && x.
                        NoeudIdArrivee == cheminCourt.ElementAt(i));
344
345                 arete = repoArete.Obtenir(arete2.FirstOrDefault().
                        NoeudIdDepart, arete2.FirstOrDefault().
                        PortIdDepart);
346                 retour.Add(arete);
347
348
349             noeudInitiale = cheminCourt.ElementAt(i);
350
351             }
```

```
352             var arete3 = graphe.aretes.Where(x => x.NoeudIdDepart ==
                    noeudInitiale && x.NoeudIdArrivee == noeudIdArrive);
353
354             arete = repoArete.Obtenir(arete3.FirstOrDefault().
                   NoeudIdDepart, arete3.FirstOrDefault().PortIdDepart);
355             retour.Add(arete);
356
357             return retour;
358
359         }
360
361         /// <summary>
362         /// Générer les déplacement pour que l'agent avec la plus
                grande étiquette rejoint
363         /// l'autre agent avec la plus petite étiquette
364         /// </summary>
365         /// <param name="cheminCourt"></param>
366         /// <param name="graphe"></param>
367         /// <param name="noeudIdDepart"></param>
368         /// <param name="noeudIdArrive"></param>
369         /// <param name="agentID"></param>
370         /// <param name="repoArete"></param>
371         /// <returns></returns>
372         public List<Deplacement> AllerAuPlusPetit(IEnumerable<byte>
                cheminCourt, IGraphe graphe, byte noeudIdDepart, byte
                noeudIdArrive, byte agentID,IRepoAretes repoArete)
373         {
374             List<Deplacement> dep = new List<Deplacement>();
375
376             var noeudInitiale = noeudIdDepart;
377
378
379             for (var i = cheminCourt.Count() - 1; i >= 0; i--)
380             {
381
382                 var arete2 = graphe.aretes.Where(x => x.
                        NoeudIdDepart == noeudInitiale && x.
                        NoeudIdArrivee == cheminCourt.ElementAt(i));
383                 lock (dep)
384                 {
```

```
385                     dep.Add(new Deplacement(agentID, arete2.
                            FirstOrDefault().PortIdDepart));
386
387                 }
388
389                 noeudInitiale = cheminCourt.ElementAt(i);
390
391             }
392             var arete3 = graphe.aretes.Where(x => x.NoeudIdDepart ==
                    noeudInitiale && x.NoeudIdArrivee == noeudIdArrive);
393             lock (dep)
394             {
395                 dep.Add(new Deplacement(agentID, arete3.
                        FirstOrDefault().PortIdDepart));
396
397
398             }
399
400
401             return dep;
402         }
403
404         /// <summary>
405         /// changer la couleur du chemin le plus court
406         /// </summary>
407         /// <param name="listArete"></param>
408         /// <param name="repoArete"></param>
409         public async void TracerAreteCourtChemin(List<IArete>
                listArete, IRepoAretes repoArete)
410         {
411
412
413             foreach (var arete in listArete)
414             {
415                 repoArete.Obtenir(arete.NoeudIdDepart, arete.
                        PortIdDepart).Couleur = "Red";
416                 repoArete.Obtenir(arete.NoeudIdArrivee, arete.
                        PortIdArrivee).Couleur = "Red";
417             }
418
419
```

```
420
421             await Task.Delay(10);
422         }
423
424         ///Envoyer les déplacements générés par les agents à l'
                interface graphique
425         public async void FaireDeplacemment(ObservableCollection<
                IDeplacement> deplacements, ObservableCollection<string>
                commentaires, IRepoAgents repoAgent, IRepoNoeuds
                repoNoeud, IRepoAretes repoArete)
426         {
427             var sortir = false;
428
429             var agents = repoAgent.Obtenir();
430             ///Envoyer à l'interfaces les jetons qui représente l'
                    étiquette de l'agent
431             ///et qui vont être poser sur la position initiale
432             foreach(var ag in agents)
433             {
434                 var jetons = repoNoeud.Obtenir(ag.NoeudId).JetonsT.
                        Where(v => v.Couleur == ag.Couleur);
435                 var nombreJeton = 0;
436                 if (Int32.TryParse(ag.Etiquette, out nombreJeton))
437                 {
438                     for (var x = 0; x < nombreJeton; x++)
439                     {
440
441                         repoNoeud.Obtenir(ag.NoeudId).Jetons.Add(new
                                Jeton { Couleur = ag.Couleur });
442                         repoNoeud.Obtenir(ag.NoeudId).JetonsT.Remove
                            (jetons.FirstOrDefault());
443                     }
444
445
446                 }
447
448             }
449             var conteur = 0;
450
451             while (!sortir)
452             {
```

```
453                 conteur++;
454
455                 sortir = true;
456                 lock (deplace)
457                 {
458                     foreach (var dec in deplace)
459                     {
460                         if (dec.Value.ListDeplacement.Count != 0)
461                         {
462                             var valeurRondeEXP = dec.Value.
                                RondeExploration;
463                             if (valeurRondeEXP != 0 &&
                                valeurRondeEXP + 1 == conteur)
464                             {
465                                 var traces = dec.Value.
                                    TracesRendezVous;
466                                 TracerAreteCourtChemin(traces,
                                    repoArete);
467
468                             }
469                             sortir = false;
470                             var dep = dec.Value.ListDeplacement.
                                First();
471                             ///Ajouter les jetons à chaque
                                dépllacement de l'agent
472                             if (dep.PortId != 0)
473                             {
474                                 var nombre = repoNoeud.Obtenir(
                                    repoAgent.Obtenir(dec.Key).
                                    NoeudId).JetonsT;
475                                 var nombreT = repoNoeud.Obtenir(
                                    repoAgent.Obtenir(dec.Key).
                                    NoeudId).Jetons;
476                                 if (nombre.Count > 0)
477                                 {
478
479                                     repoNoeud.Obtenir(repoAgent.
                                        Obtenir(dec.Key).NoeudId).
                                        Jetons.Add(nombre.
                                        FirstOrDefault());
```

```
480                                     repoNoeud.Obtenir(repoAgent.
                                            Obtenir(dec.Key).NoeudId).
                                            JetonsT.Remove(nombre.
                                            FirstOrDefault());
481
482                                 }
483
484                                 deplacements.Add(new Deplacement(dec
                                        .Key, dep.PortId));
485
486
487                             }
488                             dec.Value.ListDeplacement.RemoveAt(0);
489
490
491                         }
492                         if (dec.Value.Commentaire.Count != 0)
493                         {
494                             sortir = false;
495                             commentaires.Add(dec.Value.Commentaire.
                                First());
496                             dec.Value.Commentaire.RemoveAt(0);
497                         }
498
499                     }
500
501
502                 }
503                 await Task.Delay(2000);
504
505             }
506
507         }
508
509
510     }
511 }
```

Listing A.15 – Class AlgorithmeDijkstra

```
1
```

```
2  using Dijkstra.NET.Contract;
3  using Dijkstra.NET.Model;
4  using Dijkstra.NET.ShortestPath;
5  using GraphExpert.Algorithmes.Interfaces;
6  using GraphExpert.Data.Interfaces.Modeles;
7  using GraphExpert.Data.Interfaces.Repos;
8  using System;
9  using System.Collections.Generic;
10 using System.Linq;
11 using System.Text;
12 using System.Threading.Tasks;
13
14 namespace GraphExpert.Algorithmes
15 {
16     /// <summary>
17     /// L'algorithme Dijkstra pour trouver le court chemin entre
           deux point
18     /// </summary>
19     public class AlgorithmeDijkstra : IAlgorithmeDijkstra
20     {
21
22         /// <summary>
23         /// Construire une structure de graphe utilisé par Dijkstra
24         /// </summary>
25         /// <param name="graphe"></param>
26         /// <returns></returns>
27         public Dictionary<byte, Dictionary<byte, int>> Construire(
               IGraphe graphe)
28         {
29             Dictionary<byte, Dictionary<byte, int>> vertices = new
                   Dictionary<byte, Dictionary<byte, int>>();
30             vertices.Clear();
31             foreach(var noeud in graphe.noeuds)
32             {
33                 Dictionary<byte, int> e = new Dictionary<byte, int
                       >();
34                 e.Clear();
35
36                 foreach (var voisin in graphe.aretes.ToList().Where(
                       n => n.NoeudIdDepart == noeud.Id))
37                 {
```

```
38                      if(!e.ContainsKey(voisin.NoeudIdArrivee))
39                      {
40                          e.Add(voisin.NoeudIdArrivee, 1);
41                      }
42
43                  }
44                  vertices.Add(noeud.Id, e);
45              }
46
47              return vertices;
48          }
49
50          /// <summary>
51          /// Trouver le court chemin entre deux noeud avec l'
                algorithme Dijkstra
52          /// </summary>
53          /// <param name="premierNoeud"></param>
54          /// <param name="deuxiemeNeoud"></param>
55          /// <param name="graphe"></param>
56          /// <returns></returns>
57          public IEnumerable<byte> TrouverCourtChemin(byte
                premierNoeud, byte deuxiemeNeoud, IGraphe graphe)
58          {
59              var previous = new Dictionary<byte, byte>();
60              var distances = new Dictionary<byte, int>();
61              var nodes = new List<byte>();
62              Dictionary<byte, Dictionary<byte, int>> vertices =
                    Construire(graphe);
63              Dictionary<byte, List<byte>> adja = new Dictionary<byte,
                    List<byte>>();
64
65              List<byte> path = null;
66
67              foreach (var vertex in vertices)
68              {
69                  if (vertex.Key == premierNoeud)
70                  {
71                      distances[vertex.Key] = 0;
72                  }
73                  else
74                  {
```

```
75                      distances[vertex.Key] = int.MaxValue;
76                  }
77
78                  nodes.Add(vertex.Key);
79                  adja[vertex.Key] = new List<byte>();
80              }
81
82              while (nodes.Count != 0)
83              {
84                  nodes.Sort((x, y) => distances[x] - distances[y]);
85
86                  var smallest = nodes[0];
87                  nodes.Remove(smallest);
88
89                  if (smallest == deuxiemeNeoud)
90                  {
91                      path = new List<byte>();
92                      while (previous.ContainsKey(smallest))
93                      {
94                          path.Add(smallest);
95                          smallest = previous[smallest];
96                      }
97
98                      break;
99                  }
100
101                 if (distances[smallest] == int.MaxValue)
102                 {
103                     break;
104                 }
105
106                 foreach (var neighbor in vertices[smallest])
107                 {
108                     var alt = distances[smallest] + neighbor.Value;
109                     adja[neighbor.Key].Add(smallest);
110                     if (alt < distances[neighbor.Key])
111                     {
112                         distances[neighbor.Key] = alt;
113                         previous[neighbor.Key] = smallest;
114
115
```

```
116                         }
117                     }
118                 }
119
120                 path.RemoveAll(x => x == deuxiemeNeoud);
121
122                 return path;
123             }
124
125     }
126 }
```

Listing A.16 – Class RendezVous

```
1
2 using System;
3 using System.Collections.Generic;
4 using System.Linq;
5 using System.Text;
6 using System.Threading;
7 using System.Threading.Tasks;
8
9 namespace GraphExpert.Algorithmes
10 {
11     public static class RendezVous
12     {
13         public static int nombre = 0;
14         public static int nombreFin = 0;
15
16         public static EventWaitHandle wh = new AutoResetEvent(false)
               ;
17         public static bool RV = false;
18         public static bool Fin = false;
19
20
21
22         public static bool IsRendezVous(Dictionary<Task, byte>
               listTask, bool verfierRV, int finExecution)
23         {
24
25             nombreFin = nombreFin + finExecution;
```

```
26              if (nombreFin % 2 != 0)
27              {
28                  Fin = true;
29              }
30
31              nombre++;
32              if(nombre % 2 == 0)
33              {
34
35                  lock (listTask)
36                  {
37                      if (listTask.Values.Where(x => x != 0).Distinct
                            ().Count() < 2 && verfierRV)
38                      {
39
40                          RV = true;
41                          wh.Set();
42                          return true;
43                      }
44
45                  }
46
47                  wh.Set();
48              }
49              else
50              {
51
52                  wh.WaitOne();
53              }
54
55              nombreFin = 0;
56
57              return false;
58          }
59      }
60  }
```

Listing A.17 – Class Resolveur

```
1
2  using GraphExpert.Algorithmes.Interfaces;
```

```
3  using GraphExpert.Animations;
4  using GraphExpert.Data.Interfaces.Repos;
5  using System.Collections.ObjectModel;
6
7  namespace GraphExpert.Algorithmes
8  {
9      /// <summary>
10     /// Effectue la résolution selon le type d'algorithme utilisé.
11     /// </summary>
12     public class Resolveur : IResolveur
13     {
14         /// <summary>
15         /// Conteneur.
16         /// </summary>
17         private IStrategieAlgorithme _strategie;
18
19         /// <summary>
20         /// Constructeur par défaut.
21         /// </summary>
22         /// <param name="strategie">Initialise la stratégie à
               adopter.</param>
23         public Resolveur(IStrategieAlgorithme strategie)
24         {
25             _strategie = strategie;
26         }
27
28         /// <summary>
29         /// Résoudre.
30         /// </summary>
31         /// <param name="type">Type algorithme.</param>
32         /// <param name="matrice">Matrice des poids selon les arrêts
               et leurs liaisons.</param>
33         public void Resoudre(TypeAlogorithmeEnum type,
               ObservableCollection<IDeplacement> deplacements,
               ObservableCollection<string> commentaire, IRepoAgents
               repoAgent, IRepoNoeuds repoNoeud, IRepoAretes repoArete)
34         {
35             _strategie.Obtenir(type).Resoudre(deplacements,
                   commentaire, repoAgent, repoNoeud, repoArete);
36         }
37     }
```

```
38 }
```

Listing A.18 – Class StrategieAlgorithme

```
1
2  using GraphExpert.Algorithmes.Interfaces;
3  using System;
4
5  namespace GraphExpert.Algorithmes
6  {
7      /// <summary>
8      /// Strategy Pattern : sélection de l'algorithme à utiliser.
9      /// </summary>
10     public class StrategieAlgorithme : IStrategieAlgorithme
11     {
12         /// <summary>
13         /// Algorithmes.
14         /// </summary>
15
16         private IAlgorithmeJetonIllimite _algoDfs;
17
18
19         /// <summary>
20         /// Constructeur par défaut.
21         /// Instanciation des algorithmes.
22         /// </summary>
23         /// <param name="algoDfs">Algorithme DFS.</param>
24         public StrategieAlgorithme(IAlgorithmeJetonIllimite algoDfs)
25         {
26
27             _algoDfs = algoDfs;
28
29         }
30
31         /// <summary>
32         /// Obtenir l'instance de son choix.
33         /// </summary>
34         /// <param name="type">Type désiré.</param>
35         /// <returns>Algo.</returns>
36         public IAlgorithme Obtenir(TypeAlogorithmeEnum type)
37         {
```

```
38              switch (type)
39              {
40
41                  case TypeAlogorithmeEnum.DFS:
42                      return _algoDfs;
43
44
45              }
46
47              throw new NotImplementedException();
48          }
49      }
50  }
```

Listing A.19 – Class AnimationDeplacement

```
1
2  using GraphExpert.Data.Interfaces.Repos;
3  using GraphExpert.Wpf.Controles;
4  using GraphExpert.Wpf.Interfaces;
5  using GraphExpert.Wpf.Models;
6  using System;
7  using System.Collections.Generic;
8  using System.Linq;
9  using System.Windows.Controls;
10 using System.Windows.Media;
11 using System.Windows.Media.Animation;
12 using GraphExpert.Animations;
13
14 namespace GraphExpert.Wpf.Services
15 {
16     public class AnimationDeplacement : IAnimationDeplacement
17     {
18         /// <summary>
19         /// Conteneurs.
20         /// </summary>
21         private IRepoNoeuds _repoNoeuds;
22         private IRepoPorts _repoPorts;
23         private IRepoAretes _repoAretes;
24         private IRepoAgents _repoAgents;
25
```

```
26          /// <summary>
27          /// Constructeur par défaut.
28          /// </summary>
29          /// <param name="repoNoeuds">Repository des noeuds.</param>
30          /// <param name="repoPorts">Repository des ports.</param>
31          /// <param name="repoAretes">Repository des arêtes.</param>
32          /// <param name="repoAgents">Repository des agents.</param>
33          public AnimationDeplacement(IRepoNoeuds repoNoeuds,
                IRepoPorts repoPorts, IRepoAretes repoAretes, IRepoAgents
                 repoAgents)
34          {
35              _repoNoeuds = repoNoeuds;
36              _repoPorts = repoPorts;
37              _repoAretes = repoAretes;
38              _repoAgents = repoAgents;
39          }
40
41          /// <summary>
42          /// Construire l'animation pour nous.
43          /// </summary>
44          /// <param name="controleListe">Contrôle de la liste.</param
                >
45          /// <param name="noeuds">Noeuds dans l'interface.</param>
46          /// <param name="deplacements">Déplacements à effectuer.</
                param>
47          public void Animer(ItemsControl controleListe, IEnumerable<
                IPositionCanvas> noeuds, params IDeplacement[]
                deplacements)
48          {
49              foreach (var deplacement in deplacements)
50              {
51                  AnimerAgent(controleListe, deplacement.AgentId,
                        deplacement.PortId, noeuds);
52              }
53          }
54
55          /// <summary>
56          /// Animer l'agent.
57          /// </summary>
58          /// <param name="controleListe"></param>
59          /// <param name="agentId"></param>
```

```
60          /// <param name="portId"></param>
61          /// <param name="noeuds"></param>
62          private void AnimerAgent(ItemsControl controleListe, byte
                agentId, byte portId, IEnumerable<IPositionCanvas> noeuds
                )
63          {
64              // Variables de travail.
65              var storyBoard1 = new Storyboard();
66              var storyBoard2 = new Storyboard();
67              var agentVM = noeuds.OfType<AgentVM>().SingleOrDefault(p
                    => p.Id == agentId);
68              var controleAgent = VisualTreeHelper.GetChild(
                   controleListe.ItemContainerGenerator.
                   ContainerFromItem(agentVM), 0) as Agent;
69      StopVM noeudDest = null;
70
71              if (portId != 0)
72              {
73                  noeudDest = ObtenirNoeud(agentId, portId, noeuds.
                       OfType<StopVM>());
74              }
75              else
76              {
77                  noeudDest = noeuds.OfType<StopVM>().SingleOrDefault(
                       p => p.Id == _repoAgents.Obtenir(agentId).NoeudId
                       ); ;
78              }
79
80              var duree = TimeSpan.FromSeconds(1d);
81              var lineaireXDebut = new LinearDoubleKeyFrame(
                   controleAgent.Left, KeyTime.FromTimeSpan(TimeSpan.
                   Zero));
82              var lineaireXFin = new LinearDoubleKeyFrame(noeudDest.X,
                    KeyTime.FromTimeSpan(TimeSpan.Zero + duree));
83              var lineaireYDebut = new LinearDoubleKeyFrame(
                   controleAgent.Top, KeyTime.FromTimeSpan(TimeSpan.Zero
                   ));
84              var lineaireYFin = new LinearDoubleKeyFrame(noeudDest.Y,
                    KeyTime.FromTimeSpan(TimeSpan.Zero + duree));
85              var collection1 = new DoubleAnimationUsingKeyFrames();
86              var collection2 = new DoubleAnimationUsingKeyFrames();
```

```
87
88              collection1.KeyFrames.Add(lineaireXDebut);
89              collection2.KeyFrames.Add(lineaireYDebut);
90              collection1.KeyFrames.Add(lineaireXFin);
91              collection2.KeyFrames.Add(lineaireYFin);
92
93              // Construire l'animation avec le point de départ et le
                    point d'arrivée.
94              storyBoard1.Children
95                  .Add(collection1);
96
97              storyBoard2.Children
98                  .Add(collection2);
99
100             // Assigner la cible de l'animation.
101             Storyboard.SetTarget(storyBoard1, controleAgent);
102             Storyboard.SetTargetProperty(storyBoard1, new System.
                    Windows.PropertyPath(Agent.LeftProperty.Name));
103             Storyboard.SetTarget(storyBoard2, controleAgent);
104             Storyboard.SetTargetProperty(storyBoard2, new System.
                    Windows.PropertyPath(Agent.TopProperty.Name));
105
106             // Démarrer l'animation.
107             storyBoard1.Begin();
108             storyBoard2.Begin();
109
110             // Changer le noeud de l'agent.
111             agentVM.NoeudId = noeudDest.Id;
112             _repoAgents.Obtenir(agentId).NoeudId = noeudDest.Id;
113         }
114
115         /// <summary>
116         /// Obtenir le noeud (visuel) pour l'animation.
117         /// </summary>
118         /// <param name="agentId">Nř de l'agent de départ.</param>
119         /// <param name="portId">Nř du pord de départ.</param>
120         /// <param name="noeuds">Noeuds (visuel).</param>
121         /// <returns>Noeud d'arrivée.</returns>
122         private StopVM ObtenirNoeud(byte agentId, byte portId,
                IEnumerable<StopVM> noeuds)
123         {
```

```
124             var noeudIdDepart = _repoAgents.Obtenir(agentId).NoeudId
                    ;
125             var arete = _repoAretes.Obtenir(noeudIdDepart, portId);
126             return noeuds.SingleOrDefault(p => p.Id == arete.
                    NoeudIdArrivee);
127         }
128     }
129 }
```

Listing A.20 – Class MainWindowViewModel

```
1
2 using GraphExpert.Algorithmes.Interfaces;
3 using GraphExpert.Animations;
4 using GraphExpert.Data.Interfaces.Modeles;
5 using GraphExpert.Data.Interfaces.Repos;
6 using GraphExpert.Wpf.Interfaces;
7 using GraphExpert.Wpf.Models;
8 using GraphExpert.Wpf.Services;
9 using Prism.Commands;
10 using System;
11 using System.Collections.Generic;
12 using System.Collections.ObjectModel;
13 using System.Collections.Specialized;
14 using System.Linq;
15 using System.Windows.Controls;
16 using System.Windows.Input;
17 using System.Windows.Media;
18
19 namespace GraphExpert.Wpf.ViewModels
20 {
21     /// <summary>
22     /// Selon la convention MVVM.
23     /// Vue-modèle de l'écran d'accueil.
24     /// </summary>
25     public class MainWindowViewModel : Prism.Mvvm.BindableBase
26     {
27         #region Fields
28
29         /// <summary>
30         /// Algorithme sélectionné.
```

```
31          /// </summary>
32          private TypeAlogorithmeEnum _algorithme =
                TypeAlogorithmeEnum.FloydWarshall;
33
34          /// <summary>
35          /// Service de déplacement.
36          /// </summary>
37          private IAnimationDeplacement _animation;
38
39          /// <summary>
40          /// Arrêt nř1 cliqué.
41          /// </summary>
42          private StopVM _arret1;
43
44          /// <summary>
45          /// Déplacements.
46          /// </summary>
47          private ObservableCollection<IDeplacement> _deplacements =
                new ObservableCollection<IDeplacement>();
48
49      private ObservableCollection<string> _commentaire = new
            ObservableCollection<string>();
50
51          /// <summary>
52          /// Contrôle des items.
53          /// </summary>
54          private ItemsControl _itemsControl;
55
56          /// <summary>
57          /// Repos.
58          /// </summary>
59          private IRepoAgents _repoAgents;
60          private IRepoNoeuds _repoArrets;
61          private IRepoAretes _repoLiaisons;
62          private IRepoPorts _repoPorts;
63
64          /// <summary>
65          /// Résolveur.
66          /// </summary>
67          private IResolveur _resolveur;
68
```

```
69          /// <summary>
70          /// Conteneur pour la propriété.
71          /// </summary>
72          private string _solution = string.Empty;
73
74          #endregion Fields
75
76          #region Constructors
77
78          /// <summary>
79          /// Constructeur par défaut.
80          /// </summary>
81          /// <param name="repoArrets">Repository des arrêts.</param>
82          /// <param name="repoLiaisons">Repository des liaisons.</
                param>
83          /// <param name="repoAgents">Repository des agents.</param>
84          /// <param name="repoPorts">Repository des ports.</param>
85          /// <param name="resolveur">Résoudre par l'algorithme voulu
                .</param>
86          /// <param name="animation">Animation.</param>
87          public MainWindowViewModel(IRepoNoeuds repoArrets,
                IRepoAretes repoLiaisons, IRepoAgents repoAgents,
                IRepoPorts repoPorts, IResolveur resolveur,
                IAnimationDeplacement animation)
88          {
89              _repoArrets = repoArrets;
90              _repoLiaisons = repoLiaisons;
91              _repoAgents = repoAgents;
92              _repoPorts = repoPorts;
93              _resolveur = resolveur;
94              _animation = animation;
95
96              // Initialisation des commandes.
97              CommandeResoudre = new DelegateCommand(Resoudre,
                    PeutResoudre);
98              CommandeNettoyer = new DelegateCommand(Nettoyer);
99              CommandeReinitialiser = new DelegateCommand(
                    Reinitialiser, PeutReinitialiser);
100
101             _deplacements.CollectionChanged +=
                    ListeDeplacements_CollectionChanged;
```

```
102         _commentaire.CollectionChanged +=
                ListeCommentaire_CollectionChanged;
103           }
104
105           #endregion Constructors
106
107           #region Destructors
108
109           /// <summary>
110           /// Disposer des évènements.
111           /// </summary>
112           ~MainWindowViewModel()
113           {
114               _deplacements.CollectionChanged -=
                      ListeDeplacements_CollectionChanged;
115               _deplacements = null;
116         _commentaire.CollectionChanged -=
                ListeCommentaire_CollectionChanged;
117               _commentaire = null;
118           }
119
120           #endregion Destructors
121
122           #region Properties
123
124           /// <summary>
125           /// Obtient la liste des agents.
126           /// </summary>
127           public IEnumerable<AgentVM> Agents => Formes.OfType<AgentVM
                  >();
128
129           /// <summary>
130           /// Menu contextuel 'Nettoyer'.
131           /// </summary>
132           public ICommand CommandeNettoyer { get; private set; }
133
134           /// <summary>
135           /// Commande pour réinitialiser les agents et les jetons.
136           /// </summary>
137           public DelegateCommand CommandeReinitialiser { get; private
                  set; }
```

```
138
139         /// <summary>
140         /// Bouton 'résoudre'.
141         /// </summary>
142         public DelegateCommand CommandeResoudre { get; private set;
                }
143
144         /// <summary>
145         /// Sélection du choix.
146         /// </summary>
147         public bool EstAlgoBFS
148         {
149             get { return _algorithme == TypeAlogorithmeEnum.BFS; }
150             set
151             {
152                 SetProperty(ref _algorithme, TypeAlogorithmeEnum.BFS
                        , @"EstAlgoBFS");
153                 RaisePropertyChanged(@"EstAlgoDFS");
154                 RaisePropertyChanged(@"EstAlgoFW");
155             }
156         }
157
158         /// <summary>
159         /// Sélection du choix.
160         /// </summary>
161         public bool EstAlgoDFS
162         {
163             get { return _algorithme == TypeAlogorithmeEnum.DFS; }
164             set
165             {
166                 SetProperty(ref _algorithme, TypeAlogorithmeEnum.DFS
                        , @"EstAlgoDFS");
167                 RaisePropertyChanged(@"EstAlgoFW");
168                 RaisePropertyChanged(@"EstAlgoBFS");
169             }
170         }
171
172         /// <summary>
173         /// Sélection du choix.
174         /// </summary>
175         public bool EstAlgoFW
```

```
176         {
177             get { return _algorithme == TypeAlogorithmeEnum.
                    FloydWarshall; }
178             set
179             {
180                 SetProperty(ref _algorithme, TypeAlogorithmeEnum.
                        FloydWarshall, @"EstAlgoFW");
181                 RaisePropertyChanged(@"EstAlgoDFS");
182                 RaisePropertyChanged(@"EstAlgoBFS");
183             }
184         }
185
186         /// <summary>
187         /// Formes à afficher.
188         /// </summary>
189         public ObservableCollection<IPositionCanvas> Formes { get;
                private set; } = new ObservableCollection<IPositionCanvas
                >();
190
191         /// <summary>
192         /// Obtient la liste de noeuds.
193         /// </summary>
194         public IEnumerable<StopVM> Noeuds => Formes.OfType<StopVM>()
                ;
195
196         /// <summary>
197         /// Solution à afficher.
198         /// </summary>
199         public string Solution
200         {
201             get => _solution;
202             set
203             {
204                 _solution = value;
205                 RaisePropertyChanged(@"Solution");
206             }
207         }
208
209         #endregion Properties
210
211         #region Methods
```

```
212
213         /// <summary>
214         /// Ajouter l'arrêt à la persistance et ensuite l'afficher.
215         /// </summary>
216         /// <param name="x">Coordonnée X à l'écran.</param>
217         /// <param name="y">Coordonnée Y à l'écran.</param>
218         /// <param name="etiquette">Étiquette à afficher.</param>
219         public void AjouterArret(double x, double y, string
                etiquette)
220         {
221             // Ajout dans la persistance.
222             var arret = _repoArrets.Ajouter(etiquette);
223
224             // Afficher.
225             Formes.Add(new StopVM(x, y, arret));
226             RaisePropertyChanged(@"Noeuds");
227
228             // Aviser l'interface pour résoudre.
229             CommandeResoudre.RaiseCanExecuteChanged();
230         }
231
232         /// <summary>
233         /// Ajouter une liaison entre ces deux arrêts à l'écran.
234         /// </summary>
235         /// <param name="arret1">Arrêt de départ.</param>
236         /// <param name="arret2">Arrêt d'arrivée.</param>
237         /// <returns>Vrai si la liaison est effective.</returns>
238         public void AjouterLiaison(StopVM arret)
239         {
240             // Il s'agit du premier clic.
241             if (null == _arret1)
242             {
243                 _arret1 = arret;
244             }
245             // Il s'agit du deuxième ; on crée la liaison.
246             else
247             {
248
249
250                 // Ajouter les deux liaisons dans les deux sens.
251                 var portDepa = _repoPorts.Ajouter(_arret1.Id);
```

```
252                 var portArri = _repoPorts.Ajouter(arret.Id);
253             var arete = _repoLiaisons.Ajouter(portDepa.NoeudId, portDepa
                    .Id, portArri.NoeudId, portArri.Id);
254                 _repoLiaisons.Ajouter(portDepa.NoeudId, portDepa.Id,
                        portArri.NoeudId, portArri.Id);
255                 _repoLiaisons.Ajouter(portArri.NoeudId, portArri.Id,
                        portDepa.NoeudId, portDepa.Id);
256                 _repoArrets.Obtenir(_arret1.Id).Degree = _repoArrets
                        .Obtenir(_arret1.Id).Degree + 1;
257                 _repoArrets.Obtenir(arret.Id).Degree = _repoArrets.
                        Obtenir(arret.Id).Degree + 1;
258                 // Ajout aux éléments à afficher.
259                 var ligneUI = new LineVM(_arret1, arret, arete);
260                 Formes.Add(ligneUI);
261
262                 AjouterPortVM(_arret1, ligneUI, portDepa);
263                 AjouterPortVM(arret, ligneUI, portArri);
264
265                 // Retirer l'arrêt.
266                 _arret1 = null;
267
268                 // Aviser l'interface pour résoudre.
269                 CommandeResoudre.RaiseCanExecuteChanged();
270                 CommandeReinitialiser.RaiseCanExecuteChanged();
271             }
272         }
273
274         /// <summary>
275         /// Assigner la valeur du contrôle (pour les déplacements).
276         /// </summary>
277         /// <param name="itemsControl">Contrôle.</param>
278         public void AssignerItemsControl(ItemsControl itemsControl)
279         {
280             _itemsControl = itemsControl;
281         }
282
283         /// <summary>
284         /// Nettoyer le tout.
285         /// </summary>
286         public void Nettoyer()
287         {
```

```
288             // Vider les repos.
289             _repoArrets.Vider();
290             _repoLiaisons.Vider();
291             _repoPorts.Vider();
292             _repoAgents.Vider();
293 
294             // Vider l'instance en mémoire d'un arrêt cliqué.
295             _arret1 = null;
296 
297             // Vider l'IU.
298             Formes.Clear();
299             RaisePropertyChanged(@"Noeuds");
300             RaisePropertyChanged(@"Agents");
301 
302             // Vider la partie "Solution".
303             Solution = string.Empty;
304         RaisePropertyChanged(@"Solution");
305 
306             // Aviser l'interface pour rafraichir les commandes.
307             CommandeResoudre.RaiseCanExecuteChanged();
308         }
309 
310         /// <summary>
311         /// Résoudre par l'algorithme choisi.
312         /// </summary>
313         public bool PeutResoudre()
314         {
315             // Autorisation accordée quand l'interface possède :
316             // 2 agents
317             // 3 noeuds
318             // 3 arêtes et
319             // 6 ports.
320             return ((2 <= Formes.OfType<AgentVM>()?.Count()) &&
321                 (2 <= Formes.OfType<StopVM>()?.Count()) &&
322                 (1 <= Formes.OfType<LineVM>()?.Count()) &&
323                 (2 <= Formes.OfType<PortVM>()?.Count()));
324         }
325 
326         /// <summary>
327         /// Résoudre par l'algorithme choisi.
328         /// </summary>
```

```
329         public bool PeutReinitialiser()
330         {
331             return ((true == Formes.OfType<AgentVM>()?.Any()) &&
332                 (true == Formes.OfType<StopVM>()?.Any()));
333         }
334 
335         /// <summary>
336         /// Résoudre par l'algorithme choisi.
337         /// </summary>
338         public void Reinitialiser()
339         {
340             for (int i = (Formes.Count - 1); i >= 0; i--)
341             {
342                 if (Formes[i].GetType() == typeof(AgentVM))
343                 {
344                     Formes.RemoveAt(i);
345                 }
346                 else if (Formes[i].GetType() == typeof(StopVM))
347                 {
348                     ((StopVM)Formes.ElementAt(i)).Jetons?.Clear();
349                 }
350             }
351 
352             // Vider les agents.
353             _repoAgents.Vider();
354 
355             // Vider l'instance en mémoire d'un arrêt cliqué.
356             _arret1 = null;
357 
358             // Vider la partie "Solution".
359             Solution = string.Empty;
360             RaisePropertyChanged(@"Solution");
361 
362             // Réinitialiser les listes.
363             RaisePropertyChanged(@"Noeuds");
364             RaisePropertyChanged(@"Agents");
365         }
366 
367         /// <summary>
368         /// Résoudre par l'algorithme choisi.
369         /// </summary>
```

```
370         public void Resoudre()
371         {
372             // Remettre à zéro la solution.
373             Solution = string.Empty;
374 
375             _resolveur.Resoudre(_algorithme, _deplacements,
                    _commentaire, _repoAgents, _repoArrets, _repoLiaisons
                    );
376         }
377 
378         /// <summary>
379         /// Ajouter un agent.
380         /// </summary>
381         /// <param name="x">Coordonnée X à l'écran.</param>
382         /// <param name="y">Coordonnée Y à l'écran.</param>
383         /// <param name="noeudId">Nř du noeud.</param>
384         /// <param name="etiquette">Etiquette.</param>
385         internal void AjouterAgent(double x, double y, byte noeudId,
                string etiquette)
386         {
387             // Ajout dans la persistance.
388             var agent = _repoAgents.Ajouter(noeudId, etiquette);
389 
390             if (null != agent)
391             {
392                 // Afficher.
393                 Formes.Add(new AgentVM(x, y, agent));
394                 RaisePropertyChanged(@"Agents");
395 
396                 // Aviser l'interface pour rafraichir les commandes.
397                 CommandeResoudre.RaiseCanExecuteChanged();
398                 CommandeReinitialiser.RaiseCanExecuteChanged();
399             }
400         }
401 
402         /// <summary>
403         /// Ajoute le port au noeud demandé.
404         /// </summary>
405         /// <param name="noeud">Noeud.</param>
406         /// <param name="arete">Arête.</param>
407         /// <param name="port">ÍPort.</param>
```

```
408         private void AjouterPortVM(StopVM noeud, LineVM arete, IPort
                port)
409         {
410             // Positionner.
411             double x = 0d, y = 0d;
412 
413             PositionnerLabelPort(noeud, arete, ref x, ref y);
414 
415             // Ajout du port.
416             Formes.Add(new PortVM(port, x, y));
417 
418             // Aviser l'interface pour résoudre.
419             CommandeResoudre.RaiseCanExecuteChanged();
420         }
421 
422         /// <summary>
423         /// Positionne l'étiquette (label) du port.
424         /// </summary>
425         /// <param name="noeud">Noeud de départ.</param>
426         /// <param name="arete">Arête.</param>
427         /// <param name="x">Position X.</param>
428         /// <param name="y">Position Y.</param>
429         private void PositionnerLabelPort(StopVM noeud, LineVM arete
              , ref double x, ref double y)
430         {
431             double D = 30; // diametre du cercle
432             double
433                 nx = noeud.X + D / 2,
434                 ny = noeud.Y + D / 2;
435             double Ax = nx, Ay = ny; // Position du centre du cercle
                    courant
436 
437             double d1 = (arete.X - Ax) * (arete.X - Ax) + (arete.Y -
                    Ay) * (arete.Y - Ay);
438             double d2 = (arete.X2 - Ax) * (arete.X2 - Ax) + (arete.
                Y2 - Ay) * (arete.Y2 - Ay);
439 
440             double Bx, By; // position du de l'autre bout de la
                ligne
441 
442             if (d1 > d2)
```

```
443             {
444                 // Arrete P1 est le plus différent de A
445                 Bx = arete.X;
446                 By = arete.Y;
447             }
448             else
449             {
450                 // Arrete P2 est le plus différent de A
451                 Bx = arete.X2;
452                 By = arete.Y2;
453             }
454 
455             double
456                 U = 10, // distance parallèle à a ligne du # (à
                        partir de la circonférence)
457                 V = 10; // distance perpendiculaire à a ligne du #
458 
459             double
460                 Lx = Bx - Ax,
461                 Ly = By - Ay; // Description vecteur de la ligne a->
                        b
462 
463             double LL = Math.Sqrt(Lx * Lx + Ly * Ly);
464 
465             double
466                 L1x = Lx / LL,
467                 L1y = Ly / LL; //Direction de L, normalisé
468 
469             double
470                 M1x = -L1y, // L1 avec rotation de -90 deg.
471                 M1y = L1x;
472 
473             double UR = U + D / 2;
474 
475             double
476                 LUx = L1x * UR,
477                 LUy = L1y * UR;
478 
479             double
480                 MVx = M1x * V,
481                 MVy = M1y * V;
```

```
482
483             double charW = 6, charH = 8; // taille aprox. d'un
                    charactere
484
485             // Position du haut gauche du # de port
486             x = Ax + LUx + MVx - charW / 2;
487             y = Ay + LUy + MVy - charH / 2;
488         }
489
490         /// <summary>
491         /// Sur changement de la collection, animer!
492         /// </summary>
493         /// <param name="sender">Objet envoyant l'évènement.</param>
494         /// <param name="e">Arguments de notification.</param>
495         private void ListeDeplacements_CollectionChanged(object
                sender, NotifyCollectionChangedEventArgs e)
496         {
497             if (e.Action == NotifyCollectionChangedAction.Add)
498             {
499                 var deplacements = e.NewItems.OfType<IDeplacement>()
                        ;
500
501                 _animation.Animer(_itemsControl, Formes,
                        deplacements.ToArray());
502
503                 //if (Solution.Length != 0) Solution += ", ";
504                 //Solution += string.Join(", ", deplacements.Select(
                        p => p.AgentId));
505             }
506         }
507
508     private void ListeCommentaire_CollectionChanged(object sender,
            NotifyCollectionChangedEventArgs e)
509         {
510             if (e.Action == NotifyCollectionChangedAction.Add)
511             {
512
513                 foreach(var chaine in e.NewItems.OfType<string>().
                        ToArray())
514                 {
515
```

```
516
517
518                         Solution = Solution + chaine;
519                         OnPropertyChanged("Solution");
520
521
522
523
524
525                     }
526
527                 }
528             }
529
530             #endregion Methods
531
532         }
533 }
```

Bibliographie

[1] S. Albers, and M. R. Henzinger, Exploring unknown environments, SIAM Journal on Computing 29 (2000), 1164–1188.

[2] S. Bouchard, Y. Dieudonné, and B. Ducourthial, Byzantine gathering in networks, Distributed Computing 29 (2016), 435–457.

[3] J. Chalopin, S. Das, and A. Kosowski, Constructing a map of an anonymous graph : applications of universal sequences, Proc. 14th International Conference On Principles Of Distributed Systems (OPODIS 2010), 119–134.

[4] J. Chalopin, S. Das, and N. Santoro, Rendezvous of mobile agents in unknown graphs with faulty links, Proc. 21th International Conference on Distributed Computing (DISC 2007), 108–122.

[5] J. Chalopin, Y. Dieudonné, A. Labourel, A. Pelc, Rendezvous in networks in spite of delay faults, Distributed Computing 29 (2016), 187–205.

[6] M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro, Distributed computing by mobile robots : gathering, SIAM Journal on Computing 41 (2012), 829–879.

[7] J. Czyzowicz, A. Kosowski, and A. Pelc, How to meet when you forget : log-space rendezvous in arbitrary graphs, Distributed Computing 25 (2012), 165–178.

[8] J. Czyzowicz, A. Labourel, and A. Pelc, How to meet asynchronously (almost) everywhere, ACM Transactions on Algorithms 8 (2012), 1505–1509.

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

[10] A. Dessmark, and A. Pelc, Optimal graph exploration without good maps, Theoretical Computer Science 326 (2004), 343–362.

[11] Y. Dieudonné, and A. Pelc, Anonymous meeting in networks, Algorithmica 74 (2016), 908–946.

[12] Y. Dieudonné, A. Pelc, and D. Peleg, Gathering despite mischief, ACM Transactions on Algorithms 11 (2014), article 1.

[13] Y. Dieudonné, A. Pelc, and V. Villain, How to meet asynchronously at polynomial cost, SIAM Journal on Computing 44 (2015), 844–867.

[14] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro, Multiple agents rendezVous in a ring in spite of a black hole, Proc. 7th International Conference On Principles Of Distributed Systems (OPODIS 2003), 34–46.

[15] C. A. Duncan, S. G. Kobourov, and V. S. A. Kumar, Optimal constrained graph exploration, Proc. 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2001), 807–814.

[16] S. Elouasbi, and A. Pelc, Time of anonymous rendezvous in trees : determinism vs. randomization, Proc. 19th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2012), 291–302.

[17] P. Fraigniaud, and A. Pelc, Delays induce an exponential memory gap for rendezvous in trees, ACM Transactions on Algorithms 9 (2013), article 17.

[18] S. Guilbault, and A. Pelc, Asynchronous rendezvous of anonymous agents in arbitrary graphs, Proc. 15th International Conference on Principles of Distributed Systems (OPODIS 2011), 162–173.

[19] E. Kranakis, D. Krizanc, F. Macquarrie, and S. Shende, Randomized rendezvous algorithms for agents on a ring with different speeds, Proc. 15th International Conference on Distributed Computing and Networking (ICDCN 2015), article 9.

[20] E. Kranakis, D. Krizanc, and P. Morin, Randomized rendez-vous with limited memory, ACM Transactions on Algorithms 7 (2011), article 34.

[21] G. De Marco, L. Gargano, E. Kranakis, D. Krizanc, A. Pelc, and U. Vaccaro, Asynchronous deterministic rendezvous in graphs, Theoretical Computer Science 355 (2006), 315–326.

[22] A. Miller, and A. Pelc, Fast rendezvous with advice, Theoretical Computer Science 608 (2015), 190–198.

[23] A. Miller, and A. Pelc, Tradeoffs between cost and information for rendezvous and treasure hunt, Journal of Parallel and Distributed Computing 83 (2015), 159–167.

[24] F. Ooshita, S. Kawai, H. Kakugawa, and T. Masuzawa, Randomized gathering of mobile agents in anonymous unidirectional ring networks, IEEE Transactions on Parallel and Distributed Systems 25 (2014), 1289-1296.

[25] A. Pelc, Deterministic rendezvous in networks : a comprehensive survey, Networks 59 (2012), 331–347.

[26] A. Ta-Shma, and U. Zwick, Deterministic rendezvous, treasure hunts and strongly universal exploration sequences, Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA 2007), 599–608.