L'exploration optimale d'un segment par un ensemble de robots mobiles

Mémoire
présenté comme exigence partielle
de la maitrise en informatique

Présenté par
Artur Stec

sous la direction de
Prof. Jurek Czyzowicz

Département d'informatique et d'ingénierie
Université du Québec en Outaouais

2015

UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

Département d'informatique et d'ingénierie

Ce mémoire intitulé :

L'EXPLORATION OPTIMALE D'UN SEGMENT PAR UN ENSEMBLE DE ROBOTS MOBILES

présenté par

Artur Stec

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

à été évalué par un jury composé des personnes suivantes :

Dr Jurek Czyżowicz ..... Directeur de recherche

Dr Mohand Saïd Allili ..... Président du jury

Dr Andrzej Pelc ..... Membre du jury

Mémoire accepté le 24 août 2015

Remerciements

J'aimerais remercier mon directeur de recherche Dr Jurek Czyżowicz pour ses conseils et son engagement, ainsi que sa patience sans limites. Merci à ma femme Beata de m'avoir soutenu et encouragé tout au long de ce projet et merci à mon fils Kevin pour son aide à la relecture et la correction du français.

Table des matières

Remerciements **i**

Liste des figures **iv**

Résumé **v**

1 Introduction **1**

1.1 Problème 2

1.2 Objectif 3

1.3 Méthodologie 4

2 État des connaissances **5**

2.1 Planification de trajectoire 6

2.2 Évaluation d'efficacité des algorithmes impliquant des agents mobiles 9

2.3 Environnement géométrique ou graphe 11

2.4 Environnement connu ou non connu 12

2.5 Un ou plusieurs robots 13

2.6 Catégorie de problèmes de recherche et d'exploration 14

2.7 Environnements 1-dimensionnels 15

3 Les bases de l'algorithme d'exploration **19**

3.1 Parcours par un robot 20

3.2 Parcours collectif 23

3.3 Parcours sans interférences 25

3.4 Parcours optimaux 30

4 Algorithme approximatif **33**

4.1 Fonctions utilisées par l'algorithme 33

4.2 Présentation de l'algorithme 38

5 Algorithme de calcul exact **40**

5.1 L'idée de l'algorithme 40

5.2 Procédures et fonctions utilisées par l'algorithme 47

5.3 Présentation de l'algorithme 53

6 Programme pour le système d'exploitation iOS **56**

6.1 Description de fonctionnement 57

6.2 Exemple d'exécution de l'algorithme TA 62

6.3 Exemple d'exécution de l'algorithme TE 67

7 Conclusion **73**

Annexes **75**

A Implémentation iOS **75**

Bibliographie **97**

Liste des figures

3.1 Exemple de parcours d'un robot 21
3.2 Les parcours optimaux 22
3.3 Exemple d'un parcours (collectif) 23
3.4 Parcours d'exploration 24
3.5 Parcours sans croisements 26
3.6 Parcours sans interférences 27
3.7 Parcours correct, sans croisements, sans interférences 30
3.8 Parcours optimal 31

5.1 Robot en état 1 44
5.2 Robot en état 2 45
5.3 Robot change l'état de 0 à 1 51
5.4 Robot change état de 1 à 2 51
5.5 Robot change l'état de 2 à 3 52

Résumé

L'ensemble de n robots est placé sur un segment dans des emplacements prédéterminés. Tous les robots peuvent se déplacer sur le segment à la même vitesse. Le but du mémoire est de trouver le parcours d'exploration optimal c'est-à-dire qui s'effectue dans un temps minimal. Le projet vise à approfondir les recherches de Lessard [30] afin de proposer une solution qui permettrait l'emploi d'un nombre quelconque de robots. Nous avons proposé deux algorithmes pour résoudre le problème. Le premier algorithme TA calcule le temps minimal (nécessaire pour compléter l'exploration) avec une précision donnée en paramètre. Le deuxième algorithme TE calcule le temps minimal exact nécessaire pour effectuer un parcours d'exploration. Nous prouvons l'exactitude et la complexité de chaque algorithme. Nous avons également crée une application pour le système d'exploitation iOS, qui illustre, à des fins didactiques, les algorithmes présentés dans le mémoire. À l'aide d'un écran tactile, un utilisateur peut ajuster la position et la quantité de robots, contrôler le temps disponible par robot et voir la trajectoire optimale de chaque robot en temps réel. Il peut aussi voir le fonctionnement de deux algorithmes TA et de TE en suivant chaque itération de la boucle principale de chaque algorithme.

Abstract

The set of n robots is positioned on a line segment in predetermined initial locations. All robots can walk along the segment with the same speed. The purpose of the thesis is to find the trajectories of the robots resulting in the optimal exploration of the segment. The project aims to build upon Lessard [30] research to propose a solution that would allow the use of any number of robots. We proposed two algorithms to solve the problem. The first algorithm TA computes the minimum time (required to complete the exploration) with accuracy given as a parameter. The second algorithm TE calculates the exact minimum time required to complete an exploration. We prove the correctness and the complexity of each algorithm. We have also created an application for iOS operating system, which illustrates, for educational purposes, the algorithms presented in the thesis. Using a touchscreen, a user can adjust the positions and the number of robots, control the time available per robot and see the optimal path of each robot in real time. The user can also follow the execution of each algorithm step by step.

Chapitre 1

Introduction

Dans le modèle de l'exploration d'un environnement géométrique, trouver le parcours optimal pour un ou plusieurs robots, constitue un défi fondamental. Ce défi apparaît surtout lorsqu'on cherche à déterminer le parcours le plus efficace possible pour les environnements géométriques complexes. Cependant, même dans le cas de l'exploration des environnements simples (par exemple 1-dimensionnels), on est souvent en présence de problèmes algorithmiques intéressants. Dans son mémoire [30], Lessard présente le résultat d'une exploration optimale d'un segment par deux robots. Bien que concluante dans le contexte donné, cette solution s'avère insuffisante dans des situations de la présence d'un nombre de robots supérieur à deux.

Typiquement, les robots mobiles possèdent les quatre capacités suivantes :

— la *mobilité* - le robot se déplace dans différentes directions à une vitesse constante ou à différentes vitesses, sans dépasser la vitesse maximale,

— la *perception de l'environnement* - de façon générale, un robot est capable de détecter une rencontre avec un autre robot, un obstacle ou la bordure de son environnement géométrique, même s'il ne connait pas sa position, et dans un graphe, il détecte le degré du sommet,

— la *communication* - un robot est en mesure de communiquer avec d'autres robots ; par exemple, un robot peut échanger de l'information lors d'une rencontre avec un autre robot, laisser un message sur place ou communiquer à distance ; le type d'information transmis pourrait varier d'un simple message contenant une valeur ou même un bit jusqu'à l'ensemble de la connaissance possédée par un robot,

— la *capacité de calcul* - un robot est muni d'un processeur ayant la capacité de calcul. Dans la plupart des cas, le temps nécessaire aux calculs n'est pas considéré comme une ressource que l'on cherche à optimiser.

Le présent projet vise à approfondir les recherches de Lessard afin de proposer une solution qui permettrait l'emploi d'un nombre quelconque de robots. L'application concrète de cette solution pourrait servir dans d'autres algorithmes.

1.1 Problème

Voici la description du modèle qui servira à nos recherches.

L'environnement est constitué d'un segment sur lequel se trouvent des robots,

modélisés par des points, placés dans des emplacements prédéterminés.

Tous les robots peuvent se déplacer sur le segment à la même vitesse. Par *parcours d'exploration* d'un segment on entend les mouvements des robots qui s'effectuent de telle sorte que chaque point du segment est visité par au moins un robot à un moment donné de son mouvement. L'objectif du problème est de trouver le parcours d'exploration optimal, c'est-à-dire qui s'effectue dans un temps minimal.

Dans notre problème, on suppose qu'une autorité centralisée calcule les trajectoires optimales pour tous les robots. Chaque robot reçoit une instruction sur la direction à suivre et sur la distance à parcourir. En conséquence, dans notre problème, les robots n'ont pas besoin de percevoir l'environnement ni de communiquer entre eux.

1.2 Objectif

Notre objectif est de trouver un algorithme qui permettra de déterminer la trajectoire et le temps minimal utilisé par chaque robot de sorte que tout le segment puisse être exploré. À part la solution approximative, nous construisons un algorithme trouvant une solution exacte. L'exactitude et la complexité de cette solution sont prouvées formellement.

Notre algorithme a été programmé et son implémentation en système d'opération iOS est présentée en Annexe A. L'objectif de cette implémentation est d'illustrer le

fonctionnement des algorithmes (approximatif et exact). Cet objectif est à la fois pédagogique et scientifique, puisque la solution, surtout dans le cas de l'algorithme exact n'est pas évidente intuitivement. L'implémentation sera installée sur le serveur dans le cadre de iOS Developer University Program.

1.3 Méthodologie

Notre analyse commence par une analyse du problème dans les cas de plus en plus généraux.

En premier lieu, nous faisons les observations faciles concernant les déplacements d'un seul robot sur le segment. L'étude du second sous-problème sera axée sur un robot se déplaçant sur une demi-droite, ou bien sur un segment dont l'extrémité droite varie. Dans le troisième cas on examinera deux robots sur une demi-droite. Ensuite nous allons construire un algorithme pour trouver une solution pour n-robots, sur une demi-droite. Plus exactement, notre algorithme assume les positions initiales des robots fixes (par rapport au début du segment). Dépendamment de la position du début du segment, qui est un paramètre de notre algorithme, le temps nécessaire pour compléter l'exploration sera déterminé. Cette solution sera appliquée pour résoudre le problème de notre projet - n robots sur un segment de droite.

Chapitre 2

État des connaissances

Le problème de planification des trajectoires (pour une ou plusieurs entités mobiles) attire une attention considérable dans la littérature. De nombreux algorithmes trouvent l'application dans la vie courante. À titre d'exemple, nous pouvons citer les systèmes de surveillance, les actions de sauvetage en montagnes, en mer ou dans des cavernes, ou encore les activités de déneigement, de livraison de colis ou même de l'implémentation des réseaux sans fil.

La tâche consiste en un parcours de l'environnement par les agents mobiles de sorte que dans un temps fini chaque point de l'environnement soit visité par au moins un agent. Diverses variations du problème ont été étudiées, dépendamment, entre autres, du type de l'environnement à traverser ou des capacités des agents mobiles. Dans la plupart des cas, les solutions recherchées visent à optimiser le temps de traversée nécessaire à effectuer le parcours complet de l'environnement.

2.1 Planification de trajectoire

La planification de trajectoire est très présente dans la vie quotidienne. Par exemple, en hiver, dans certaines villes, il faut enlever la neige accumulée. Dans ce cas on a un problème de plusieurs camions qui partent d'un ou plusieurs endroits, ramassent la neige et la transportent vers un ou plusieurs dépôts. Il y a deux vitesses pour chaque camion ; une vitesse plus petite de remplissage et une vitesse plus élevée de transport. La charge de chaque camion est limitée. L'efficacité d'algorithme pour résoudre le problème est le coût total de transport ou le temps nécessaire pour compléter le travail. Le nombre de camions devrait être le plus petit possible pour diminuer les coûts, mais pas trop petit pour permettre de faire le travail dans un temps raisonnable.

Un autre exemple de planification de trajectoire est apparu récemment lors des recherches d'un avion perdu. Le terrain de recherche était très vaste. Chaque machine qui faisait la recherche avait une visibilité réduite et une quantité d'énergie limitée. La cible de recherche était statique.

Encore un autre exemple qu'on peut donner c'est le nettoyage de la mer après une fuite de pétrole. Il faut limiter le plus vite possible la propagation de dégâts.

Il y a aussi des exemples présents dans la vie de tous les jours, comme la livraison de courrier. La planification du trajet d'un facteur est un exemple d'une traversée continue, connue sous le nom de *patrouille*. Chaque jour, il part du même endroit,

visite les points stratégiques (boîtes aux lettres) et retourne à la base.

Dans la littérature, les problèmes de planification de trajectoire ont été étudiés dans plusieurs domaines tel que :

— la *robotique* (la plupart des solutions sont des algorithmes expérimentaux) [1], [2], [12], [14], [13], [32], [34], [35]. Les vrais robots ont été construits et des algorithmes théoriques ont été étudiés et testés dans des situations réelles (voir [2], [32], [33], [35]). Dans la réalité, il est très difficile de créer des conditions idéales, telles que supposées en théorie et pour cette raison il est important de prévoir que les robots peuvent ne pas être fiables ou que la communication entre eux peut être parfois brouillée [2].

— la *géométrie algorithmique*. De nombreux travaux présentés dans [31] étudient différentes méthodes visant à trouver une trajectoire optimale dans l'environnement géométrique. Il y a un nombre illimité de chemins possibles à traverser (par exemple à l'intérieur d'un polygone), ayant des longueurs parfois très variées. Dans l'exemple plus réel il y a des obstacles ou d'autres limitations sous forme de portes d'entrée et de sortie. Des méthodes basées sur la décomposition d'environnement en cellules ou en utilisant une bande élastique (pour contourner des obstacles) ont été étudiées [20].

— la *recherche opérationnelle*. Parfois un problème peut avoir plusieurs solutions. Selon les critères donnés, certaines solutions sont plus efficaces que d'autres.

Prenons l'exemple des travaux [11], [23], [31], où l'optimisation du temps passé pendant le parcours a été prise en considération. Dans [4], [5],[8] on a pour but la minimisation de l'énergie minimale qui est nécessaire à la réalisation de la tâche.

— l'*algorithmique et aussi l'algorithmique distribuée* [3], [7], [6], [9], [28], [37]. Un ou plusieurs robots collaborent afin de parcourir un environnement. Plusieurs robots sont en général plus efficaces qu'un seul robot. Pour améliorer l'efficacité, ils devraient communiquer entre eux. Dans certains cas la communication directe entre les robots pendant une rencontre est utilisée. Une autre possibilité est que des robots communiquent en laissant l'information dans l'environnement (voir par ex. [38]). Dans de tels cas il faut supposer que les éléments de l'environnement (par ex. les sommets ou les arêtes d'un graphe) sont équipés des éléments de mémoire que les robots sont capables de lire et de modifier. Équiper les robots d'un moyen de communication complique généralement la construction des robots. Dans certains travaux on essaye de minimiser le coût de la communication entre les robots ou bien de l'éviter tout simplement (par ex. [22], [25]).

2.2 Évaluation d'efficacité des algorithmes impliquant des agents mobiles

Ce mémoire, ainsi que la plupart des travaux s'y référant, concerne les algorithmes déterministes, c'est-à-dire les algorithmes qui produisent toujours les mêmes résultats pour les mêmes ensembles de données. Dans la plupart de travaux présentant les algorithmes déterministes leur efficacité est mesurée *en pire cas*, c'est-à-dire lorsque les données d'entrée sont les plus défavorables pour l'algorithme. Souvent, cette performance est paramétrée par la taille de données d'entrée ou la valeur d'une donnée d'entrée critique. Cependant, pour certains autres types d'algorithmes comme expérimentaux, heuristiques, etc, leurs performances peuvent être évaluées d'une autre façon, par exemple en moyenne. Les algorithmes randomisés (probabilistes) sont par contre évalués ayant en vue leurs performances espérées.

Pour les algorithmes impliquant un robot mobile, le critère d'efficacité est, la plupart du temps, la longueur de la trajectoire traversée, ce qui normalement équivaut au temps consacré au parcours ([3], [6], [17], [20], [23], [24], [26], [31]). Dans certains travaux de recherche, la question posée est la faisabilité du problème, c'est-à-dire est-ce que les capacités du (ou des) robot(s) sont suffisantes pour que la tâche demandée soit réalisable ([5], [9], [17], [22], [23]). En particulier dans [9] les auteurs montrent qu'un jeton est suffisant pour explorer un arbre par un robot si le robot connaît le nombre de sommets de l'arbre. Pourtant il est prouvé dans [9] qu'il faut $O(log\ log\ n)$

jetons si l'arbre est inconnu.

Les algorithmes pour plusieurs robots abordent aussi la question de la minimisation de la somme de toutes les trajectoires de robots ([13], [31], [35], [36]). Cependant, pour optimiser le temps total du processus, l'objectif recherché est souvent la minimisation du temps maximal (parmi tous les robots) passé en parcours, ce qui implique le partage de la tâche totale de façon la plus uniforme possible entre les robots participants (voir par ex. [5], [7], [13], [17], [30]). Cet objectif de partition uniforme est présent très souvent dans les algorithmes impliquant des équipes de plusieurs robots. On peut l'interpréter comme la minimisation de l'énergie disponible à chaque robot pour la réalisation de la tâche - un sujet important vu une relativement faible évolution de la miniaturisation de piles pour les ordinateurs.

Il existe divers autres critères pour les problèmes impliquant des ensembles de robots, que les chercheurs envisagent dans leurs travaux, comme la capacité de mémoire nécessaire pour réaliser la tâche, comportement en moyenne (surtout dans les heuristiques, les algorithmes expérimentaux ou probabilistes), minimisation de la communication entre les robots, etc. Par exemple Kranakis et al. [29] analyse la complexité de mémoire pour effectuer le rendez-vous de deux agents dans un anneau anonyme de largeur n. En cas de la mémoire d'agent de taille $O(1)$ et de la distance d entre les robots, telle que $d <= \frac{n}{2}$, leur rendez-vous est assuré en $\frac{d}{2(n-d)}$ rondes. En cas de la mémoire disponible de $O(log\ n)$ bits, pour $d > 0$ et n impair il suffit n rondes pour le rendez-vous. En cas de k bits de mémoire accessible et de la distance

$d > 0$ et n paire, les robots se rencontrent après $O\left(\left[\frac{n}{2^{2^k}}\right]^2 2^{2^k}\right)$ rondes. Concernant la borne inférieure, il est montré dans [29] que pour n'importe quel algorithm assurant un rendez-vous de 2 agents dans $O(n)$ rondes, le nombre de $\Omega(log\ log\ n)$ bits de mémoire est nécessaire.

2.3 Environnement géométrique ou graphe

La littérature concernée peut être regroupée en fonction de l'environnement dans lequel les agents opèrent. Les deux principaux types d'environnement sont : *l'environnement géométrique* et le *graphe*.

Un environnement géométrique peut être limité : unidimensionnel (1D), en forme de segment [17] [30], 2D en forme de figure géométrique comme un polygone [28] ou un cercle, ou encore multidimensionnel. Une autre possibilité est un environnement illimité : 1D une ligne [8], 2D un plan [6] etc. Les environnements 3D et plus sont rarement présents dans la littérature.

Dans un environnement géométrique, l'entité mobile est appelée *robot*. Le robot peut se déplacer vers n'importe quel point dans l'environnement.

Dans un graphe, l'entité mobile est appelée *agent*. L'agent peut se déplacer entre les sommets selon la direction des arcs, dans un graphe orienté, ou en empruntant les arêtes, dans un graphe non orienté. La distance entre deux sommets connectés par un arc ou une arête est la même (voir par ex. [3] [10] [24] [36]).

Certains types d'environnements sont *hybrides* en regroupant les propriétés de graphes avec les caractéristiques de l'environnement géométrique. Un exemple d'un tel environnement est un graphe pondéré où avec chaque arc (ou arête) est associé une valeur. Cette valeur peut représenter une distance entre les sommets ([5], [24]) ou le temps à parcourir, ou un autre coût associé au passage de l'arc entre les sommets. Le robot peut se déplacer en suivant des liens, mais, on admet les arrêts de robots à l'intérieur de l'arc (voir [5], par exemple).

Pour simplifier le problème, l'environnement géométrique est souvent modélisé par un graphe, hybride dans la plupart des cas.

2.4 Environnement connu ou non connu

La connaissance de l'environnement est un attribut affectant l'algorithme de planification de trajectoire. Dans le cas de l'environnement connu, la carte de l'environnement peut être consultée par un agent et un algorithme optimal peut être souvent offert. L'exploration dans un environnement connu est d'habitude appelée une *recherche* (voir [17], [30]).

Dans le cas d'environnement inconnu, la plupart du temps il manque d'information pour calculer un algorithme de parcours optimal. Deux approches sont possibles. La première consiste à faire passer un robot afin d'explorer un environnement et effectuer une recherche par la suite (voir [1], [3] par exemple). La deuxième approche

consiste à créer une carte de l'environnement en temps réel et à ajuster le parcours en conséquence (voir [3], [12], [20], [32]). Par exemple, Albers et al. [3] propose le premier algorithm sous-exponentiel pour la construction de la carte d'un graphe inconnu. Dans leur algorithme la borne supérieure est $d^{O(log\ d)}m$, et la borne inférieure est $d^{\Omega(log\ d)}m$, où m est le nombre d'arêtes dans un graphe et d est le nombre minimal d'arêtes à ajouter au graphe pour qu'il devient Eulérien.

2.5 Un ou plusieurs robots

La littérature mentionne des algorithmes de base pour l'exploration par un robot (voir [3], [6], [8], [6] par exemple). Dans ce type d'environnement le robot est le seul à faire tout le travail. Si l'environnement est connu, le chemin optimal peut être précalculé en avance (voir [24], [31]).

L'exploration par un groupe de robots devrait être, en principe, plus efficace. Néanmoins, un nouveau problème apparaît qui relève de la communication entre les robots et la distribution du travail entre eux (voir [2], [12], [14], [13], [33], [34] par exemple). La communication entre les robots peut se faire au moment de la rencontre (par ex. [5]), par des messages laissés dans l'environnement ([9], [38]) ou encore à distance ([14], [36]). Il peut également ne pas y avoir de communication du tout ([22], [30]). Dans [5], des agents échangent toute l'information avec un autre agent rencontré.

L'algorithme peut être centralisé, où les mouvements sont planifiés par une autorité centrale ([15], [27], [30], [36]). Un autre type d'algorithme est un algorithme distribué où les robots décident quoi faire selon l'information obtenue, en analysant l'information reçue grâce à une rencontre ou à un message ([9], [38]). Dans [13], les auteurs analysent la portée de la communication entre les robots limitée par la distance. Dans un autre exemple [2] la perte de communication temporaire, en raison des interférences, est analysée.

2.6 Catégorie de problèmes de recherche et d'exploration

La littérature sur la recherche et l'exploration des environnements géométriques ou graphes est très vaste (par ex. [3], [6], [8], [10], [11], [14], [13], [24], [28]). L'objectif cherché est un parcours complet de l'environnement, pas seulement pour effectuer une recherche, mais par exemple pour créer sa carte afin de faciliter les traversées subséquentes (voir [3], [32], [37]).

Le parcours de l'environnement connu par un robot est souvent appelé *traversée* (voir par ex. [11], [24], [30]). Dans la plupart des cas elle est faite dans un but précis, qui est souvent la recherche d'un objet se trouvant dans une position inconnue de l'environnement ([6], [8], [9], [10], [11], [16], [23], [24], [26]). L'objet cherché par des robots peut être stable ([11], [20], [23], [24]) ou mobile ([16], [23], [24], [26]).

La littérature au sujet de la *poursuite* d'un objet mobile est très riche, surtout celle développée dans les quarante dernières années (voir [23], [24]). La poursuite - c'est une recherche d'objet en mouvement. C'est le type d'algorithmes de type *cops & robbers* (ex. [23], [24], [26]), lion & homme (par ex. [16]) - version géométrique de *cops & robbers*, etc. Un objet peut changer d'emplacement à l'aide de différents niveaux d'intelligence (par ex. [24]). L'objet peut être visible (un hélicoptère) ou invisible (une cave) [11].

Chemin le plus court - dans [31] on se penche sur la façon de calculer le chemin le plus court dans un environnement géométrique. Divers cas sont analysés, le premier où la carte de l'environnement est connue, ainsi que les obstacles, et le chemin le plus court peut être calculé. Le deuxième cas suppose que l'environnement n'est pas connu et un robot découvre la présence d'un obstacle lorsqu'il tombe dessus. De plus, on étudie les divers cas où le problème du chemin le plus court se réduit à une variante du problème de voyageur de commerce.

2.7 Environnements 1-dimensionnels

Beaucoup d'algorithmes de recherche ont été étudiés dans le contexte d'environnements 1-dimensionnels (segment de droite, droite ou demi-droite, cercle), et malgré la simplicité de l'environnement, plusieurs solutions intéressantes ont été proposées (voir [5], [8], [15], [17], [18], [22], [27], [30]). La plupart de travaux présentés dans

le reste de ce survol de littérature concernent les robots qui se déplacent dans les environnements 1-dimensionnels. Par exemple, dans [22] les auteurs analysent l'exploration de l'anneau anonyme. Ils montrent que $O(logn)$ robots peuvent explorer l'anneau de largeur n. Ils prouvent aussi que pour n suffisamment grand il faut au moins $\Omega(logn)$ robots. Ils observent que si n est un nombre premier plus grand que 17, le nombre de robots suffisant pour explorer l'anneau est toujours égale à 17.

Récemment, certains travaux de recherche ont été consacrés à la traversée perpétuelle de l'environnement, nommée *patrouille*. Pendant une dizaine d'années, la communauté de la robotique a étudié le problème de la patrouille de frontière (voir par ex. [34], [35]), en analysant la patrouille surtout d'une perspective expérimentale. Dernièrement, les problèmes de patrouille de frontière ont été étudiés théoriquement dans les travaux en algorithmique ([10], [15], [18], [27], [38]). Le but du problème est d'empêcher un intrus de pénétrer dans une région bornée par une frontière. L'intrus nécessite un temps minimum pour traverser la frontière, mais il peut choisir le point et le moment de son entrée. Conséquemment, les algorithmes de patrouille cherchent à minimiser le temps de revisite de chaque point de la frontière. On étudie les robots ayant la même vitesse ([10], [38]) ou les vitesses possiblement distinctes ([15], [18], [27]) de traversée.

Dans [17], les auteurs étudient les robots qui peuvent être dans l'un des deux états : l'état d'exploration (le robot observe l'environnement en se déplaçant sans dépasser sa vitesse d'exploration) ou l'état de traversée (le robot se déplace sans

observer l'environnement, ne dépassant pas sa vitesse de déplacement, plus grande que la vitesse d'exploration). Un algorithme optimal, fonctionnant en temps $O(n)$, centralisé et l'algorithme en ligne sont proposés en [17] pour les ensembles de robots ayant des paires de vitesses possiblement distinctes.

Le problème de communication entre les robots mobiles a été étudié dans [5], où les robots sont initialement placés sur la ligne dans les positions arbitraires. Les robots traversent la distance proportionnelle à l'énergie utilisée. Deux robots échangent toute l'information qu'il possèdent au moment de leur rencontre. On cherche à minimiser l'énergie attribuée à chaque robot afin que l'information possédée par un robot se répande à tous les autres robots de la collection (problème de diffusion) ou l'union de l'information de tous les robots arrive à un robot de la collection (problème de convergence).

L'article [5] analyse le problème de l'échange de l'information entre des agents mobiles en réseau. Le but est de transférer l'information initiale que chaque agent possède de telle sorte qu'un seul agent recueille l'information de tous les autres agents. Chaque agent possède une certaine quantité d'énergie. La question est de savoir quelle est la quantité d'énergie minimale nécessaire pour atteindre le but, c'est-à-dire, obtenir l'information visée par un agent.

On étudie deux cas : centralisé et distribué. Dans le premier cas, la question est de savoir s'il existe une solution pour une quantité d'énergie donnée, dans le deuxième, comment calculer la quantité d'énergie optimale suffisante pour atteindre l'objectif.

Dans le cas centralisé on présente dans [5] l'algorithme optimal, fonctionnant en temps $O(n)$. Dans le cas distribué, tous les robots sont identiques et effectuent le même programme. Ils ne connaissent pas le réseau et ne savent pas combien il y a d'agents. Les auteurs proposent une solution avec ratio de compétitivité 2 pour des agents qui se trouvent sur une ligne. Le problème de calculer la quantité exacte de l'énergie pour faire l'échange de l'information se complique dans le cas d'un arbre. Les auteurs de [5] prouvent que ce problème est NP-complet.

Le problème de *localisation* pour les *robots bondissants* a été étudié dans [19], [21] et [25]. Les robots se trouvant sur le cercle, sur la ligne ou sur un segment, n'ont aucun contrôle sur leurs mouvements. Ils se déplacent à la même vitesse ([19], [25] ou aux vitesses possiblement distinctes [21] et rebondissent en rencontrant d'autres robots selon les lois de physique. Les robots peuvent enregistrer leurs moments de rebondissement et ils doivent utiliser ces mesures pour déterminer l'existence et les positions de tous les autres robots de la collection. On caractérise dans [19] et [25] toutes les configurations où le problème de localisation est faisable pour les robots ayant la même vitesse et se trouvent sur le segment et sur le cercle. Dans [21] on caractérise toutes les configurations sur le cercle qui mènent à la faisabilité du problème de localisation pour les robots ayant des vitesses quelconques.

Chapitre 3

Les bases de l'algorithme d'exploration

Considérons un segment $S = [0, L]$. Plusieurs robots mobiles $R_1, R_2, \ldots, R_n$ se déplacent à une vitesse constante sur le segment S. Un robot R_i se trouve à une distance r_i du début du segment S, c'est-à-dire r_j dénote la position initiale du robot R_j, pour $j = 1, 2, \ldots, n$ et que $r_1 < r_2 \cdots < r_n$. Il est important d'observer, que dans notre texte, les notions de temps, d'énergie et de distance sont commensurables. Autrement dit, on assumera que dans le temps 1, le robot traverse une distance 1, utilisant 1 unité d'énergie.

3.1 Parcours par un robot

Définition 1 *Le parcours d'un robot R_j (pour $j = 1, 2, ..., n$) est une fonction continue (voir aussi [30])*

$$f_j : [0, T] \to [0, L] \tag{3.1}$$

telle qu'il existe des moments dans le temps

$$t_0, t_1, t_2, \ldots, t_k \in [0, T] \tag{3.2}$$

tels que $\forall x \leq |t_{i+1} - t_i|$,

$$f_j(t_i + x) = \begin{cases} f_j(t_i), \textit{ ou} \\ f_j(t_i) + x, \textit{ ou bien} \\ f_j(t_i) - x \end{cases} \tag{3.3}$$

Le coût du parcours f_j est égal à t_k, soit la durée du parcours du robot R_j.

Observons que dans intervalle de temps $[t_i, t_{j+1}]$, $i = 0, 1, \ldots, k-1$ la vitesse de robot est égale à 0, 1 ou -1, c'est-à-dire le robot reste immobile, se déplace à une vitesse unitaire de gauche à droite ou bien il se déplace à une vitesse unitaire de droite à gauche.

Considérons l'ensemble de points $I_i(t)$ visités par le robot R_i durant l'intervalle de temps $[0, t]$. Clairement $I_i(t)$ est un segment contenant la position initiale du robot. Conséquemment on dénotera

$$I_i(t) = [g_i(t), d_i(t)] \tag{3.4}$$

avec $g_i(t) \leqslant r_i \leqslant d_i(t)$ pour $t > 0$ et $i = 1, 2, \ldots, n$.

Pour simplicité, si t est égal au temps du parcours T, cet ensemble de points sera nommé l'*intervalle d'opération* du robot R_i et dénoté $I_i = [g_i, d_i]$.

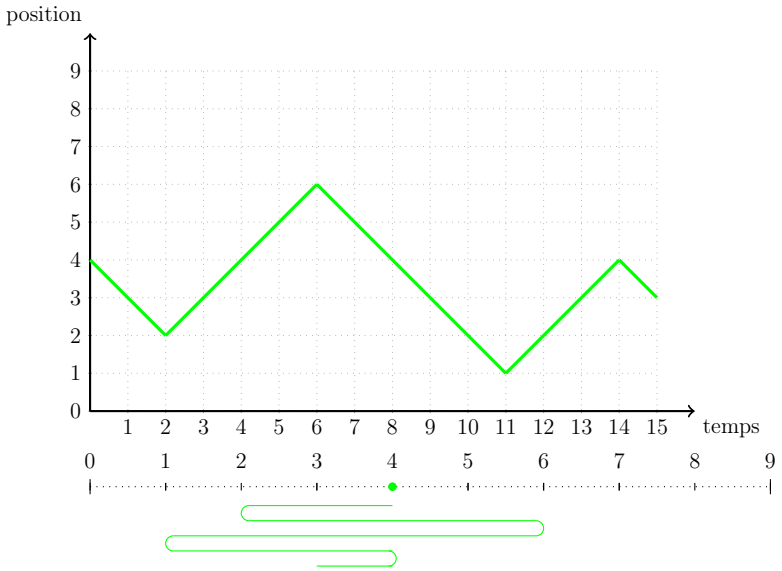


FIGURE 3.1 – Exemple de parcours d'un robot

Maintenant, nous allons introduire la notion de visite d'un point. Un point est considéré visité si un des robots se trouve dans ce point, à un moment donné.

Définition 2 *On dit qu'un point* $p \in [0, L]$ *est* visité *par le robot* R_j *effectuant le parcours* f_j *au moment* t*, ssi* $f_j(t) = p$.

Sur la figure 3.1 tous les points $p \in [1, 6]$ ont été visités. Les points $p \in [0, 1)$ et $p \in (6, 9]$ n'ont pas été visités.

Le lemme définit un parcours le moins couteux d'un segment par un robot mobile.

Lemme 1 *(voir [30])*

Le parcours optimal d'un segment par un robot consiste à se diriger tout droit vers l'extrémité la plus proche du robot et ensuite parcourir entièrement le segment en terminant le parcours à l'autre extrémité du segment.

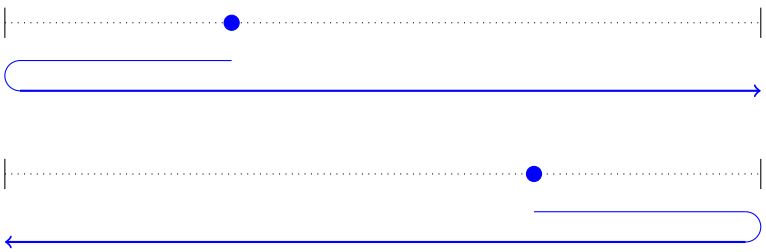


FIGURE 3.2 – Les parcours optimaux

La figure 3.2 présente deux scénarios possibles pour un parcours optimal.

On peut observer que si le robot est à l'extrémité du segment à explorer la première partie du parcours se réduit à nulle.

3.2 Parcours collectif

Par un *parcours collectif*, on va comprendre une suite de tous les parcours de tous les robots. La définition suivante présente ceci formellement.

Définition 3 *Un* parcours collectif *est une suite de parcours de robots* $(f_1, f_2, \ldots, f_n)$ *tel que pour chaque* $j = 1, 2, \ldots, n$ $f_j(0) = r_j$.

Dans la suite de ce texte, quand l'ensemble des robots $R_1, R_2, ..., R_n$, participant au parcours collectif est connu du contexte, un parcours collectif sera nommé simplement un *parcours*.

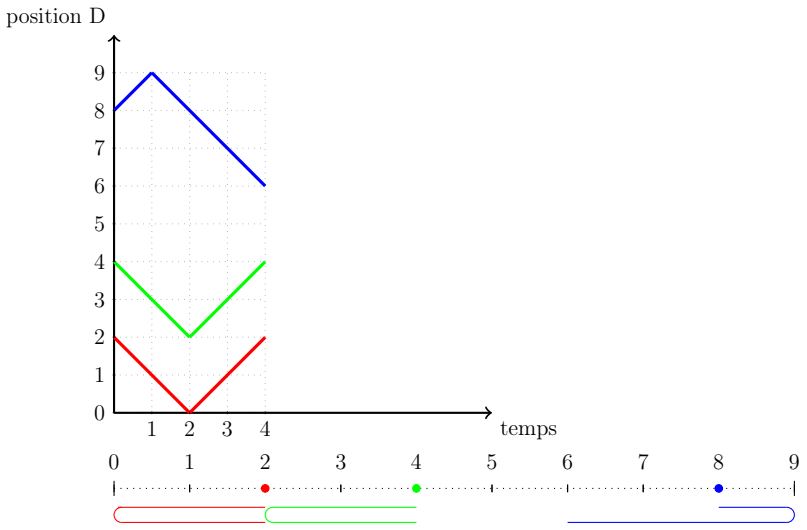


FIGURE 3.3 – Exemple d'un parcours (collectif)

La prochaine définition introduit la notion de parcours *d'exploration*, c'est-à-dire, tel qu'à sa fin le segment est complètement exploré.

Définition 4 *Pour un ensemble de robots* $R_1, R_2, ..., R_n$ *le parcours collectif est nommé un parcours d'exploration, si* $\forall\, p \in [0, L]\ \exists_j\ 1 \leqslant j \leqslant n\ \exists\, t \in [0, T]$ *où* $f_j(t) = p$.

La figure 3.3 représente un parcours qui n'est pas un parcours d'exploration. Il ne satisfait pas aux exigences définies dans la définition 4, parce qu'il existe au moins un point $p \in L$, qui n'est visité par aucun robot. Dans l'exemple, ce sont les points $p \in (4, 6)$.

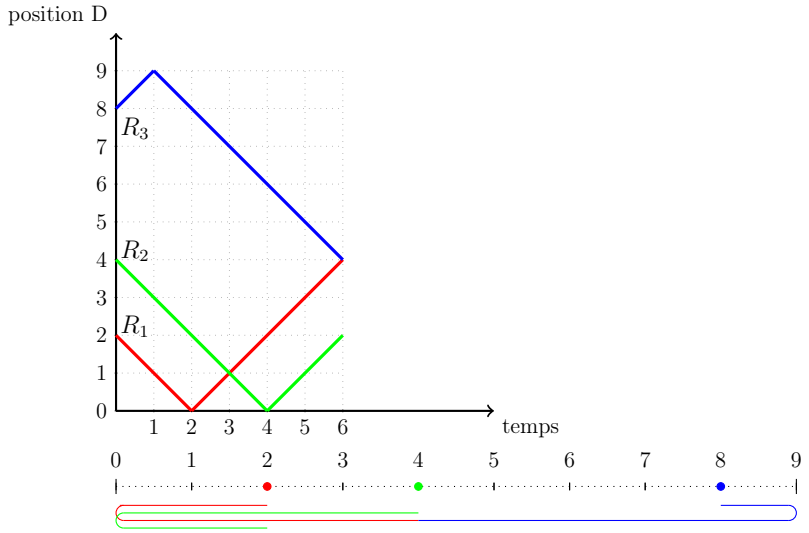


FIGURE 3.4 – Parcours d'exploration

La figure 3.4 représente un parcours d'exploration.

Les robots commencent leurs parcours simultanément donc le robot qui passe le plus de temps dénoté par $|f_i|$ détermine la durée du parcours. Ce temps sera appelé *coût du parcours d'exploration.*

Définition 5 *Le* coût *du parcours d'exploration* $F = (f_1, f_2, \dots f_n)$ *dénoté par* $|F|$ *est le maximum parmi les coûts des parcours* $f_1, f_2, \dots f_n$.

$$|F| = \max_{1 \leqslant i \leqslant n} |f_i| \tag{3.5}$$

3.3 Parcours sans interférences

Si les robots arrivant au même point interchangent leur ordre sur le segment, on va dire que ceci crée un *croisement* (dans leurs parcours).

Définition 6 *Un parcours d'exploration* $(f_1, f_2, \dots f_n)$ *est dit* sans croisement *ssi* $\forall\, t \in [0, T],\ \forall\, i \in [1, n-1]\ \ f_i(t) \leq f_{i+1}(t)$

La figure 3.4 représente un parcours avec croisements. On voit que les robots R_1 et R_2 arrivent en temps 3 au point 1 et ils interchangent leur ordre.

Le nombre de croisements d'un parcours d'exploration est le nombre de fois les paires de robots changent leur ordre sur le segment.

Corollaire 1 *Pour le parcours sans croisement nous avons* $g_i \leqslant g_{i+1}$ *et* $d_i \leqslant d_{i+1}$, *pour* $i = 1, 2, \dots, n-1$

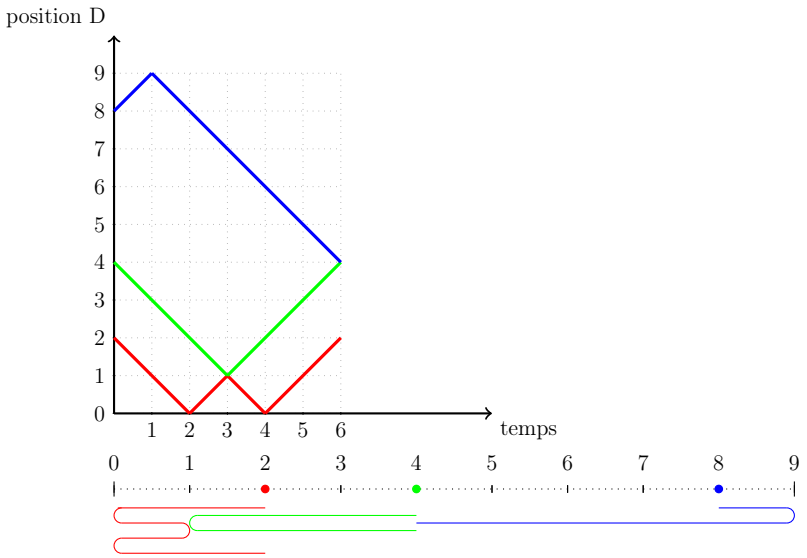


FIGURE 3.5 – Parcours sans croisements

Il est facile de voir, que pour chaque parcours d'exploration il existe un parcours d'exploration sans intersection ayant le même coût. Effectivement, chaque fois les robots interchangent, créant une intersection, on peut changer les rôles des robots et éliminer l'intersection. De cette façon on peut éliminer successivement toutes les intersections en obtenant un parcours du même coût. Conséquemment, nous avons :

Lemme 2 *Pour chaque parcours d'exploration il existe un parcours d'exploration sans croisement tel que le coût du parcours d'exploration* $(f'_1, f'_2, ..., f'_n)$ *est le même que celui de* $(f_1, f_2, ..., f_n)$.

Le parcours présenté dans la figure 3.5 va être appelé *parcours avec interférences*.

Un parcours d'exploration est sans interférences si aucun intervalle n'est visité par plus d'un robot. Un parcours d'exploration qui n'a pas d'interférences va être appelé *parcours sans interférences.*

Définition 7 *Un parcours d'exploration* $(f_1, f_2, \dots f_n)$ *est dit* sans interférences *ssi* $\forall\, t_1, t_2 \in [0, T],\ \forall\, i \in [1, n-1]\ \ f_i(t_1) \leq f_{i+1}(t_2)$

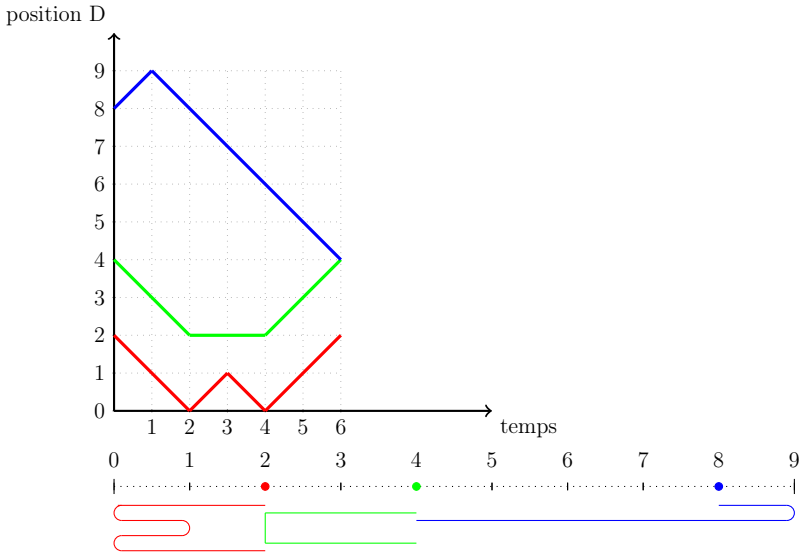


FIGURE 3.6 – Parcours sans interférences

Nous avons le corollaire suivant :

Corollaire 2 *Pour un parcours d'exploration sans interférences les intervalles d'opération des robots ont des intérieurs disjoints, c'est-à-dire*

$$d_i = g_{i+1},\ \ pour\ \ i = 1, 2, \dots, n-1$$

Lemme 3 *Pour chaque parcours d'exploration $F = (f_1, f_2, \ldots, f_n)$ il existe un parcours d'exploration $F' = (f'_1, f'_2, \ldots, f'_n)$ sans interférences tel que le coût d'exploration de F' n'a pas un coût plus élevé que le coût de F.*

Démonstration : Par Lemme 2 nous pouvons supposer que le parcours F est sans croisements. Comme F est un parcours d'exploration

$$\bigcup_{j=1}^{n} I_j \supseteq [0, L] \tag{3.6}$$

Selon Corollaire 1 nous avons $d_j \geqslant g_{j+1}$, pour $j = 1, 2, \ldots, n$. Prenons le plus petit indice i, tel que le robot R_i est impliqué dans une interférence, c'est-à-dire tel que $d_j = g_{j+1}$, pour $j = 1, 2, \ldots, n-1$ et $d_i > g_{j+1}$. Observons que les positions initiales r_i, r_{i+1} des robots R_i et R_{j+1} respectent les conditions $r_j \leqslant r_{i+1}$, $r_j \leqslant d_i$ et $r_{j+1} \leqslant g_{i+1}$. Conséquemment l'intersection des segments $[g_{i+1}, d_i]$ et $[r_i, r_{i+1}]$ est non vide. Prenons un point p, tel que

$$p \in [g_{i+1}, d_i] \cap [r_i, r_{i+1}] \tag{3.7}$$

Définissons le parcours f'_i de façon suivante (voir figure 3.6)

$$f'_i(t) = \begin{cases} f_i(t) & \text{si } f_i(t) \leqslant p \\ p & \text{si } f_i(t) > p \end{cases} \tag{3.8}$$

Observons que $f'_i(t)$ est un parcours de R_i bien défini. Comme $r_i \leqslant p$ et $f_i(0) = r_i$ nous avons $f'_i(0) = r_i$. La fonction $f_i(t)$ étant continue dans l'intervalle $[0, T]$, la fonction$f'_i(t)$ y est aussi continue. Finalement, la vitesse de parcours $f'_i(t)$ du robot R_i est soit celle du parcours $f_i(t)$ ou bien, elle est nulle (le robot reste au point p). Alors f_i respecte toutes les conditions d'un parcours correct.

Pour chaque $j > i$ définissons le parcours f'_j de façon suivante

$$f'_j(t) = \begin{cases} f_j(t) & \text{si } f_j(t) \geqslant p \\ p & \text{si } f_j(t) < p \end{cases} \tag{3.9}$$

Observons que la fonction $f'_j(t)$ est un parcours de R_j bien défini. Effectivement $f'_j(0) = r_j$, f_j est continue et la vitesse de R_j est toujours 0, 1 ou -1.

Conséquemment pour tout $j = 1, 2, \ldots, n$, f'_j est un parcours bien défini du robot R_j. Observons que F' est un parcours d'exploration. Effectivement, les parcours $f'_1, f'_2, \ldots, f'_i$ explorent conjointement le sous-segment $[0, p]$, parce qu'ils sont équivalents au parcours $f_1, f_2, \ldots, f_i$ dans ce sous-segment. Pour la même raison les parcours $f'_{i+1}, \ldots, f'_n$ explorent le sous-segment $[p, L]$.

Comme le coût de f'_j est le même que le coût de f_j, pour $j = 1, 2, \ldots, n$, le coût de F' est égal à celui de F. □

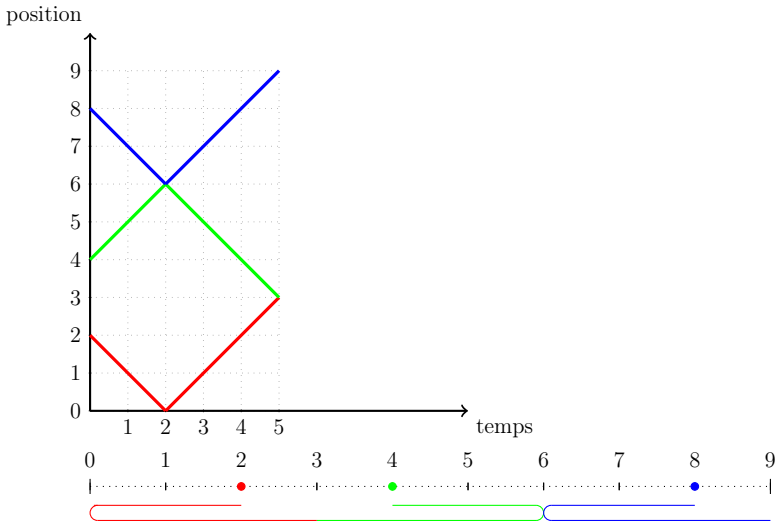


FIGURE 3.7 – Parcours correct, sans croisements, sans interférences

La figure 3.7 représente un parcours d'exploration correct. Tous les points $p \in D$ sont visités. C'est un exemple de parcours sans croisements. Il n'y a pas de répétitions. Les robots font demi tour au maximum une fois.

3.4 Parcours optimaux

Le nombre de parcours d'exploration par robots est illimité. Certains parcours sont plus couteux que d'autres. Le parcours d'exploration avec le coût le plus petit va être nommé *parcours optimal*.

Définition 8 *Un parcours d'exploration* $(f_1, f_2, \dots f_n)$ *est dit* optimal *s'il n'existe pas un parcours d'exploration* $(f'_1, f'_2, \dots f'_n)$ *ayant un coût plus petit.*

La figure 3.7 représente un parcours non optimal parce qu'il existe un parcours ayant un coût plus petit.

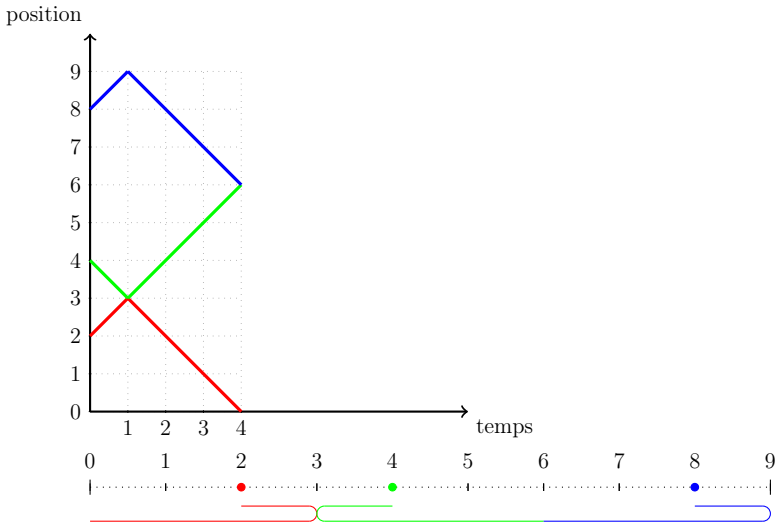


FIGURE 3.8 – Parcours optimal

La figure 3.8 représente un parcours d'exploration optimal. Tous les points $p \in L$ sont visités. Il est possible de prouver qu'il n'existe pas un parcours ayant un coût plus petit.

Définition 9 *Le parcours* $F = (f_1, f_2, \ldots, f_n)$ *est un parcours* régulier *ssi* F *est sans interférences et pour chaque* $i = 1, 2, \ldots, n$ *le robot* R_i *explorant son intervalle d'opération* $I_i = [g_i, d_i]$

1. *D'abord marche à sa vitesse maximale vers l'extrémité de* I_i *plus proche de sa position initiale* r_i *(marche vers* g_i *si* r_i *est au milieu de* I_i*)*
2. *Ensuite marche à la vitesse maximale vers l'autre extrémité*

3. Reste immobile après

Théorème 1 *Il existe un parcours régulier $F = (f_1, f_2, \ldots, f_n)$ optimal.*

Démonstration : Comme l'ensemble de parcours de robot est défini sur un ensemble compact (fermé et borné) il admet une valeur ayant le maximum minimal. Conséquemment, il existe un parcours optimal $F' = (f'_1, f'_2, \ldots, f'_n)$. Par Lemme 3 nous pouvons supposer que le parcours F' est sans interférences. Définissions un parcours régulier $F = (f_1, f_2, \ldots, f_n)$ tel que pour chaque $i = 1, 2, \ldots, n$ f_i explore le même sous-segment que f'_i.

Par Lemme 1 $|f_i| \leqslant |f'_i|$ alors $|F| = \max\limits_{1 \leqslant i \leqslant n} |f_i| \leqslant \max\limits_{1 \leqslant i \leqslant n} |f'_i| = |F'|$. □

Chapitre 4

Algorithme approximatif

Dans ce chapitre nous présentons un algorithme qui calcule le temps optimal nécessaire pour l'exploration du segment de façon approximative. Le temps d'exécution de notre algorithme dépend de la précision désirée.

4.1 Fonctions utilisées par l'algorithme

Nous présentons d'abord une fonction $ExploPossible(T)$, qui vérifie si le temps T est suffisant pour l'exploration du segment $[0, L]$ donné. L'ingrédient important de cette fonction est la fonction *PortéeRobot(i, G, T)*. La fonction *PortéeRobot(i, G, T)* suppose que le robot R_i, durant le temps T doit explorer un segment ayant son extrémité gauche au point G. La fonction détermine une trajectoire permettant au robot d'explorer le segment le plus long possible.

fonction *PortéeRobot(i, G, T)* $\rightarrow$ *(D)*

1. $\delta \leftarrow (r_i - G)$
2. **si** $\delta \leqslant 0$ **alors**
3. $\quad D \leftarrow r_i + T$
4. **sinon si** $\delta > T$ **alors**
5. $\quad D \leftarrow r_i$
6. **sinon si** $\delta \geqslant \dfrac{T}{3}$ **alors**
7. $\quad D \leftarrow r_i + \dfrac{T - \delta}{2}$
8. **sinon si** $\delta < \dfrac{T}{3}$ **alors**
9. $\quad D \leftarrow r_i + T - 2\delta$
10. **si** $i < n$ **et** $D > r_{i+1}$ **alors**
11. $\quad D \leftarrow r_{i+1}$
12. **retourner** D

Lemme 4 *La fonction PortéeRobot(i, G, T) trouve correctement la plus grande valeur possible D telle que le sous-segment $[G, D]$ est exploré par robot R_i durant le temps T. Si le temps T est insuffisant pour atteindre G à partir de la position initiale de r_i la valeur r_i est retournée.*

Démonstration : Le robot R_i se trouve à la position r_i. Si l'extrémité gauche G se trouve après ou au point r_i, le robot R_i doit se déplacer seulement vers la droite, donc le point exploré le plus à droite sera égal au temps utilisé pour l'exploration à partir du point r_i, donc $r_i + T$ (lignes 2-3).

Quand la position $r_i > G$, pour que les intervalles d'opération I_i et I_{i-1} soient

connectés, le robot doit explorer au moins la distance $\delta = r_i - G$. Si cette distance dépasse le temps disponible aucune exploration n'est possible (la valeur r_i est retournée, voir lignes 4-5).

Le parcours optimal du robot R_i dépend de la distance δ et de la quantité de temps disponible T. Selon le lemme 1 il y a deux possibilités : marcher vers G et ensuite vers D ou marcher vers D et ensuite vers G.

Dans le premier cas, le robot parcourt la distance δ et ensuite marche jusqu'à l'épuisement du temps en établissant la nouvelle extrémité D. Donc, il utilise deux fois δ pour arriver au point de départ. Alors $D = r_i + T - 2\delta$ (voir lignes 8-9).

Dans le deuxième cas on calcule d'abord le nouveau point D. Le robot R_i marche jusqu'à D, fais demi tour et marche jusqu'à G. Il passe le temps $T - \delta$ pour faire une partie à partir de r_i jusqu'à D. Le D sera égal à $r_i + (T - \delta)/2$ (voir lignes 6-7).

Comme la portée de robot ne dépasse pas la position initiale du robot subséquent, les lignes 10-11 de la fonction font la vérification pour retourner la valeur qui vérifie cette condition. □

Présentons maintenant la fonction *ExploPossible*, qui retourne le résultat logique "vrai" si et seulement si le temps T est suffisant pour explorer entièrement le segment.

fonction *ExploPossible(T)* $\rightarrow$ *(B)*

1. $Gauche \leftarrow 0$
2. **pour** $i = 1$ **à** n **faire**
3. **si** $r_i - Gauche > T$
4. **retourner** $faux$
5. $Droite \leftarrow$ PortéeRobot$(i, Gauche, T)$
6. $Gauche \leftarrow Droite$
7. **retourner** $Gauche \geqslant L$

Lemme 5 *La fonction ExploPossible(T) vérifie correctement si les robots* $R_1, R_2, \ldots, R_n$*, chacun pendant le temps* T*, sont capables d'explorer le segment* $[0, L]$

Démonstration : On prouve par induction l'assertion suivante : la valeur *Gauche* de la i-ème itération de la boucle **pour**, obtenue à la ligne 6 est égale à la valeur L_i, telle que les robots $R_1, R_2, \ldots, R_i$ durant le temps T explorent collectivement le plus long possible segment $[0, L_i]$.

Comme base d'induction on peut supposer qu'un robot virtuel R_0 explore exactement le point 0. Supposons que $r_i - T < 0$ et L_{i-1} est la plus grande valeur, telle que le segment $[0, L_{i-1}]$ est exploré par $R_1, R_2, \ldots, R_{i-1}$. Prenons un robot R_i dans la position initiale r_i. Si le temps T est insuffisant pour atteindre L_{i-1} par R_i (c'est-à-dire $r_i - T > L_{i-1}$) en lignes 3-4 on retourne l'impossibilité de l'exploration puisque le voisinage droit du point L_{i-1} ne pourra jamais être exploré. Dans le cas $r_i - T \leqslant L_{i-1}$, par Lemme 4, R_i explore le segment $[L_{i-1}, L_i]$ qui est le plus long

possible à explorer pendant le temps T, ce qui complète l'induction.

Conséquemment, à la ligne 7 de l'algorithme, à la sortie de la boucle **pour**, *Gauche* est égale à la plus grande valeur telle que les robots $R_1, R_2, \ldots, R_n$ sont capable d'explorer $[0, Gauche]$. Si $Gauche \geqslant L$ alors l'exploration du segment $[0, L]$ est possible sinon elle est impossible. Ceci complète la preuve. □

Il est évident qu'en augmentant le temps T donné à robot 1 le segment maximal $[0, L]$ exploré par ce robot ne peut pas diminuer. En plus la fonction $L_1(T)$ est une fonction continue, puisque l'extrémité gauche du segment exploré par robot 1 est constante. Par induction sur i, la fonction $L_i(T)$ est aussi continue et non-décroissante, donc la valeur $L = L_n(T)$ peut être trouvé par la méthode de dichotomie.

Notons que la quantité de temps moindre que r_1, c'est-à-dire la distance de la position initiale de R_1 au début du segment n'est pas suffisante pour l'exploration. De l'autre côté, pour chaque robot explorant un sous-segment de longueur l, le temps $\frac{3}{2}l$ est suffisant. En conséquence le temps $T = \frac{3}{2}L$ est surement suffisant pour l'exploration.

Notre solution consiste à faire la recherche binaire dans l'intervalle $[r_1, \frac{3}{2}L]$. Dans chaque itération on estime la quantité de temps correspondant au milieu de l'intervalle courant. Ensuite, pour tous les robots, nous vérifions si la quantité de temps est suffisante pour l'exploration. Si le temps est insuffisant, la solution se trouve dans la moitié supérieure de l'intervalle sinon on cherche dans la partie inférieure. On choisit

le milieu du nouveau intervalle et on répète la vérification jusqu'à la précision requise.

L'idée générale de l'algorithme est présentée dans l'algorithme *TA* (Temps approximatif).

4.2 Présentation de l'algorithme

Algorithme TA

1. #Initialiser l'intervalle $[Min, Max]$
   $Min \leftarrow r_1; Max \leftarrow 1.5 * L$
2. **tant que** $Max - Min > Precision$ **faire**
3. Choisir le milieu d'intervalle $Mid \leftarrow (Min + Max)/2$
4. **si** ExploPossible(Mid) **alors**
5. $Max \leftarrow Mid$
6. **sinon**
7. $Min \leftarrow Mid$
8. La réponse est Max

Théorème 2 *L'algorithme TA trouve la quantité minimale de temps, nécessaire pour compléter un parcours par n robots, avec une erreur ϵ. Le temps d'exécution de l'algorithme TA est* $O\left(nlog\frac{L}{\epsilon}\right)$.

Démonstration : La solution se trouve dans l'intervalle $[r_1, \frac{3}{2}L]$. R_1 peut être dans la position minimale égale à 0, donc l'intervalle est de longueur maximale de $\frac{3}{2}L$. Dans chaque itération de la boucle de lignes 2-6 l'intervalle est réduit de moitié. Après un

certain nombre k d'itérations, l'intervalle sera plus petit que l'erreur demandée ϵ provoquant une sortie de la boucle. Cela implique $\frac{\frac{3}{2}L}{2^k} \leqslant \epsilon$. Pour calculer la valeur k on obtient $\frac{3}{2}L \leqslant \epsilon 2^k$, donc $2^k \geqslant \frac{3}{2\epsilon}L$. Conséquemment $log 2^k \geqslant log\frac{3}{2\epsilon}L$, donc $k \geqslant log\frac{3}{2\epsilon}L$, ce qui implique $O\left(log\frac{L}{\epsilon}\right)$ itérations de la boucle 2-7.

Durant chaque itération on appelle la fonction *ExploPossible*, s'exécutant en temps $O(n)$, impliquant le temps d'exécution de l'algorithme TA dans $O\left(nlog\frac{L}{\epsilon}\right)$.

□

Chapitre 5

Algorithme de calcul exact

5.1 L'idée de l'algorithme

Nous allons présenter d'abord l'idée de l'algorithme *TE* (Temps exact).

L'algorithme TE est une "discrétisation" de la simulation programmée pour le système iOS (voir l'annexe A). Imaginons un processus dans lequel tous les robots possèdent la même quantité de temps T, et T augmente de façon continue. Chaque robot de R_1 à R_n, en utilisant son temps, couvre une portion du segment à gauche de sa position initiale, (jusqu'à la position couverte par son prédécesseur). L'excédent de son temps est utilisé pour couvrir une portion à droite de sa position initiale. Le robot utilise ce temps de façon optimale, en essayant de couvrir la portion la plus grande (à droite de sa position initiale) sans dépasser la position initiale de son successeur. Durant ce processus chaque robot change son « état », en marchant

seulement à gauche, d'abord à droite ensuite à gauche, d'abord à gauche ensuite à droite, et finalement uniquement à droite. Durant ce processus, avec l'augmentation de temps T, les sous-segments explorés par les robots consécutifs s'élargissent et les espaces non explorés rétrécissent et finalement disparaissent. Le processus se termine quand le dernier espace non exploré disparait. Le rôle de l'algorithme TE est de suivre l'évolution de ce processus, identifier les changements consécutifs d'états des robots en arrivant à celui qui résulte de la disparition du dernier espace non exploré.

Dénotons par $I(t)$ le sous-ensemble des points du segment $[0, L]$ qui sont explorés en temps t de l'algorithme. $S(t)$ est défini comme

$$I(t) = \bigcup_{i=1}^{n} I_i(t) = \bigcup_{i=1}^{n} [g_i(t), d_i(t)] \tag{5.1}$$

Évidement $I(0) = \{r_1, r_2, \ldots, r_n\}$. Les segments $[g_i(t), d_i(t)]$ évoluent de sorte qu'à la fin de l'algorithme au moment T

$$I(T) \supseteq [0, L] \tag{5.2}$$

On va décrire l'évolution du segment $[g_i(t), d_i(t)]$ exploré par R_i. Si le robot R_{i-1}, n'a pas encore visité le point $g_i(t)$, c'est-à-dire $d_{i-1}(t) < g_i(t)$, le robot R_i marche à gauche.

Lorsque les intervalles d'opération des robots R_{i-1} et R_i se touchent, le robot

R_i essaye en plus d'explorer le plus grand possible segment à droite de sa position initiale r_i. Finalement, quand $d_i(t) = r_{i+1}$, le robot R_i n'explore plus à droite et l'excès d'énergie possédée par R_i devient inutile.

Notons que le robot R_i est toujours dans l'état S_i, qui peut prendre une de quatre valeurs : $0, 1, 2, 3$. Dans l'état $S_i = 0$, R_i marche à gauche ($d_{i-1} < g_i$). Dans l'état 1, R_i marche d'abord à droite, puis à gauche jusqu'à $d_{i-1}(t)$. Dans l'état 2, R_i marche d'abord à gauche et puis il continue à droite dans la direction de r_{i+1}. Finalement lorsque $d_{i-1}(t) = r_i$, c'est-à-dire le segment d'opération de R_{i-1} achève la position initiale r_i de robot R_i, il passe à l'état $S_i = 3$ (observons que R_1 n'arrive jamais à l'état $S_1 = 3$.

Il est important d'observer que le passage de S_1 à S_2 se fait au moment quand $r_i - g_i(t) = d_i(t) - r_i$, c'est-à-dire que la position initiale du robot R_i est exactement au milieu de son segment d'opération.

De cette façon le groupe de robots $R_1, R_2, \ldots, R_i$ essayent de maximiser le segment $[0, D]$ exploré collectivement par $R_1, R_2, \ldots, R_i$.

Pour les parcours optimales il est intéressant d'observer l'évolution en temps t du segment $[g_i(t), d_i(t)]$ étant le segment d'opération du robot i de l'algorithme TE. Au début $g_i(t) = d_i(t) = r_i$. Ensuite $g_j(t)$ se déplace à gauche, ($g_i(t + \Delta) < g_i(t)$). Puis les deux extrémités se déplacent à droite ($g_i(t + \Delta) > g_i(t)$ et $d_i(t + \Delta) > d_i(t)$).

Dans notre algorithme nous allons calculer la vitesse V_i d'évolution de $d_i(t)$:

$$V_i(t) = \frac{d_i(t+\Delta) - d_i(t)}{\Delta} \tag{5.3}$$

L'idée de notre algorithme est la suivante. On garde le vecteur d'états de tous les robots $S_1, S_2, \ldots, S_n$ ainsi que les vitesses $V_i(t)$, pour $i = 1, 2, \ldots, n$. Au début $S_i = V_i = 0$ pour $i = 1, 2, \ldots, n$. Dans la boucle principale itérativement on calcule les moments de temps où chaque robot change son état (dans le tableau Ch). On détermine le plus proche temps c du tableau Ch, on change l'état du robot concerné et on recalcule les nouvelles vitesses $V_i(c)$. L'algorithme se termine quand $g_i = 0$; $d_{i-1} = g_i$, pour $i = 1, 2, \ldots, n$ et $d_n \geqslant L$. Cette dernière condition est contrôlée par l'introduction d'un robot virtuel R_{n+1} qui ne bouge pas et il est artificiellement soit dans l'état 0 (pas atteint par R_n) ou l'état 3 ($d_n(t) \geqslant L$).

L'annexe 6.3 illustre le fonctionnement de cet algorithme pour un exemple.

Dans notre algorithme on suppose l'existence d'un robot virtuel R_0 ayant comme segment d'opération $I_o(t) = [g_0(t), d_o(t)] = [0, 0]$. Nous avons le lemme suivant :

Lemme 6 *La vitesse $V_i(t)$ est définie récursivement de façon suivante :*

$$V_i(t) = \begin{cases} 0 & si\ R_i\ est\ dans\ l'état\ 0 \\ \dfrac{V_{i-1}(t)+1}{2} & si\ R_i\ est\ dans\ l'état\ 1 \\ 2\,V_{i-1}(t)+1 & si\ R_i\ est\ dans\ l'état\ 2 \\ 1 & si\ R_i\ est\ dans\ l'état\ 3 \end{cases} \tag{5.4}$$

où $V_0(t) = 0$

Démonstration : Un robot R_i en état 0 se déplace à gauche alors le point le plus à droite du segment visité par R_i reste le même. Alors

$$d_i(t+\Delta) - d_i(t) = 0$$

et

$$V_i(t) = \frac{d_i(t+\Delta) - d_i(t)}{\Delta} = 0$$

FIGURE 5.1 – Robot en état 1

Considérons maintenant le robot R_i en état 1. Entre le moment t et le moment

$t + \Delta$ la valeur de d_i change de sorte que $d_i(t + \Delta) > d_i(t)$ (voir Figure 5.1). Comme R_i marche à une vitesse 1 la longueur de sa trajectoire au moment $t + \Delta$ dépasse de Δ la longueur de sa trajectoire au moment t. En conséquence nous avons :

$$2(d_i(t + \Delta) - r_i) + r_i - d_{i-1}(t + \Delta) = \Delta + 2(d_i(t) - r_i) + r_i - d_{i-1}(t)$$

alors

$$V_i(t) = \frac{d_i(t + \Delta) - d_i(t)}{\Delta} = \frac{d_{i-1}(t + \Delta) - d_{i-1}(t) + \Delta}{2\Delta} = \frac{v_{i-1}(t) + 1}{2}$$

FIGURE 5.2 – Robot en état 2

Quand le robot R_i se trouve dans l'état 2 (voir Figure 5.2) par l'analyse semblable au cas précédent nous avons la formule :

$$2(r_i - d_{i-1}(t)) + d_i(t) - r_i + \Delta = 2(r_i - d_{i-1}(t + \Delta)) + d_i(t + \Delta) - r_i$$

alors

$$V_i(t) = \frac{d_i(t+\Delta) - d_i(t)}{\Delta} = \frac{2(d_{i-1}(t+\Delta) - d_{i-1}(t)) + \Delta}{\Delta} = 2V_{i-1}(t) + 1$$

Finalement, quand le robot R_i est dans l'état 3, $d_i(t)$ avance à la vitesse de parcours de R_i, c'est-à-dire $V_i(t) = 1$. □

Le corollaire suivant est une conséquence immédiate du Lemme 6.

Corollaire 3 *Si en intervalle de temps $[t, t+\Delta]$ aucun robot $R_1, \ldots, R_n$ ne change pas son état alors la vitesse $V_i(t)$ dans cet intervalle est donnée par la formule*

$$V_i(t) = a_i t + b_i \tag{5.5}$$

pour certaines constantes a_i, b_i, et $i = 1, 2, \ldots, n$

Nous avons le lemme suivant :

Lemme 7 *Chaque fonction $V_i(t)$ peut changer sa formule $O(n)$ fois.*

Démonstration : Chaque robot en état 0 va seulement à gauche de son point initiale. En un moment donné, une fois l'extrémité droite de son prédécesseur est atteinte, l'extrémité gauche de son intervalle d'opération évolue à droite (en fonction du temps T) et le robot reste en état 1. Une fois T atteint trois fois la distance de

sa position initiale à l'extrémité droite de son prédécesseur, le robot passe à l'état 2. Finalement lorsque l'extrémité droite de prédécesseur atteint sa position initiale, le robot passe à l'état 3.

En conséquence, chaque robot peut passer seulement de l'état i à l'état $i+1$, pour $i = 0, 1, 2$ ce qui implique que chaque robot peut changer son état un nombre constant de fois, ce qui implique l'assertion du lemme.

5.2 Procédures et fonctions utilisées par l'algorithme

Avant de présenter l'algorithme exact, nous allons introduire quelques fonctions utilisées par cet algorithme.

D'abord, nous présentons une fonction *ExisteRobotEnÉtatZero(S)*, qui vérifie s'il existe encore un robot en état égal à 0.

fonction *ExisteRobotEnÉtatZero(S)* $\rightarrow$ *(B)*

1. **pour** $i = 1$ **à** $n + 1$ **faire**
2. **si** $S_i = 0$ **alors**
3. **retourner** $vrai$
4. **retourner** $faux$

La deuxième fonction est une fonction $CalculerVitesses()$ qui a pour but de calculer la vitesse V_i d'avancement du point $d_i(t)$, selon l'état actuel S_i du robot R_i explorant le segment $[g_i(t), d_i(t)]$.

procédure *CalculerVitesses(V)*

1. **pour** $i = 1$ **à** n **faire**
2. **si** $S_i = 0$ **alors** $V_i = 0$
3. **si** $S_i = 1$ **alors** $V_i = \dfrac{V_{i-1} + 1}{2}$
4. **si** $S_i = 2$ **alors** $V_i = 2\,V_{i-1} + 1$
5. **si** $S_i = 3$ **alors** $V_i = 1$

La prochaine fonction *ProchainChangement(T)* est la fonction clé de l'algorithme. Cette fonction est exécutée en supposant qu'en temps T, qui est le paramètre de la fonction, les robots se trouvent dans les états $S_1, S_2, \ldots, S_n$. La fonction calcule quel est le prochain moment de temps $TempsCh$ où un des robots change son état. $TempsCh$ et le numéro $iRobotCh$ du robot qui a subi ce changement d'état sont retournés par la fonction.

fonction *ProchainChangement(T)* $\rightarrow$ *(TempsCh, iRobotCh)*

1. $TempsCh \leftarrow \infty$
2. $Gauche \leftarrow 0$
3. **pour** $i = 1$ **à** n **faire**
4. $\quad Ch \leftarrow$ **cas** S_i **de**
5. $\quad\quad 0 : \dfrac{T * V_{i-1} + r_i - Gauche}{V_{i-1} + 1}$
6. $\quad\quad 1 : \dfrac{T * V_{i-1} + r_i - Gauche}{V_{i-1} + 1/3}$
7. $\quad\quad 2 : i = 1 \,\mathbf{?}\, \infty \,\mathbf{:}\, T + \dfrac{r_i - Gauche}{V_{i-1}}$
8. $\quad\quad 3 : \infty$
9. $\quad$ **si** $Ch < TempsCh$ **alors**
10. $\quad\quad TempsCh \leftarrow Ch$
11. $\quad\quad iRobotCh \leftarrow i$
12. $\quad Droit \leftarrow \text{PortéeRobot}(i, Gauche, T)$
13. $\quad Gauche \leftarrow Droit$
14. **si** $S_{n+1} = 0$ **et** $S_n > 0$ **alors**
15. $\quad Ch \leftarrow T + \dfrac{L - Gauche}{V_n}$
16. $\quad$ **si** $Ch < TempsCh$ **alors**
17. $\quad\quad TempsCh \leftarrow Ch$
18. $\quad\quad iRobotCh \leftarrow n + 1$
19. **retourner** $(TempsCh, iRobotCh)$

Lemme 8 *La fonction* ProchainChangement(T) *trouve correctement le prochain changement d'état de robot qui arrive après un temps donnée* T.

Démonstration : Dans les lignes 4-8 de la fonction on calcule le temps de changement de l'état du robot R_i (pour $i = 1, \ldots, n$). Ce temps est pour l'instant hypothétique, parce qu'il est possible qu'avant ce temps là un autre robot R_j (pour $j < i$) puisse changer son état ce qui rendra le calcul de temps du changement de l'état de R_i invalide. Cependant le plus petit temps de changement sera calculé correctement, parce qu'aucun événement, qui pourra le rendre invalide n'aura lieu.

Prouvons d'abord, par induction sur i, l'assertion que dans chaque itération de la boucle pour des lignes 3-13, variable *Gauche* contient en ligne 4 la valeur de $d_{i-1}(T)$ - l'extrémité droite du segment exploré par robot R_{i-1}.

Comme le robot virtuel R_0 ne bouge pas et $Gauche = 0$ dans ligne 2, l'assertion est vraie en première itération, c'est-à-dire pour $i = 1$.

Supposons que l'assertion est vraie en itération $i - 1$, c'est-à-dire $Gauche = d_{i-1}(T)$. Par Lemme 4, en lignes 12-13, la nouvelle valeur de *Gauche* sera $d_i(T)$, ce qui conclue la preuve de l'assertion.

Prouvons maintenant que le calcul du temps du prochain changement d'état en lignes 4-8 est correct (pour chaque cas de l'état actuel dans lequel R_i se trouve).

1. **Cas** $S_i = 0$: Dans ce cas le robot R_i marche à gauche à la vitesse 1 tandis que le point d_{i-1} se déplace à droite à la vitesse V_{i-1}. L'état du robot change en temps Ch quand R_i arrive au point $d_{i-1}(Ch) = Gauche + (Ch - T)V_{i-1}$. (voir Figure 5.3).

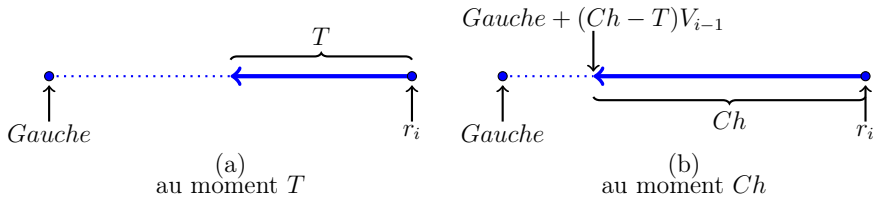


FIGURE 5.3 – Robot change l'état de 0 à 1

Nous avons

$$r_i - Gauche - (Ch - T)V_{i-1} = Ch$$

donc

$$Ch = \frac{r_i - Gauche + TV_{i-1}}{V_{i-1} + 1}$$

2. **Cas** $S_i = 1$: Le robot change son état au moment où l'extrémité gauche $g_i(Ch)$ de segment exploré est à la même distance de r_i que son extrémité droite $d_i(Ch)$. Dans ce cas : $d_i(Ch) - r_i = r_i - g_i(Ch)$.

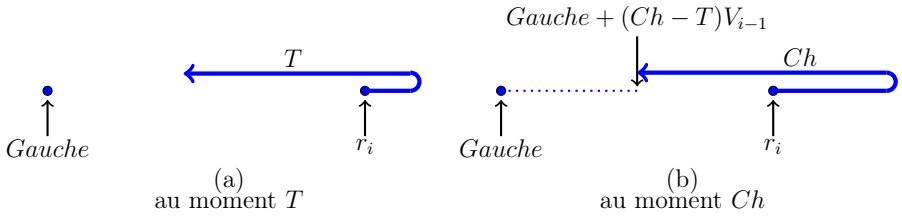


FIGURE 5.4 – Robot change état de 1 à 2

Comme

$$g_i(Ch) = d_{i-1}(Ch) = Gauche + (Ch - T)V_{i-1}$$

nous avons

$$r_i - Gauche - (Ch - T)V_{i-1} = \frac{Ch}{3}$$

donc

$$Ch = \frac{r_i + TV_{i-1} - Gauche}{V_{i-1} + \frac{1}{3}}$$

3. **Cas** $S_i = 2$: Dans ce cas le robot R_i commence à marcher uniquement à droite (voir Figure 5.5). Ceci arrive quand $d_{i-1}(Ch) = g_i(Ch) = r_i$.

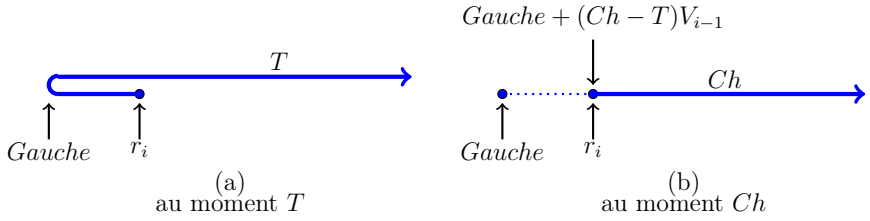


FIGURE 5.5 – Robot change l'état de 2 à 3

Comme

$$d_{i-1}(Ch) = Gauche + (Ch - T)V_{i-1} = r_i$$

alors

$$Ch = T + \frac{r_i - Gauche}{V_{i-1}}$$

Observons que le robot R_1 n'arrive jamais à l'état 3.

4. **Cas** $S_i = 3$: Cet état n'est jamais changé.

Alors les temps des changements d'état en lignes 4-8 sont calculés correctement.

Le lignes 9-11 répétées à chaque itération permettent de garder le plus petit temps de changement pour les robots $R_1, \ldots, R_n$ et l'indice $iRobotCh$ du robot qui subit ce changement.

Le robot virtuel R_{n+1} est immobile au point L. Il change son état de 0 à 1 lorsque le robot R_n le rencontre. Ceci arrive au moment $T + \dfrac{L - Gauche}{V_n}$, calculé dans les lignes 14-15. Si ce temps s'avère plus petit, que le temps minimal pour les robots $R_1, \ldots, R_n$ (lignes 16-18) il est retourné par la fonction. □

5.3 Présentation de l'algorithme

Algorithme TE.

On cherche un temps minimal T suffisant pour explorer l'intervalle $[0, L]$ par les robots $R_1, R_2, ..., R_n$

1. $T \leftarrow 0$
2. $r_0 \leftarrow 0$ # position du robot virtuel R_0 avant le premier robot
3. $r_{n+1} \leftarrow L$ # position du robot virtuel R_{n+1} après le dernier robot
4. $S_i \leftarrow 0$ pour $i = 1, \ldots, n+1$ # états de robots avec des valeurs initiales égales à 0. Les valeurs suivantes sont possibles : 0 - seulement vers la gauche, 1 - vers la droite ensuite à gauche, 2 - vers la gauche et ensuite à droite, 3 - seulement vers la droite

5. $V_0 \leftarrow 0$ # la vitesse du robot virtuel R_0
6. **tant que** *ExisteRobotEnÉtatZero(S)* **faire**
   # Calculer les vitesses d'avancement d_i, pour $i = 1, \ldots, n$
7. $CalculerVitesses()$
8. # Calculer le prochain moment de changement d'état par un robot
9. $(T, i) =$*ProchainChangement(T)*
10. $S_i \leftarrow S_i + 1$
11. La réponse est T

Bien que notre algorithme calcule le temps optimal d'exploration T, le plan de mouvement des robots est implicitement évident. Le robot R_1, à distance r_1 du début de segment détermine le plus long segment $[0, d_1(T)]$ qu'il peut explorer. Ensuite, le robot R_2 explore le plus long segment $[g_2(T), d_2(T)]$ où $g_2(T) = d_1(T)$.

On continue, par induction, de la même façon pour chaque robot subséquent. On peu conclure par le Corollaire suivant :

Corollaire 4 *Le mouvement des robots $R_1, \ldots, R_i$, pour $i = 1, \ldots, n$ implicite à l'algorithme TE résulte en exploration du plus long segment possible $[0, L_i]$. La complexité de l'algorithme TE ne dépend pas de la valeur de L.*

Nous pouvons passer à la preuve du théorème central de ce mémoire.

Théorème 3 *L'algorithme TE trouve en temps $O(n^2)$ le plus petit temps nécessaire à l'ensemble de robots $R_1, \ldots, R_n$ pour explorer le segment $[0, L]$*

Démonstration : Prouvons d'abord l'exactitude de l'algorithme. L'algorithme initialise en lignes 1-5 les états des robots $R_1, \ldots, R_n$. À chaque itération, l'algorithme calcule correctement, par Lemme 8, le prochain changement d'état d'un robot. Les nouvelles vitesses d'évolution $d_i(t)$ sont calculées correctement (Lemme 6). Par Corollaire 3 entre chaque deux changements d'états consécutifs, les vitesses ne changent pas.

Tant qu'il existe un robot R_i en état 0, il y a un espace non exploré, parce que $d_{i-1}(T) < g_i(T)$. L'algorithme se termine quand

$$d_i(T) = g_{i+1}(T), \text{pour } i = 0, \ldots, n$$

alors le segment est entièrement exploré.

Par Corollaire 4, il n'existe pas un $\epsilon > 0$, tel que le segment peut être exploré en temps $T - \epsilon$.

Pour prouver la complexité de l'algorithme observons que chaque robot peut se trouver en un de quatre états possibles, alors la boucle de la ligne 6 s'exécute $O(n)$ fois. La complexité des fonctions $CalculerVitesses$ et $ProchainChangement$ est dominée par des boucles linéaires (lignes 1 et 3, respectivement) dont les corps s'exécutent en temps $O(1)$. Conséquemment, la complexité de l'algorithme est en $O(n^2)$. □

Chapitre 6

Programme pour le système d'exploitation iOS

Pour visualiser le fonctionnement des algorithmes présentés dans le mémoire, nous avons créé une application. En utilisant un écran tactile, un utilisateur peut ajuster la position et la quantité des robots, contrôler le temps disponible par robot et voir la trajectoire optimale pour chaque robot en temps réel. Il peut aussi voir le fonctionnement de deux algorithmes TA (temps approximatif) et de TE (temps exact) suivant chaque itération de boucle de chaque algorithme.

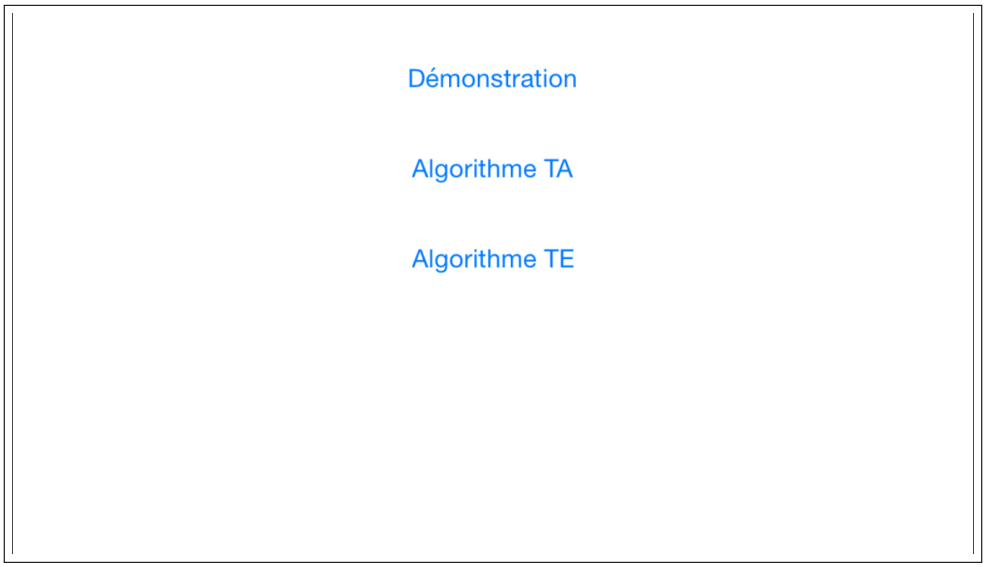
Par défaut, l'application utilise cinq robots dans des positions prédéfinies. Cependant, l'utilisateur peut ajouter ou supprimer d'autres robots, ainsi que modifier leurs positions initiales.

L'application mobile a été créée en langage Objective-C en utilisant la récente version 6.3 de logiciel Xcode sous le système d'exploitation OS X Yosemite version 10.10.3. L'application est disponible pour tout iPhone et iPad avec un système d'exploitation iOS à partir de la version 6.0. L'exigence d'espace disponible sur l'appareil est seulement de 500 Ko. Une fois l'application installée la connexion internet n'est pas nécessaire.

6.1 Description de fonctionnement

L'application se compose de quatre écrans : écran *Menu* principal et trois écrans de visualisation (*Démonstration*, *Algorithme TA* et *Algorithme TE*). La navigation entre les écrans passe toujours par l'écran *Menu*.

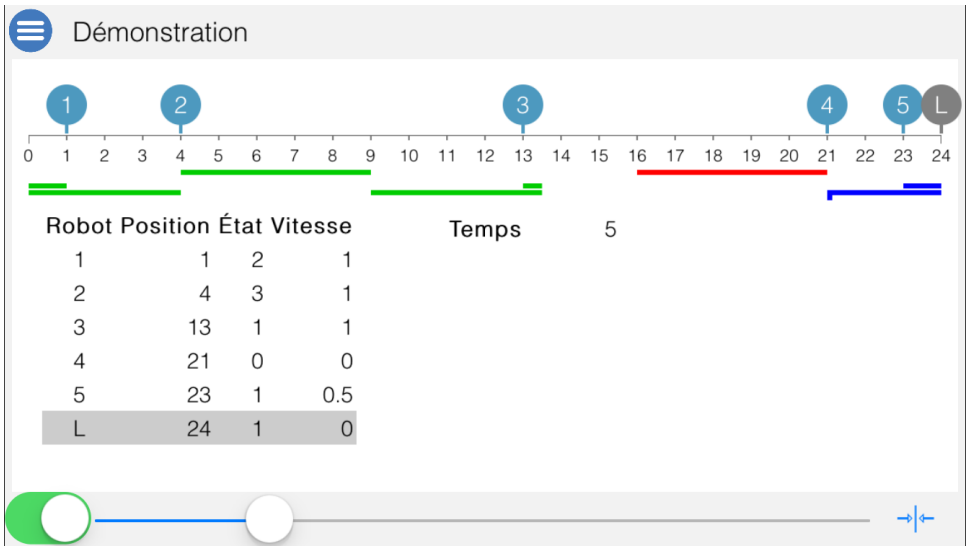
Écran Menu



L'écran menu contient le menu qui permet de sélectionner l'un de trois modes de fonctionnement :

1. Démonstration - visualisation en temps réel des trajectoires pour un temps choisi
2. Algorithme TA - visualisation de fonctionnement d'algorithme approximatif, l'utilisateur contrôle quelle itération de boucle (voir Algorithme TA lignes 2-8) devrait être présentée
3. Algorithme TE - visualisation de fonctionnement d'algorithme exact, l'utilisateur contrôle quelle itération de boucle (voir Algorithme TE lignes 6-11) devrait être présentée

Écran Démonstration



L'écran est composé de trois sections :

1. Haut de l'écran

   Le bouton dans le coin gauche - retour au menu principal

   Le retour au menu principal sauvegarde des positions de robots pour permettre de comparer les algorithmes avec les mêmes données.

2. Milieu de l'écran

   Un segment avec indication numérique d'échelle est présenté sur la largeur de l'écran. Sur le segment, les robots sont présentés graphiquement en couleur bleu avec leurs numéros. Un robot peut être :

   — ajouté - en tapant sur le segment dans la position désirée

   — déplacé - en touchant un robot et le déplaçant sur la longueur du segment

   — enlevé - en touchant le robot et le déplaçant vers le haut.

   En dessous de chaque robot sa trajectoire optimale est présentée. Trois couleurs sont possibles :

   — rouge - le robot n'a pas réussi à explorer son sous-segment

   — vert - le robot a réussi à explorer son sous-segment

   — bleu - le robot avait trop de temps disponible est pourrait explorer son sous-segment en moins de temps

   Dans le tableau, la liste des robots avec leurs positions, leurs états actuels et les vitesses de déplacement de l'extrémité droite de leurs segments d'opérations.

   Le temps actuel utilisé est affiché à côté du tableau.

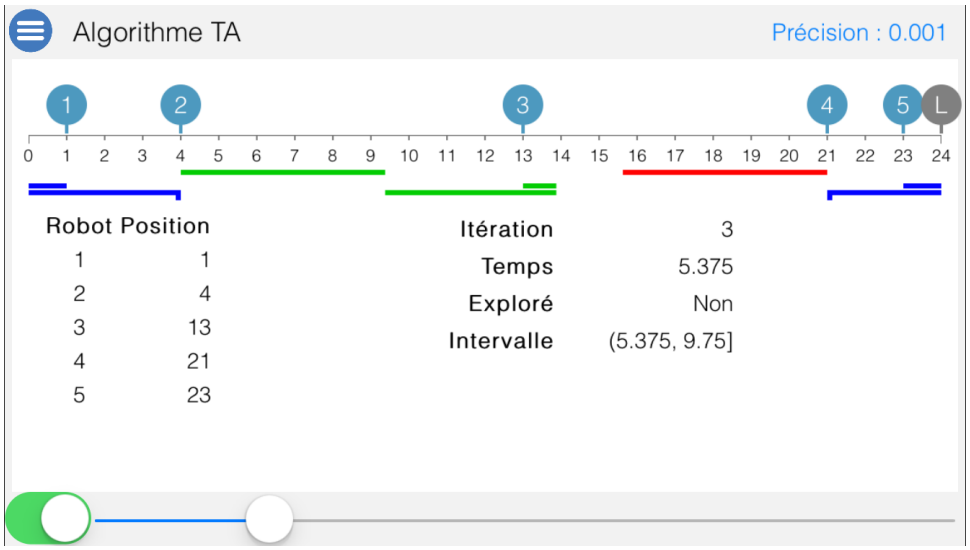
3. Bas de l'écran

   Le baladeur permet d'ajuster le temps disponible pour chaque robot. On touche la boule et on glisse sur une ligne. L'échelle du baladeur est ajustée selon le nombre de robots pour permettre l'exploration complète du segment dans le pire des cas.

   Le bouton marche-arrêt en position *marche* limite le baladeur et les nouvelles positions des robots aux valeurs entières seulement.

   Le bouton dans le coin droit permet de changer la précision de sélection de temps. Par défaut le choix de temps est possible dans toute la largeur de l'échelle. Avec la précision augmentée, la sélection est précise à ± 0.5 de la valeur du temps actuel.

Écran Algorithme TA



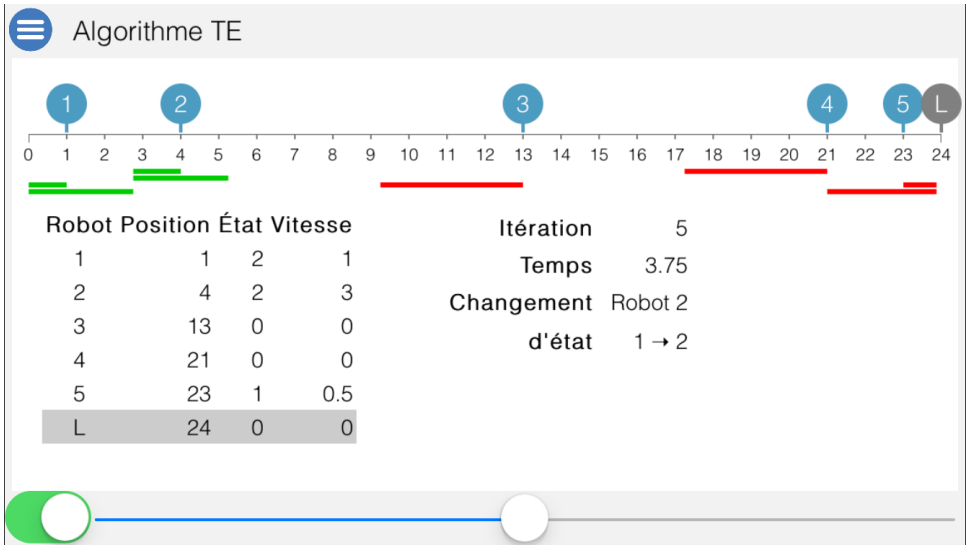
Toutes les fonctionnalités disponibles à l'écran *Démonstration* dans les sections du haut et du milieu sont toujours accessibles. En plus, dans le coin droit il y a un bouton qui permet de changer la précision de calculs. Par défaut c'est 0.001. Pour augmenter la précision il faut glisser le bouton vers la droite, pour diminuer la précision on glisse le bouton vers la gauche. Le réglage possible est entre $[0.1, 0.00000001]$, c'est-à-dire $[10^{-1}, 10^{-8}]$.

Le tableau de gauche est identique au tableau de l'écran *Démonstration*. Le tableau de droite montre le numéro d'itération, la quantité du temps disponible par itération, l'information si le segment a été exploré et l'intervalle dans lequel se trouve la solution.

En bas de l'écran, le baladeur est utilisé pour naviguer entre les itérations de la boucle de l'algorithme TA.

Le résultat est calculé jusqu'à la précision choisie.

Écran Algorithme TE



Toutes les fonctionnalités disponibles à l'écran *Démonstration* dans les sections du haut et du milieu sont toujours accessibles.

Le tableau de gauche est identique au tableau de l'écran *Démonstration*. Le tableau de droite montre le numéro d'itération, la quantité du temps disponible par itération et le changement de l'état par robot.

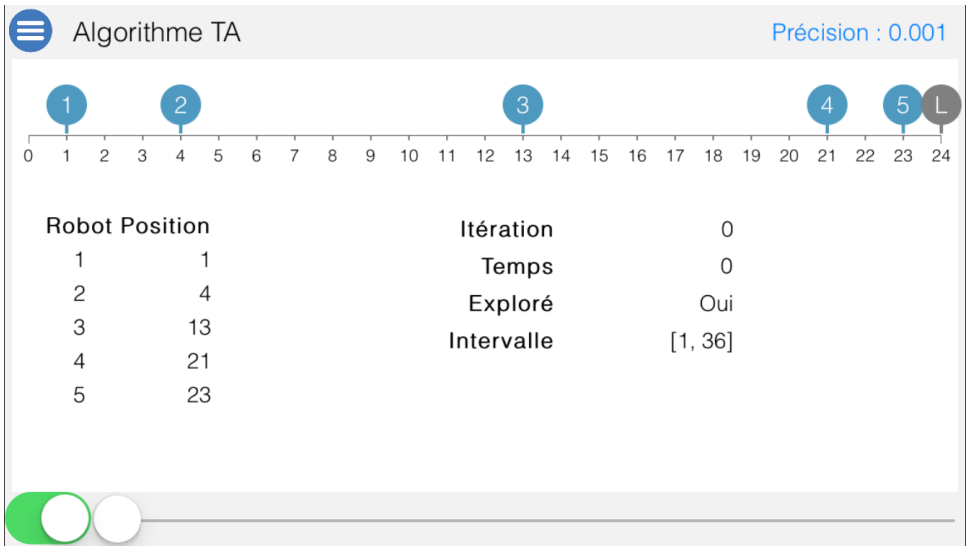
En bas de l'écran, le baladeur est utilisé pour naviguer entre les itérations de la boucle de l'algorithme TE.

Remarque :

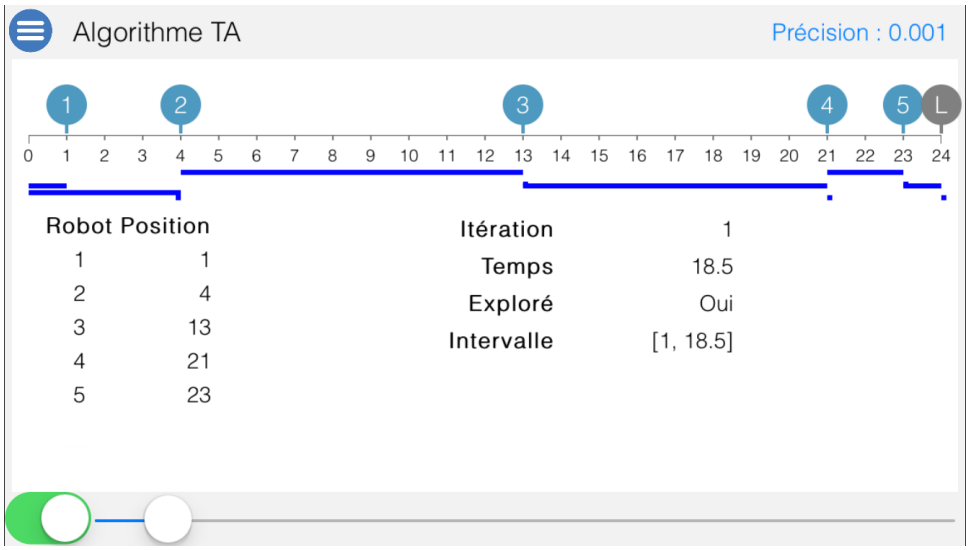
Il est possible de retourner aux valeurs par défaut en sélectionnant la quatrième option sur l'écran principal. L'option est disponible seulement quand il y a des données en mémoire.

6.2 Exemple d'exécution de l'algorithme TA

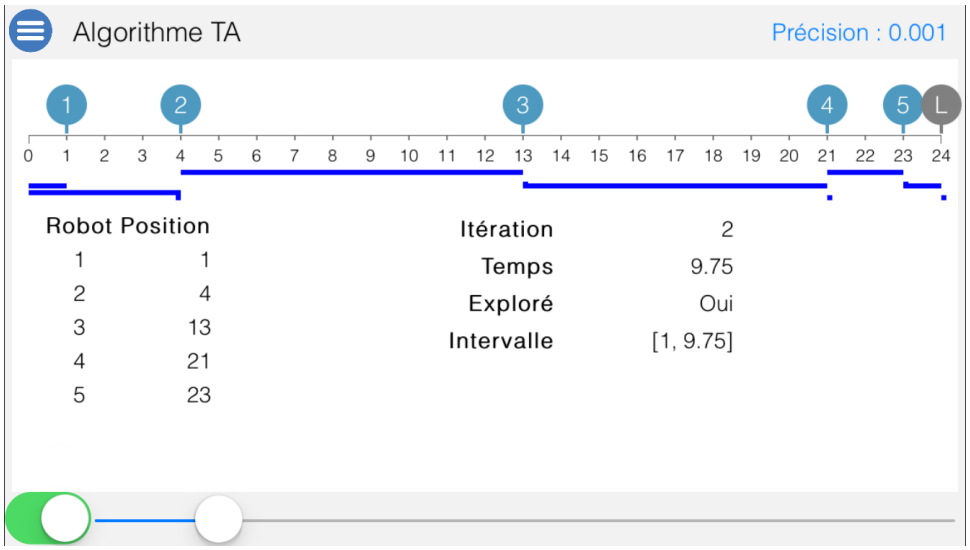
Sur le premier écran, on voit les robots dans des positions initiales. La solution se trouve dans l'intervalle $[1, 36]$ (voir ligne 1 d'algorithme TA).



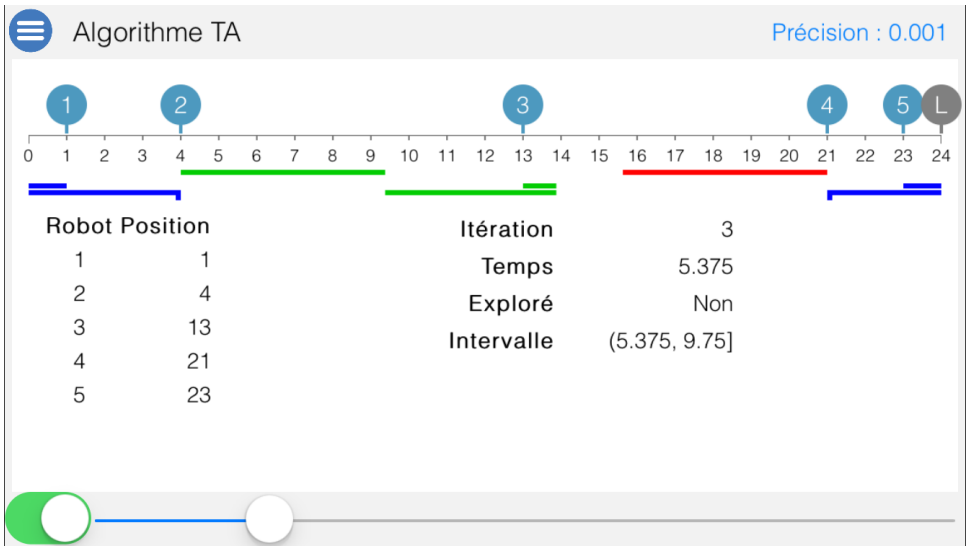
L'écran suivant montre la première itération d'algorithme. L'exploration est possible pour le temps 18.5 et l'intervalle est réduit.



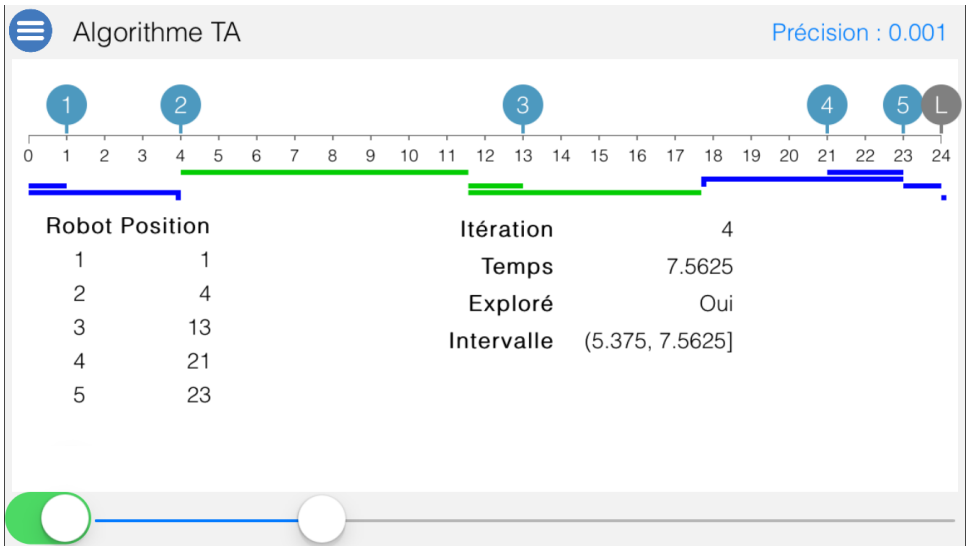
Dans la deuxième itération, l'exploration est toujours possible. L'intervalle est ajusté selon le résultat.

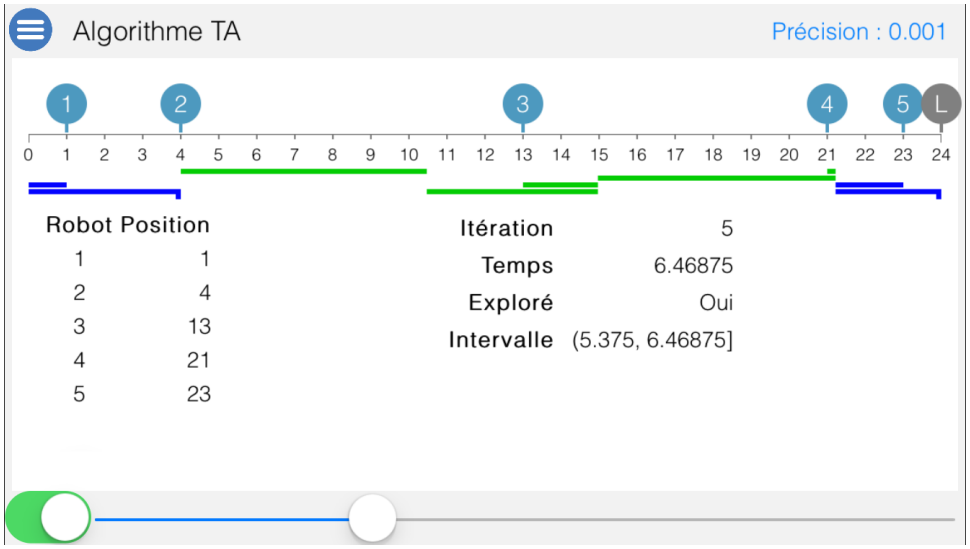


L'écran suivant montre la troisième itération. L'exploration n'est pas possible pour le temps 5.375. Cette valeur devient la nouvelle valeur minimale de l'intervalle.

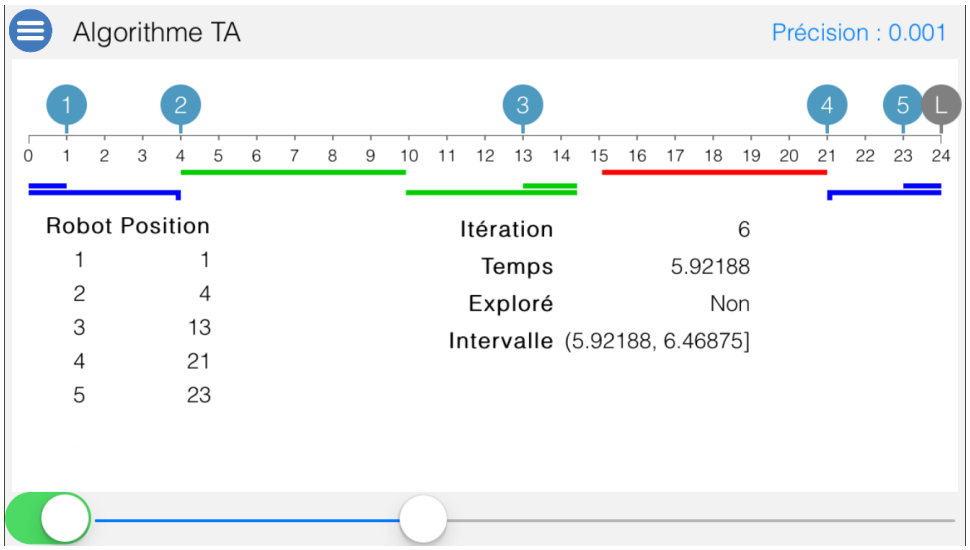


Dans les deux prochaines itérations, les écrans montrent que l'exploration est possible.

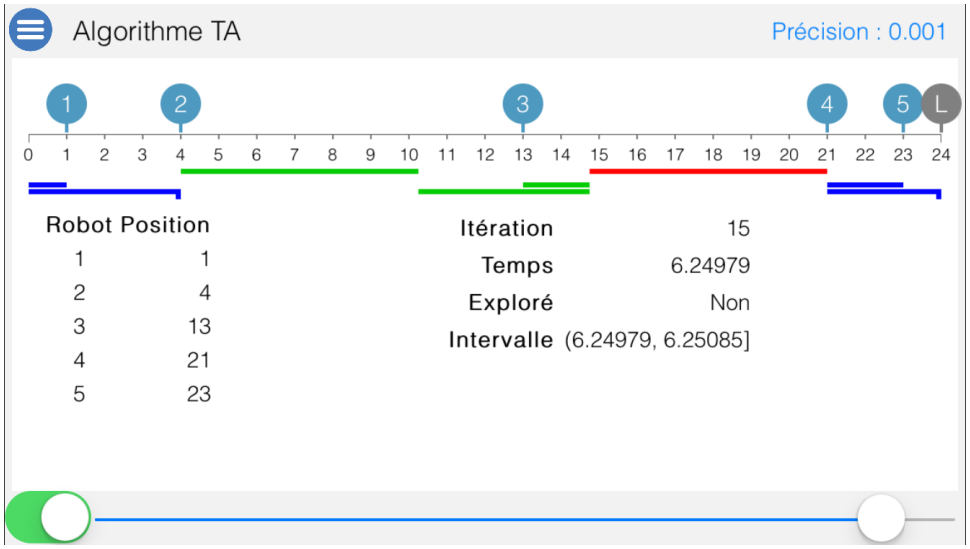




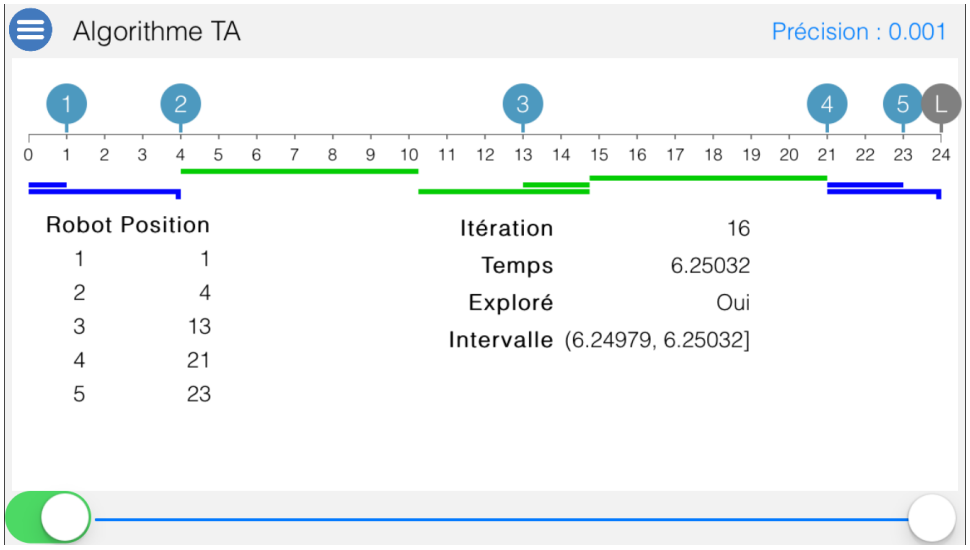
Dans l'itération numéro 6 l'exploration est de nouveau impossible.



On continue de réduire l'intervalle pour trouver une solution. Après un certain nombre d'itérations, on atteint la précision requise. Dans notre exemple, dans l'itération numéro 15 l'exploration n'est pas possible.

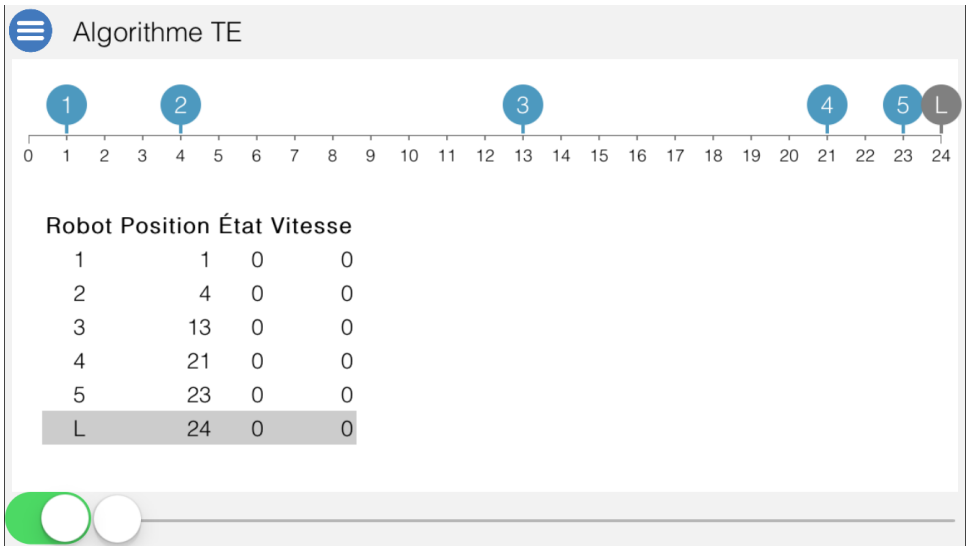


C'est la dernière itération. On arrive à notre précision requise. L'exploration est possible.

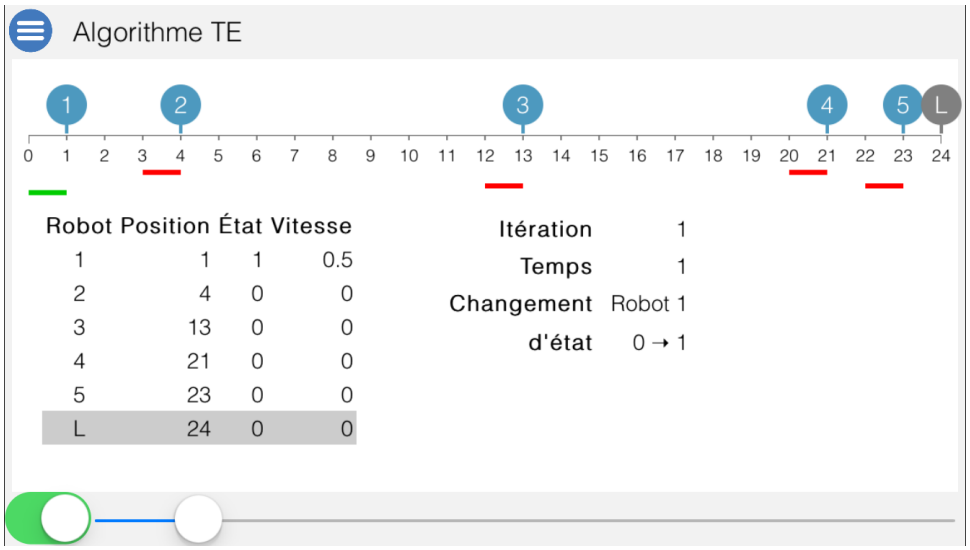


6.3 Exemple d'exécution de l'algorithme TE

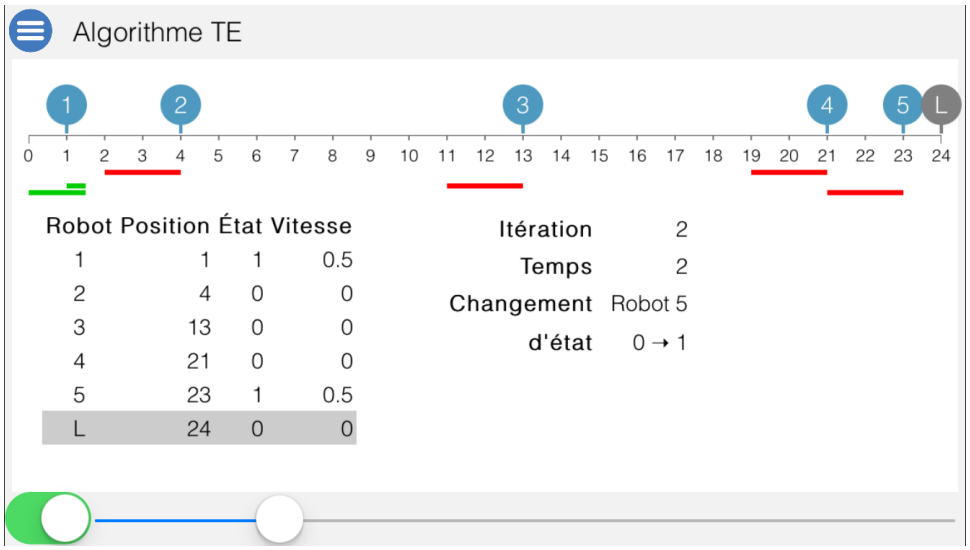
Sur le premier écran, on voit les robots dans les positions initiales.



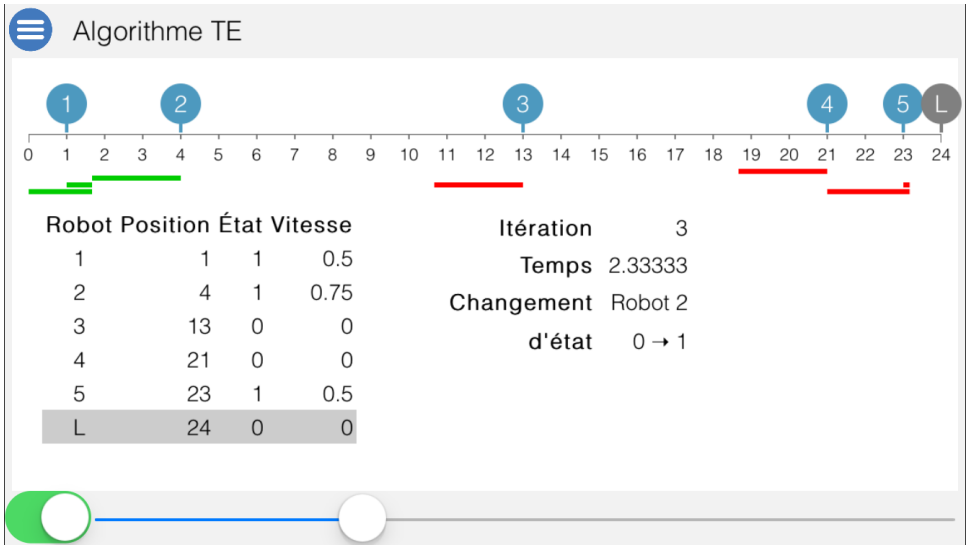
L'écran suivant montre le premier changement d'état. Le robot R_1 a changé l'état de 0 à 1.



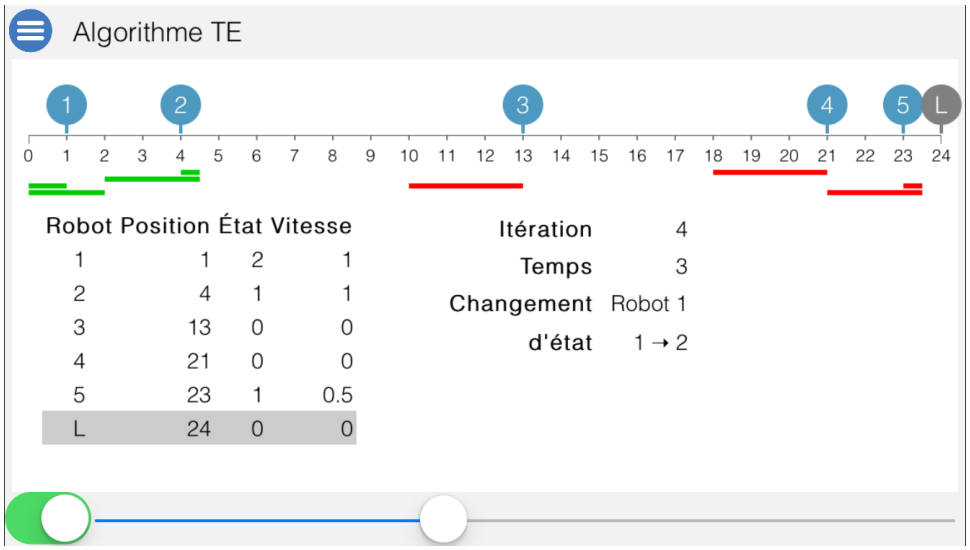
On voit sur l'image en dessous que dans la deuxième itération le robot R_5 a changé l'état de 0 à 1.



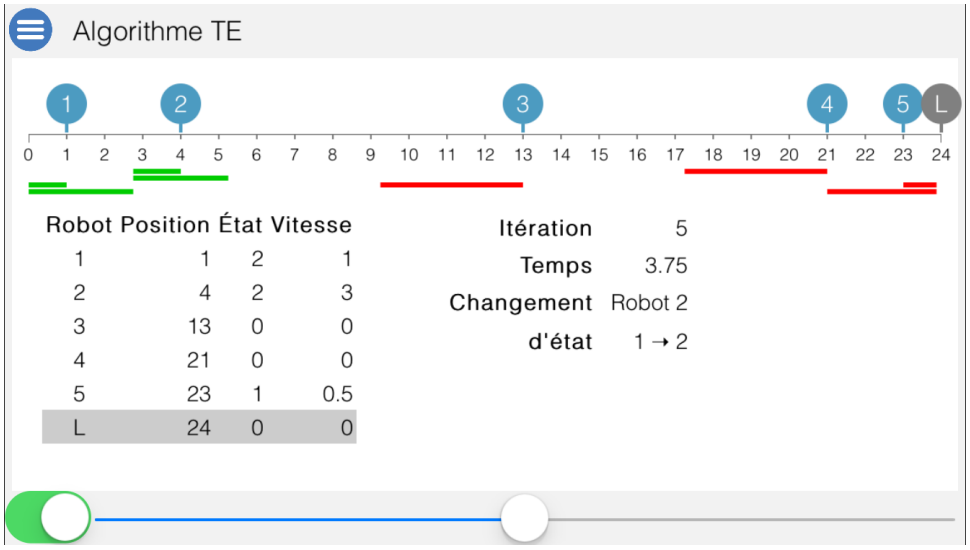
Le robot R_2 a changé son état de 0 à 1.



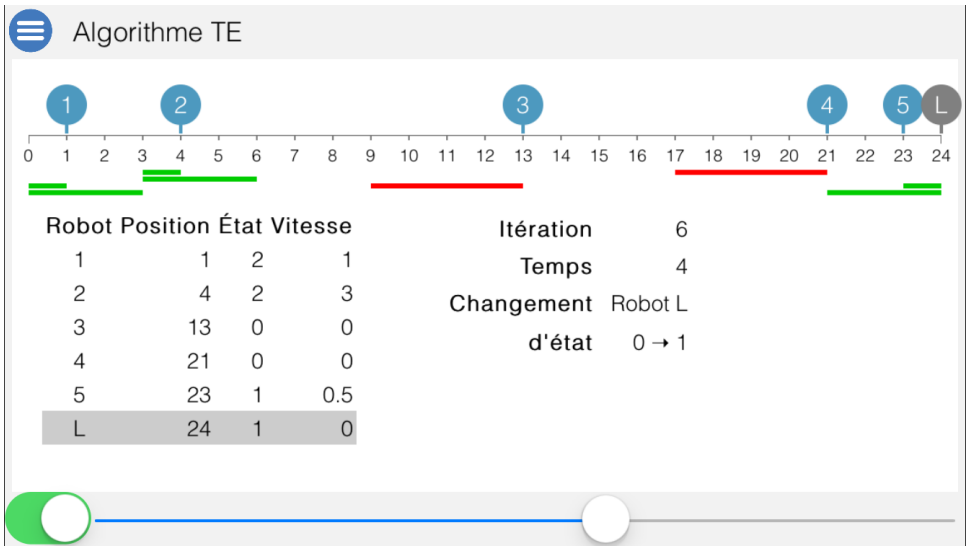
Le robot R_1 a changé son état de 1 à 2.



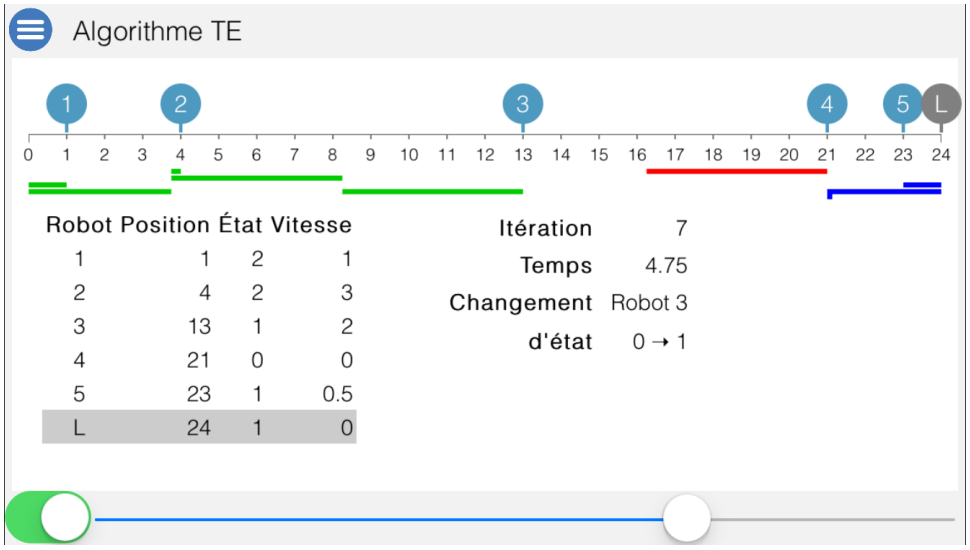
Le robot R_2 a changé son état de 1 à 2.



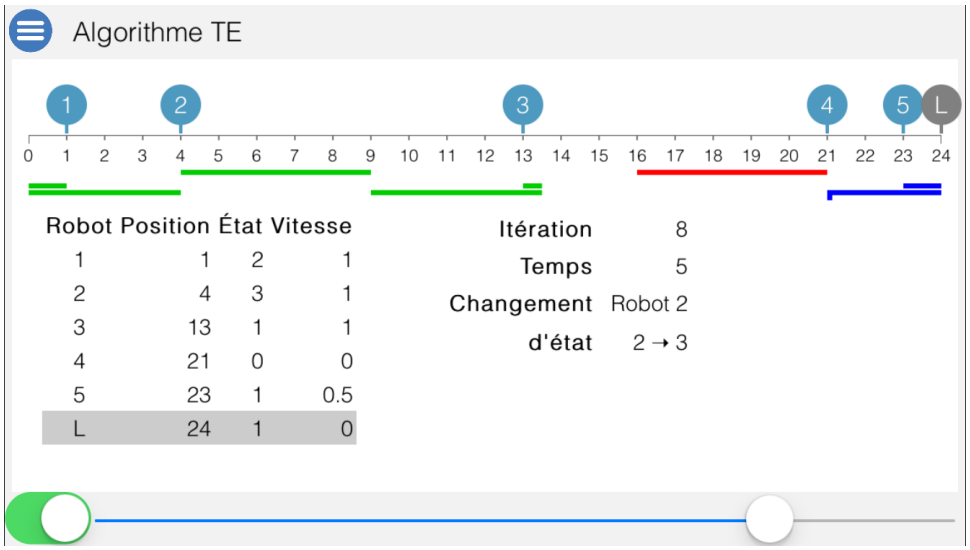
Le robot R_5 a atteint l'extrémité droite du segment. Donc le robot virtuel R_6 a changé son état de 0 à 1.



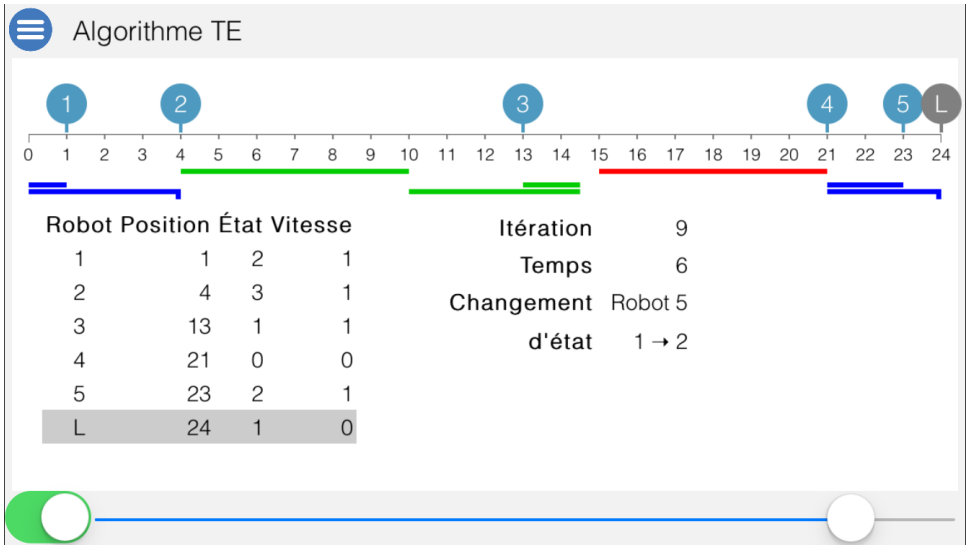
Le robot R_2 a changé son état de 2 à 3.



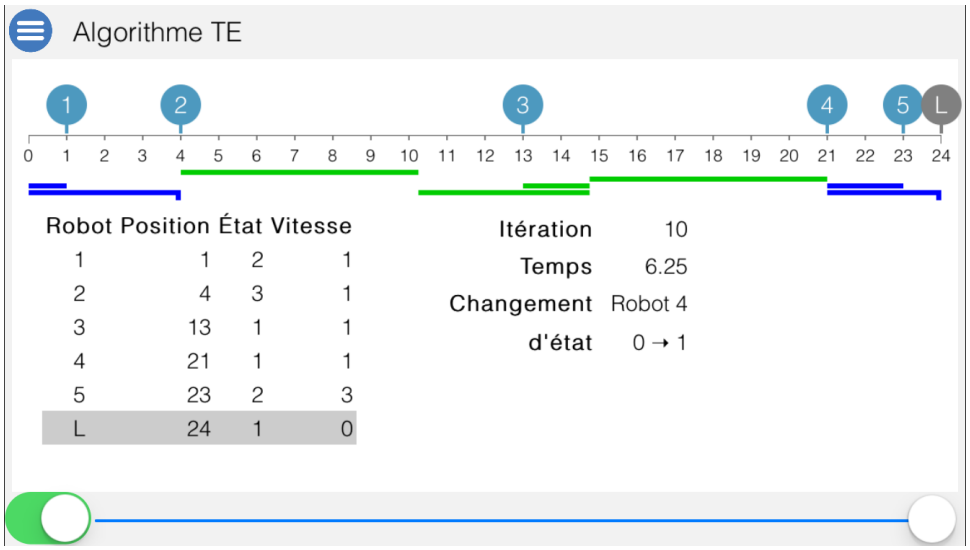
Le robot R_2 a changé son état de 0 à 1.



Le robot R_5 a changé son état de 1 à 2.



Le robot R_4 a changé son état de 0 à 1. Le segment a été entièrement exploré. Fin d'algorithme.



Chapitre 7

Conclusion

Dans le présent mémoire, nous avons présenté un problème qui consiste à traverser un segment de droite par n robots. Nous avons proposé un algorithme, qui calcule le temps minimal nécessaire pour effectuer un parcours d'exploration.

Nous avons prouvé l'exactitude de l'algorithme et nous avons estimé son complexité. Également nous avons créé une application pour iOS illustrant l'algorithme.

Il y a encore plusieurs problèmes ouverts reliés à ce mémoire. Par exemple :

1. Exploration de l'anneau (plutôt que du segment)
2. Exploration par des robots ayant des vitesses différentes
3. Les robots à deux vitesses (une vitesse inférieure pour l'exploration et une vitesse supérieure pour le déplacement)

Il semble que le problème 1 nécessiterait une complexité supérieure à celle de

l'exploration du segment. Le défi du problème 2 réside dans le fait que l'ordre des robots pourrait changer durant l'exploration (Lemme 2 n'est plus valide). Le problème 3 a été résolu dans [17] pour un cas des robots commençant tous à l'extrémité du segment. Considérer les positions initiales quelconques n'était pas étudié pour ce problème.

Annexe A

Implémentation iOS

```
//
//  MenuViewController.h
//

#import <UIKit/UIKit.h>

@interface MenuViewController : UIViewController

- (void)saveRobots:(NSArray *)Robots intOnly:(BOOL)intOnly;

@end

//
//  MenuViewController.m
//

#import "MenuViewController.h"
#import "DemoViewController.h"
#import "AlgoTAViewController.h"
#import "AlgoTEViewController.h"

@interface MenuViewController ()

@property (nonatomic, strong) NSArray *savedRobots;
@property (nonatomic) BOOL intOnly;

@property (nonatomic) UIButton *Demo;
@property (nonatomic) UIButton *AlgoTA;
@property (nonatomic) UIButton *AlgoTE;
@property (nonatomic) UIButton *Reset;

@end

@implementation MenuViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    self.Demo = [self createButton:@"Démonstration" action:@selector(btnDemoClicked:)];
    self.AlgoTA = [self createButton:@"Algorithme_TA" action:@selector(btnAlgoTAClicked:)];
    self.AlgoTE = [self createButton:@"Algorithme_TE" action:@selector(btnAlgoTEClicked:)];
    self.Reset = [self createButton:@"Positions_par_défaut" action:@selector(
    btnResetClicked:)];
}

- (UIButton *)createButton:(NSString *)title action:(SEL)action
{
    UIButton *btn = [UIButton buttonWithType:UIButtonTypeSystem];
    [btn setTitle:title forState:UIControlStateNormal];
    [btn sizeToFit];
    [btn addTarget:self action:action forControlEvents:UIControlEventTouchUpInside];
    [self.view addSubview:btn];
    return btn;
}
```

```
- (void)viewDidLayoutSubviews
{
    [self placeButton:self.Demo k:1];
    [self placeButton:self.AlgoTA k:2];
    [self placeButton:self.AlgoTE k:3];
    [self placeButton:self.Reset k:4];

    self.Reset.hidden = !self.savedRobots;
}

- (void)placeButton:(UIButton *)btn k:(int)k
{
    CGSize sizeView = self.view.frame.size;
    double h_3 = sizeView.height / 6;

    CGRect frame = btn.frame;
    frame.origin.x = (sizeView.width - frame.size.width) / 2;
    frame.origin.y = k * h_3 - frame.size.height;
    btn.frame = frame;
}

- (BOOL)prefersStatusBarHidden {
    return YES;
}

- (IBAction)btnDemoClicked:(id)sender
{
    [self callVC:[[DemoViewController alloc] init]];
}

- (IBAction)btnAlgoTAClicked:(id)sender
{
    [self callVC:[[AlgoTAViewController alloc] init]];
}

- (IBAction)btnAlgoTEClicked:(id)sender
{
    [self callVC:[[AlgoTEViewController alloc] init]];
}

- (void)callVC:(id)vc
{
    [vc assignRobots:self.savedRobots intOnly:self.intOnly];
    [vc setMenu:self];
    [self presentViewController:vc animated:YES completion:nil];
}

- (IBAction)btnResetClicked:(id)sender
{
    self.savedRobots = nil;
    self.Reset.hidden = !self.savedRobots;
}

- (void)saveRobots:(NSArray *)Robots intOnly:(BOOL)intOnly
{
    self.savedRobots = Robots;
    self.intOnly = intOnly;
}

@end

//
//  DemoViewController.h
//

#import <UIKit/UIKit.h>
#import "MenuViewController.h"

@interface DemoViewController : UIViewController {
    NSArray *Robots;
    double L;
}

#define VIEW_MARGIN_X 4
```

```
#define VIEW_MARGIN_Y 32
#define SEGMENT_VIEW_MARGIN 10
#define SEGMENT_VIEW_Y 45
#define SEGMENT_L 24
#define START_TEMPS 5

#define PRECISION 0.0001
#define COLOR_TYPE_OK 0
#define COLOR_TYPE_TOO_MUCH 1
#define COLOR_TYPE_NOT_ENOUGH -1
#define ROBOT_Y_SENSITIVITY 50
#define ROBOT_RADIUS_SENSITIVITY 20
#define STATE_0_LEFT_ONLY
#define STATE_1_RIGHT_LEFT
#define STATE_2_LEFT_RIGHT
#define STATE_3_RIGHT_ONLY

@property (nonatomic) MenuViewController *menu;
@property (nonatomic) UILabel *lblTitle;
@property (nonatomic) UISlider *sliderTemps;
@property (nonatomic) NSArray *trace;
@property (nonatomic) double precision;

- (void)algorithmModeOn;
- (void)assignRobots:(NSArray *)newRobots intOnly:(BOOL)intOnly;
- (void)showOptimalFor:(double)temps;
- (double)porteeRobot:(int)i gauche:(double)G temps:(double)T;

@end

//
//  DemoViewController.m
//

#import "DemoViewController.h"
#import "MenuViewController.h"
#import "SegmentView.h"
#import "RobotView.h"
#import "Robot.h"
#import "TrajectoryView.h"

@interface DemoViewController () <UIGestureRecognizerDelegate> {
    CGFloat segmentViewMax;
    CGFloat segmentViewUnity;
    BOOL algorithmeMode;
}

@property (nonatomic) UIButton *btnMenu;
@property (nonatomic) UIView *viewMain;
@property (nonatomic) UIWebView *tableRobotInfos;
@property (nonatomic) UISwitch *switchIntOnly;
@property (nonatomic) UIButton *btnZoom;
@property (nonatomic) BOOL zoom;
@property (nonatomic) Robot *movingRobot;
@property (nonatomic) double tempsMax;

@end

@implementation DemoViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    self.view.backgroundColor = [UIColor colorWithRed:0.95 green:0.95 blue:0.95 alpha:1];
    self.lblTitle.text = @"Démonstration";

    [self placeMainView];
    [self showSegment:SEGMENT_L atViewY:SEGMENT_VIEW_Y withViewMargin:SEGMENT_VIEW_MARGIN];

    self.precision = PRECISION;
    self.tempsMax = START_TEMPS;
    self.sliderTemps.value = self.tempsMax;
    [self showTemps];

    self.movingRobot = nil;
```

```
    UITapGestureRecognizer *tapGestureRecognizer = [[UITapGestureRecognizer alloc] init];
    [tapGestureRecognizer addTarget:self action:@selector(handleTap:)];
    tapGestureRecognizer.delegate = self;
    [self.viewMain addGestureRecognizer:tapGestureRecognizer];

    UIPanGestureRecognizer *panGestureRecognizer = [[UIPanGestureRecognizer alloc] init];
    [panGestureRecognizer addTarget:self action:@selector(handlePan:)];
    panGestureRecognizer.delegate = self;
    [self.viewMain addGestureRecognizer:panGestureRecognizer];
}

- (void)viewDidLayoutSubviews
{
    CGRect frame;

    self.btnMenu.frame = CGRectMake(0, 0, 30, 30);
    self.lblTitle.frame = CGRectMake(40, 5, self.view.frame.size.width - 40, 22);

    CGSize sizeMain = self.viewMain.frame.size;
    CGPoint posMain = self.viewMain.frame.origin;

    double m = SEGMENT_VIEW_MARGIN;
    self.tableRobotInfos.frame = CGRectMake(m, m + 70, sizeMain.width - m * 2, sizeMain.
    height - m - 80);

    frame = self.switchIntOnly.frame;
    frame.origin.x = 0;
    frame.origin.y = posMain.y + sizeMain.height;
    self.switchIntOnly.frame = frame;

    frame = self.btnZoom.frame;
    frame.origin.x = posMain.x + sizeMain.width - frame.size.width;
    frame.origin.y = posMain.y + sizeMain.height + 4;
    self.btnZoom.frame = frame;

    frame = self.sliderTemps.frame;
    frame.origin.x = self.switchIntOnly.frame.size.width;
    frame.origin.y = posMain.y + sizeMain.height;
    frame.size.width = sizeMain.width - self.switchIntOnly.frame.size.width + VIEW_MARGIN_X
    ;
    if (!algorithmeMode) {
        frame.size.width -= self.btnZoom.frame.size.width;
    }
    self.sliderTemps.frame = frame;
}

- (void)placeMainView
{
    CGFloat h = self.view.frame.size.height;
    CGFloat w = self.view.frame.size.width;

    self.viewMain.frame = CGRectMake(VIEW_MARGIN_X, VIEW_MARGIN_Y, w - 2 * VIEW_MARGIN_X, h
     - 2 * VIEW_MARGIN_Y);
}

- (void)algorithmModeOn
{
    algorithmeMode = YES;
    self.btnZoom.hidden = YES;
}

- (BOOL)prefersStatusBarHidden {
  return YES;
}

- (UIButton *)btnMenu
{
    if (!_btnMenu) {
        _btnMenu = [UIButton buttonWithType:UIButtonTypeCustom];
        [_btnMenu setImage:[UIImage imageNamed:@"Menu-Icon-5"] forState:
        UIControlStateNormal];
        [_btnMenu addTarget:self action:@selector(btnMenuClick:) forControlEvents:
        UIControlEventTouchUpInside];
        [self.view addSubview:_btnMenu];
```

```
    }
    return _btnMenu;
}

- (UILabel *)lblTitle
{
    if (!_lblTitle) {
        _lblTitle = [[UILabel alloc] init];
        _lblTitle.frame = CGRectMake(0, 0, 77, 21);
        _lblTitle.font = [UIFont fontWithName:@"Helvetica Light" size:16];
        [self.view addSubview:_lblTitle];
    }
    return _lblTitle;
}

- (UIView *)viewMain
{
    if (!_viewMain) {
        _viewMain = [[UIView alloc] init];
        _viewMain.backgroundColor = [UIColor whiteColor];
        [self.view addSubview:_viewMain];
    }
    return _viewMain;
}

- (UIWebView *)tableRobotInfos
{
    if (!_tableRobotInfos) {
        _tableRobotInfos = [[UIWebView alloc] init];
        _tableRobotInfos.opaque = NO;
        _tableRobotInfos.backgroundColor = [UIColor clearColor];
        [self.viewMain addSubview:_tableRobotInfos];
    }
    return _tableRobotInfos;
}

- (UISwitch *)switchIntOnly
{
    if (!_switchIntOnly) {
        _switchIntOnly = [[UISwitch alloc] init];
        [_switchIntOnly addTarget:self action:@selector(switchIntOnlyChanged:)
        forControlEvents:UIControlEventValueChanged];
        [self.view addSubview:_switchIntOnly];
    }
    return _switchIntOnly;
}

- (UISlider *)sliderTemps
{
    if (!_sliderTemps) {
        _sliderTemps = [[UISlider alloc] init];
        _sliderTemps.maximumValue = 1.5 * L;
        [_sliderTemps addTarget:self action:@selector(sliderTempsChanged:) forControlEvents
        :UIControlEventValueChanged];
        [self.view addSubview:_sliderTemps];
    }
    return _sliderTemps;
}

- (UIButton *)btnZoom
{
    if (!_btnZoom) {
        _btnZoom = [UIButton buttonWithType:UIButtonTypeSystem];
        _btnZoom.frame = CGRectMake(0, 0, 50, 21);
        [_btnZoom setTitle:@">|<" forState:UIControlStateNormal];
        _btnZoom.titleLabel.font = [UIFont fontWithName:@"Helvetica Light" size:16];
        [_btnZoom addTarget:self action:@selector(btnZoomClicked:) forControlEvents:
        UIControlEventTouchUpInside];
        [self.view addSubview:_btnZoom];
    }
    return _btnZoom;
}
```

```
- (void)showSegment:(double)length atViewY:(CGFloat)y withViewMargin:(CGFloat)x
{
    L = length;
    double w = self.viewMain.frame.size.width;
    segmentViewMax = w - SEGMENT_VIEW_MARGIN;
    segmentViewUnity = (segmentViewMax - SEGMENT_VIEW_MARGIN) / L;

    SegmentView *segment = [[SegmentView alloc] init];
    segment.frame = CGRectMake(SEGMENT_VIEW_MARGIN, SEGMENT_VIEW_Y, segmentViewMax -
    SEGMENT_VIEW_MARGIN, 80);
    segment.maxVal = L;

    [segment drawSegmentAt:0 withMargin:0];
    [self.viewMain addSubview:segment];
}

- (void)assignRobots:(NSArray *)newRobots intOnly:(BOOL)intOnly
{
    L = SEGMENT_L;
    if (!newRobots) {
        NSMutableArray *a = [[NSMutableArray alloc] init];
        [a addObject:[Robot robotAt: 0.0 virtuel:YES]];
        [a addObject:[Robot robotAt: 1.0 virtuel: NO]];
        [a addObject:[Robot robotAt: 4.0 virtuel: NO]];
        [a addObject:[Robot robotAt:13.0 virtuel: NO]];
        [a addObject:[Robot robotAt:21.0 virtuel: NO]];
        [a addObject:[Robot robotAt:23.0 virtuel: NO]];
        [a addObject:[Robot robotAt:L virtuel:YES]];
        newRobots = a;
        intOnly = YES;
    }
    for (Robot *robot in newRobots) {
        [self addRobotFor:robot.x virtuel:robot.virtuel];
    }
    self.switchIntOnly.on = intOnly;
    [self redrawRobots];
    [[NSNotificationCenter defaultCenter] postNotificationName:@"recalc" object:self];
}

- (void)addRobotAt:(CGFloat)xPos
{
    [self addRobotFor:(xPos - SEGMENT_VIEW_MARGIN) / segmentViewUnity];
}

- (void)addRobotFor:(double)x
{
    [self addRobotFor:x virtuel:NO];
}

- (void)addRobotFor:(double)x virtuel:(BOOL)virtuel
{
    CGFloat w = self.viewMain.frame.size.width;

    RobotView *rv = [[RobotView alloc] init];
    rv.tag = 0;
    [self.viewMain addSubview:rv];

    TrajectoryView *tv = [[TrajectoryView alloc] init];
    tv.frame = CGRectMake(0, SEGMENT_VIEW_Y + 20, w, 20);
    [self.viewMain addSubview:tv];

    Robot *r = [[Robot alloc] init];
    [r assignRobotView:rv];
    [r assignRobotTrajectory:tv];

    r.x = x;
    r.xLocation = SEGMENT_VIEW_MARGIN + r.x * segmentViewUnity;
    r.virtuel = virtuel;

    NSMutableArray *a = [NSMutableArray arrayWithArray:Robots];
    [a addObject:r];
    Robots = a;
    [self adjustSliderTemps];
```

```
}

- (void)adjustSliderTemps
{
    int n = (int)Robots.count - 2;
    double maxT = (n > 1) ? L : 1.5 * L;
    if (self.sliderTemps.value > maxT) {
        self.tempsMax = maxT;
        self.sliderTemps.value = maxT;
        [self sliderTempsChanged:self.sliderTemps];
    }
    self.sliderTemps.minimumValue = 0;
    self.sliderTemps.maximumValue = maxT;
}

- (void)redrawRobots
{
    NSArray *sortedArray;
    sortedArray = [Robots sortedArrayUsingComparator:^NSComparisonResult(id a, id b) {
        double first = [(Robot *)a x];
        double second = [(Robot *)b x];
        return first > second;
    }];

    Robots = sortedArray;
    int n = (int)Robots.count - 2;
    for (int i = 1; i <= n + 1; i++) {
        Robot *r = Robots[i];
        r.robotView.tag = i;
        if (r.virtuel) {
            [r.robotView drawRobot:[NSString stringWithFormat:i == 0 ? @"0": @"L"]
            withColor:[UIColor grayColor]];
        } else {
            [r.robotView drawRobot:[NSString stringWithFormat:@"%d", i]];
        }
        [self placeRobot:r around:r.x];
    }
}

- (void)placeRobot:(Robot *)robot around:(double)x
{
    RobotView *robotView = robot.robotView;
    robot.x = self.switchIntOnly.on ? round(x) : x;

    CGFloat xViewMin = SEGMENT_VIEW_MARGIN;
    CGFloat xViewMax = segmentViewMax;
    CGFloat xInView = SEGMENT_VIEW_MARGIN + robot.x * segmentViewUnity;

    if (xInView < xViewMin) xInView = xViewMin;
    if (xInView > xViewMax) xInView = xViewMax;

    robot.xLocation = xInView;
    [robotView moveRobotTo:CGPointMake(xInView, SEGMENT_VIEW_Y)];

    [self showOptimalFor:self.tempsMax];
}

- (double)checkReachLimit:(double)reach nextRobot:(Robot *)nextRobot lastRobot:(Robot *)
lastRobot colorType:(int *)colorType
{
    if (fabs(reach - nextRobot.x) < self.precision) {
        reach = nextRobot.x;
        *colorType = COLOR_TYPE_OK;
    } else if (reach > nextRobot.x) {
        reach = nextRobot.x;
        *colorType = COLOR_TYPE_TOO_MUCH;
    } else if (nextRobot == lastRobot) {
        if (fabs(reach - L) > self.precision) {
            *colorType = COLOR_TYPE_NOT_ENOUGH;
        }
    }
    return reach;
}
```

```
- (void)showTrajectoryForRobot:(Robot *)robot
{
    CGFloat xLeft;
    CGFloat xRight;
    double m = SEGMENT_VIEW_MARGIN;
    int n = (int)Robots.count - 2;

    Robot *prevRobot, *nextRobot, *virtualRobot0, *virtualRobotL;
    long iRobot = robot.robotView.tag;

    virtualRobot0 = Robots[0];
    virtualRobotL = Robots[n + 1];

    prevRobot = (iRobot > 0) ? Robots[iRobot - 1] : virtualRobot0;
    nextRobot = (iRobot < n) ? Robots[iRobot + 1] : virtualRobotL;

    double delta, T, g_i, d_i;
    int colorType = COLOR_TYPE_OK;
    delta = robot.x - prevRobot.D;
    T = self.tempsMax;
    long dLine = 2 + robot.robotView.tag % 2 * 8 ;
    if (delta == 0.0) {
        // seulement droite
        robot.state = 3;
        d_i = prevRobot.D + T;
        robot.D = [self checkReachLimit:d_i nextRobot:nextRobot lastRobot:virtualRobotL
        colorType:&colorType];
        xRight = m + robot.D * segmentViewUnity;
        [robot.trajectoryView drawLineAtY:dLine fromX1:robot.xLocation toX2:xRight
        colorType:colorType];
    } else if (T < delta && (delta - T) > self.precision) {
        // pas assez
        robot.state = 0;
        g_i = prevRobot.D + (delta - T);
        robot.D = robot.x;
        xLeft = m + g_i * segmentViewUnity;
        [robot.trajectoryView drawLineAtY:dLine fromX1:robot.xLocation toX2:xLeft colorType
        :COLOR_TYPE_NOT_ENOUGH];
    } else if (delta > T / 3) {
        // droite et ensuite gauche
        robot.state = 1;
        d_i = prevRobot.D + (delta + T) / 2;
        robot.D = [self checkReachLimit:d_i nextRobot:nextRobot lastRobot:virtualRobotL
        colorType:&colorType];
        xLeft = m + prevRobot.D * segmentViewUnity;
        xRight = m + robot.D * segmentViewUnity;
        [robot.trajectoryView drawLineAtY:dLine fromX1:robot.xLocation toX2:xRight toX3:
        xLeft colorType:colorType];
    } else {
        // gauche et ensuite droite
        robot.state = 2;
        d_i = prevRobot.D + (T - delta);
        robot.D = [self checkReachLimit:d_i nextRobot:nextRobot lastRobot:virtualRobotL
        colorType:&colorType];
        xLeft = m + prevRobot.D * segmentViewUnity;
        xRight = m + robot.D * segmentViewUnity;
        [robot.trajectoryView drawLineAtY:dLine fromX1:robot.xLocation toX2:xLeft toX3:
        xRight colorType:colorType];
    }
    Robot *lastRealRobot = Robots[n];
    if (lastRealRobot.state > 0 && lastRealRobot.D >= virtualRobotL.x) {
        virtualRobotL.state = 1;
    } else {
        virtualRobotL.state = 0;
    }
}

- (void)showTemps
{
    int n = (int)Robots.count - 2;
    NSString *s = @"<tr>"
                        "<th>Robot</th>"
```

```
                        "<th>Position</th>"
                        "<th>État</th>"
                        "<th>Vitesse</th>"
                    "</tr>\n";
    for (int i = 1; i <= n + 1; i++) {
        Robot *r = Robots[i];
        NSString *clr = r.virtuel ? @" style='background-color:#cccccc;'" : @"";
        s = [NSString stringWithFormat:
             @"%@<tr%@>"
                    "<td align='center'>%@</td>"
                     "<td align='right'>%g</td>"
                     "<td align='right'>%d</td>"
                     "<td align='right'>%g</td>"
             "</tr>",
             s,
             clr,
             i == n + 1 ? @"L" : [NSString stringWithFormat:@"%d", i],
             r.x,
             r.state,
             r.speed
             ];
    }
    NSString *embedHTML = [NSString stringWithFormat:@"<html><head></head>"
                        "<body style='font-size: 10pt; font-family: Helvetica Light; '>"
                        "<div style='float:left'>"
                            "<table cellspacing='0' cellpadding='2' border='0' style='font-
                            size: 10pt;'>"
                            "%@"
                            "</table>"
                        "</div>"
                        "<div style='float:left; margin-left:50px;'>"
                           "<table cellspacing='2' cellpadding='2' border='0' style='font-
                           size: 10pt;'>"
                           "%@"
                           "</table>"
                        "</div>"
                        "</body></html>", s, [self showTrace]];
    [self.tableRobotInfos loadHTMLString:embedHTML baseURL:nil];
}

- (void)showOptimalFor:(double)temps
{
    self.tempsMax = temps;
    [self showTemps];

    for (Robot *r in Robots) {
        if (!r.virtuel) {
            [self showTrajectoryForRobot:r];
        }
    }
    int n = (int)Robots.count - 2;
    for (int i = 1; i <= n; i++) {
        Robot *r = (Robot *)Robots[i];
        Robot *r_prev = (Robot *)Robots[i - 1];
        switch (r.state) {
            case 0:
                r.speed = 0.0;
                break;
            case 1:
                r.speed = (r_prev.speed + 1.0) / 2.0;
                break;
            case 2:
                r.speed = 2.0 * r_prev.speed + 1.0;
                break;
            case 3:
                r.speed = 1.0;
                break;
        }
    }
}

- (NSString *)showTrace
{
```

```
    NSString *t = [NSString stringWithFormat:
                   @"<tr><th align='right'>Temps</th><td align='right' width='50px'>%.6g</
                   td></tr>",
                   self.sliderTemps.value];
    return t;
}

// ----------------------- Actions

- (IBAction)btnMenuClick:(UIButton *)sender {
    [self.menu saveRobots:Robots intOnly:self.switchIntOnly.on];
    [self dismissViewControllerAnimated:true completion:nil];
}

- (IBAction)sliderTempsChanged:(UISlider *)sender
{
    if (self.switchIntOnly.on) {
        [sender setValue:roundf(sender.value) animated:NO];
    }
    [self showOptimalFor:[sender value]];
}

- (IBAction)switchIntOnlyChanged:(UISwitch *)sender
{
    [self sliderTempsChanged:self.sliderTemps];
}

- (IBAction)btnZoomClicked:(id)sender
{
    self.zoom = !self.zoom;
    [self.btnZoom setTitle:(self.zoom ? @"[<>]" : @">|<") forState:UIControlStateNormal];
    if (self.zoom) {
        self.switchIntOnly.on = NO;
        self.sliderTemps.minimumValue = self.sliderTemps.value - 0.1;
        self.sliderTemps.maximumValue = self.sliderTemps.value + 0.1;
        self.switchIntOnly.enabled = NO;
    } else {
        self.switchIntOnly.enabled = YES;
        [self adjustSliderTemps];
    }
}

- (IBAction)handleTap:(UITapGestureRecognizer *)tap
{
    CGPoint touchPoint = [tap locationOfTouch:0 inView:self.viewMain];

    if (touchPoint.y > SEGMENT_VIEW_Y - ROBOT_RADIUS_SENSITIVITY
        && touchPoint.y < SEGMENT_VIEW_Y + ROBOT_RADIUS_SENSITIVITY
        && touchPoint.x > self.viewMain.frame.origin.x + SEGMENT_VIEW_MARGIN
        && touchPoint.x < self.viewMain.frame.origin.x + self.viewMain.frame.size.width -
        SEGMENT_VIEW_MARGIN
        )
    {
        [self addRobotAt: touchPoint.x];
        [self redrawRobots];
        [[NSNotificationCenter defaultCenter] postNotificationName:@"recalc" object:self];
    }
}

- (IBAction)handlePan:(UIPanGestureRecognizer *)pan
{
    CGPoint location = [pan locationInView:self.viewMain];

    if (pan.state == UIGestureRecognizerStateBegan) {
        if (location.y > SEGMENT_VIEW_Y - ROBOT_Y_SENSITIVITY && location.y <
        SEGMENT_VIEW_Y + 5) {
            for (Robot *r in Robots) {
                if (!r.virtuel) {
                    if (location.x > r.xLocation - ROBOT_RADIUS_SENSITIVITY
                        && location.x < r.xLocation + ROBOT_RADIUS_SENSITIVITY) {
                        self.movingRobot = r;
                        [self.viewMain bringSubviewToFront:self.movingRobot.robotView];
                        break;
```

```
                    }
                }
            }

        }
    }

    if (pan.state == UIGestureRecognizerStateEnded) {
        self.movingRobot = nil;
    }

    if (self.movingRobot) {

        if (location.y < SEGMENT_VIEW_Y - ROBOT_Y_SENSITIVITY && Robots.count > 3)  {
            // enlever le robot
            NSMutableArray *a = [NSMutableArray arrayWithArray:Robots];
            [self.movingRobot.robotView removeFromSuperview];
            [self.movingRobot.trajectoryView removeFromSuperview];
            [a removeObject:self.movingRobot];
            Robots = a;
            [self adjustSliderTemps];
            [self redrawRobots];
        }
        CGFloat xNewLocation = location.x;

        if (xNewLocation > segmentViewMax) xNewLocation = segmentViewMax;
        if (xNewLocation < SEGMENT_VIEW_MARGIN) xNewLocation = SEGMENT_VIEW_MARGIN;

        long tag = self.movingRobot.robotView.tag;
        Robot *prevRobot, *nextRobot;
        prevRobot = (tag > 1) ? Robots[tag - 1] : nil;
        nextRobot = (tag < Robots.count - 1) ? Robots[tag + 1] : nil;

        if (prevRobot && location.x < prevRobot.xLocation) {
            xNewLocation = prevRobot.xLocation;
        }
        if (nextRobot && location.x > nextRobot.xLocation) {
                xNewLocation = nextRobot.xLocation;
        }

        self.movingRobot.xLocation = xNewLocation;
        [pan setTranslation:CGPointMake(0, 0) inView:self.viewMain];
        self.movingRobot.x = (xNewLocation - SEGMENT_VIEW_MARGIN) / segmentViewUnity;
        [self placeRobot:self.movingRobot around:self.movingRobot.x];
        [[NSNotificationCenter defaultCenter] postNotificationName:@"recalc" object:self];
    }
}

// fonction pour deux algorithmes

- (double)porteeRobot:(int)i gauche:(double)G temps:(double)T
{
    int n = (int)Robots.count - 2;
    double r_i = ((Robot *)Robots[i]).x;
    double r_iplus1 = ((Robot *)Robots[i + 1]).x;
    double D = 0.0;
    double d = r_i - G;

    if (d <= 0) {
        D = r_i + T;
    } else if (d > T) {
        D = r_i;
    } else if (d >= T / 3.0) {
        // droit et ensuite gauche
        D = r_i + (T - d) / 2.0;
    } else if (d < T / 3.0) {
        // gauche et ensuite droite
        D = r_i + T - 2.0 * d;
    }
    if (i < n && D > r_iplus1) {
        D = r_iplus1;
    }
    return D;
```

```
}

@end

//
//  AlgoTAViewController.h
//

#import "DemoViewController.h"

@interface AlgoTAViewController : DemoViewController

#define EPSILON 0.001

@end

//
//  AlgoTAViewController.m
//

#import "AlgoTAViewController.h"
#import "Robot.h"
#import "Trace.h"

@interface AlgoTAViewController ()

@property (nonatomic) UILabel *lblEpsilon;
@property (nonatomic) double epsilon;

@end

@implementation AlgoTAViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    self.lblTitle.text = @"Algorithme TA";
    [self algorithmModeOn];
    self.epsilon = EPSILON;

    [self.sliderTemps addTarget:self
                         action:@selector(sliderTempsChanged:)
               forControlEvents:UIControlEventValueChanged];

    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(recalc)
                                                 name:@"recalc"
                                               object:nil];
    [self recalc];
}

- (void)viewDidLayoutSubviews
{
    [super viewDidLayoutSubviews];
    self.lblEpsilon.frame = CGRectMake(self.view.frame.size.width - 150 - 10, 5, 150, 22);
}

- (UILabel *)lblEpsilon
{
    if (!_lblEpsilon) {
        _lblEpsilon = [[UILabel alloc] init];
        _lblEpsilon.frame = CGRectMake(0, 0, 150, 21);
        _lblEpsilon.font = [UIFont fontWithName:@"Helvetica Light" size:14];
        _lblEpsilon.userInteractionEnabled = YES;
        _lblEpsilon.textAlignment = NSTextAlignmentRight;
        _lblEpsilon.textColor = [UIColor colorWithRed:0 green:128/255.0 blue:255 alpha:1];
        [self.view addSubview:_lblEpsilon];

        UISwipeGestureRecognizer *swipeLeft = [[UISwipeGestureRecognizer alloc]
        initWithTarget:self action:@selector(swipedEpsilonLeft:)];
        swipeLeft.direction = UISwipeGestureRecognizerDirectionLeft;
        swipeLeft.numberOfTouchesRequired = 1;
        [_lblEpsilon addGestureRecognizer:swipeLeft];

        UISwipeGestureRecognizer *swipeRight = [[UISwipeGestureRecognizer alloc]
        initWithTarget:self action:@selector(swipedEpsilonRight:)];
```

```
        swipeRight.direction = UISwipeGestureRecognizerDirectionRight;
        swipeRight.numberOfTouchesRequired = 1;
        [_lblEpsilon addGestureRecognizer:swipeRight];
    }
    return _lblEpsilon;
}

- (IBAction)sliderTempsChanged:(UISlider *)sender
{
    int sliderValue = roundf(sender.value);
    [sender setValue:sliderValue animated:NO];
    if (!self.trace) {
        [self recalc];
    }
    [self showOptimalFor:[self.trace[sliderValue] temps]];
}

- (NSString *)showTrace
{
    int sliderValue = roundf(self.sliderTemps.value);
    Trace *trace = self.trace[sliderValue];
    int prec = 4;
    double p = self.precision * 10;
    while ((p *= 10.0) < 1.0) {
        prec++;
    }
    NSString *t = [NSString stringWithFormat:
                   @"<tr><th align='right'>Itération</th><td align='right' width='100px'>%g
                   </td></tr>\n"
                    "<tr><th align='right'>Temps</th><td align='right'>%.*g</td></tr>\n"
                    "<tr><th align='right'>Exploré</th><td align='right'>%@</td></tr>\n"
                    "<tr><th align='right'>Intervalle</th><td align='right'>[%.*g, %.*g]</
                    td></tr>\n",
                    self.sliderTemps.value,
                    prec, trace.temps,
                    trace.possible ? @"Oui" : @"Non",
                    prec, trace.minTemps,
                    prec, trace.maxTemps
    ];
    return t;
}

- (void)swipedEpsilonLeft:(UIGestureRecognizer *)recognizer
{
    self.epsilon *= 10;
    if (self.epsilon > 0.1) {
        self.epsilon = 0.1;
    }
    self.precision = self.epsilon / 10;
    [self displayEpsilon];
    [[NSNotificationCenter defaultCenter] postNotificationName:@"recalc" object:self];
}

- (void)swipedEpsilonRight:(UIGestureRecognizer *)recognizer
{
    self.epsilon /= 10;
    if (self.epsilon < 0.00000001) {
        self.epsilon = 0.00000001;
    }
    self.precision = self.epsilon / 10;
    [self displayEpsilon];
    [[NSNotificationCenter defaultCenter] postNotificationName:@"recalc" object:self];
}

- (void)displayEpsilon
{
    NSNumberFormatter *formatter = [[NSNumberFormatter alloc] init];
    formatter.numberStyle = NSNumberFormatterDecimalStyle;
    formatter.maximumFractionDigits = 8;
    self.lblEpsilon.text = [NSString stringWithFormat:@"Précision : %@", [formatter
    stringFromNumber:[NSNumber numberWithDouble:self.epsilon]] ];
}
```

```
- (IBAction)recalc
{
    [self AlgorithmeTA];
    self.sliderTemps.value = 0;
    [self displayEpsilon];
    [self showOptimalFor:[self.trace[0] temps]];
}

- (double)AlgorithmeTA
{
    double Min = [Robots[1] x];
    double Max = 1.5 * L;
    BOOL possible = YES;

    NSMutableArray *a = [[NSMutableArray alloc] init];
    [a addObject:[Trace traceWithTemps:Max
                               minTemps:Min
                               maxTemps:Max
                               possible:possible]];

    double Mid;
    while (Max - Min > self.epsilon) {
        Mid = (Min + Max) / 2.0;
        if ((possible = [self exploPossible:Mid])) {
            Max = Mid;
        } else {
            Min = Mid;
        }
        [a addObject:[Trace traceWithTemps:Mid
                                   minTemps:Min
                                   maxTemps:Max
                                   possible:possible]];
    }
    self.trace = a;
    self.sliderTemps.maximumValue = self.trace.count - 1;
    return Max;
}

- (BOOL)exploPossible:(double)T
{
    int n = (int)Robots.count - 2;
    double droit;
    double gauche = 0;
    for (int i = 1; i <= n; i++) {
        double r_i = ((Robot *)Robots[i]).x;
        if (r_i - gauche > T) {
            return NO;
        }
        droit = [self porteeRobot:i
                          gauche:gauche
                           temps:T];
        gauche = droit;
    }
    return gauche >= L;
}

@end

//
//  AlgoTEViewController.h
//

#import "DemoViewController.h"

@interface AlgoTEViewController : DemoViewController

@end

//
//  AlgoTEViewController.m
//

#import "AlgoTEViewController.h"
```

```
#import "Robot.h"
#import "Trace.h"

@interface AlgoTEViewController () {
    NSMutableArray *S, *V;
    double infinity;
}
@end

@implementation AlgoTEViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    self.lblTitle.text = @"Algorithme TE";
    [self algorithmModeOn];

    [self.sliderTemps addTarget:self action:@selector(sliderTempsChanged:) forControlEvents
    :UIControlEventValueChanged];

    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(recalc)
                                                 name:@"recalc"
                                               object:nil];
    [self recalc];
}

- (IBAction)sliderTempsChanged:(UISlider *)sender
{
    int sliderValue = roundf(sender.value);
    [sender setValue:sliderValue animated:NO];
    if (!self.trace) {
        [self recalc];
    }
    [self showOptimalFor:[self.trace[sliderValue] temps]];
}

- (NSString *)showTrace
{
    int n = (int)Robots.count - 2;
    int sliderValue = roundf(self.sliderTemps.value);
    Trace *trace = self.trace[sliderValue];
    NSString *t = @"";
    if (sliderValue > 0) {
        t = [NSString stringWithFormat:
                    @"<tr><th align='right'>Itération</th><td align='right' width='50px'>%g
                    </td></tr>"
                     "<tr><th align='right'>Temps</th><td align='right'>%.6g<td></tr>"
                     "<tr><th align='right'>Changement</th><td align='right'>Robot %@<td></
                     tr>"
                     "<tr><th align='right'>d'état</th><td align='right'>%d -> %d<td></tr>",
                    self.sliderTemps.value,
                    trace.temps,
                    trace.iRobotCh == (n + 1) ? @"L" : @(trace.iRobotCh),
                    trace.statePrev, trace.state
             ];
    }
    return t;
}

- (IBAction)recalc
{
    [self AlgorithmeTE];
    self.sliderTemps.value = 0;
    [self showOptimalFor:0];
}

- (double)AlgorithmeTE
{
    double T = 0.0;
    int iRobotCh;
    int n = (int)Robots.count - 2;
    infinity = 2.0 * L;
    S = [[NSMutableArray alloc] init];
    V = [[NSMutableArray alloc] init];
```

```
    for (int i = 0; i <= n + 1; i++) {
        [S addObject:@0];
    }
    for (int i = 0; i <= n; i++) {
        [V addObject:@0.0];
    }
    NSMutableArray *aTrace = [[NSMutableArray alloc] init];
    [aTrace addObject:[Trace traceWithTemps:0]];
    while ([self ExisteRobotEnEtatZero]) {
        [self CalculerVitesses];
        T = [self ProchainChangement:T iRobotCh:&iRobotCh];
        [aTrace addObject:[Trace traceWithTemps:T
                                       iRobotCh:iRobotCh
                                      statePrev:[S[iRobotCh] intValue]
                                          state:[S[iRobotCh] intValue] + 1]];
        S[iRobotCh] = @([S[iRobotCh] intValue] + 1);
    }
    self.trace = aTrace;
    self.sliderTemps.maximumValue = self.trace.count - 1;
    return T;
}

- (BOOL)ExisteRobotEnEtatZero
{
    int n = (int)Robots.count - 2;
    for (int i = 1; i <= n + 1; i++) {
        if ([S[i] intValue] == 0) {
            return YES;
        }
    }
    return NO;
}

- (void)CalculerVitesses
{
    int n = (int)Robots.count - 2;
    for (int i = 1; i <= n; i++) {
        double v_i_1 = [V[i - 1] doubleValue];
        switch ([S[i] intValue]) {
            case 0:
                V[i] = @0.0;
                break;
            case 1:
                V[i] = @((v_i_1 + 1.0) / 2.0);
                break;
            case 2:
                V[i] = @(2.0 * v_i_1 + 1.0);
                break;
            case 3:
                V[i] = @1.0;
                break;
        }
    }
}

- (double)ProchainChangement:(double)T iRobotCh:(int *)iRobotCh
{
    double TempsCh = infinity;
    double Gauche = 0, Droit;
    double Ch = 0;
    int n = (int)Robots.count - 2;
    for (int i = 1; i <= n; i++) {
        double r_i = [(Robot *)Robots[i] x];
        double v_i_1 = [V[i - 1] doubleValue];
        switch ([S[i] intValue]) {
            case 0:
                Ch = (T * v_i_1 + r_i - Gauche) / (v_i_1 + 1.0);
                break;
            case 1:
                Ch = (T * v_i_1 + r_i - Gauche) / (v_i_1 + 1.0 / 3.0);
                break;
            case 2:
                Ch = v_i_1 == 0.0 ? infinity : (T + (r_i - Gauche) / v_i_1);
```

```
                break;
            case 3:
                Ch = infinity;
        }
        if (Ch < TempsCh) {
            TempsCh = Ch;
            *iRobotCh = i;
        }
        Droit = [self porteeRobot:i gauche:Gauche temps:T];
        Gauche = Droit;
    }
    if ([S[n + 1] intValue] == 0 && [S[n] intValue] > 0) {
        Ch = T + (L - Gauche) / [V[n] doubleValue];
        if (Ch < TempsCh) {
            TempsCh = Ch;
            *iRobotCh = n + 1;
        }
    }
    return TempsCh;
}

@end

//
//  Robot.h
//

#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>
#import "RobotView.h"
#import "TrajectoryView.h"

@interface Robot : NSObject

@property (nonatomic) BOOL virtuel;
@property (nonatomic) double positionInitiale;
@property (nonatomic) double D;
@property (nonatomic) double x;
@property (nonatomic) int state;
@property (nonatomic) double speed;
@property (nonatomic) CGFloat xLocation;
@property (nonatomic, strong) RobotView *robotView;
@property (nonatomic, strong) TrajectoryView *trajectoryView;

+ (Robot *)robotAt:(double)x virtuel:(BOOL)virtuel;

- (void)assignRobotView:(RobotView *)rv;
- (void)assignRobotTrajectory:(TrajectoryView *)tv;

@end

//
//  Robot.m
//

#import "Robot.h"

@implementation Robot

- (void)assignRobotView:(RobotView *)rv
{
    self.robotView = rv;
}

- (void)assignRobotTrajectory:(TrajectoryView *)tv
{
    self.trajectoryView = tv;
}

+ (Robot *)robotAt:(double)x virtuel:(BOOL)virtuel
{
    Robot *robot = [[Robot alloc] init];
    robot.x = x;
    robot.virtuel = virtuel;
```

```
    return robot;
}

@end

//
//  RobotView.h
//

#import <UIKit/UIKit.h>

@interface RobotView : UIView

- (void)drawRobot:(NSString *)title;
- (void)drawRobot:(NSString *)title withColor:(UIColor *)pColor;
- (void)moveRobotTo:(CGPoint)point;

@end

//
//  RobotView.m
//

#import "RobotView.h"

@implementation RobotView {
    CGPoint p0;
    UIColor *penColor;
    UIColor *xFillColor;
    UILabel *label;
}

- (id)init
{
    CGRect frame = CGRectMake(0, 0, 40, 40);
    return [self initWithFrame:frame];
}

- (id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        self.backgroundColor = [UIColor clearColor];
    }
    return self;
}

- (void)drawRect:(CGRect)rect
{
    CGContextRef context = UIGraphicsGetCurrentContext();
    int penWidth = 2;
    CGContextSetLineWidth(context, penWidth);
    CGContextSetStrokeColorWithColor(context, penColor.CGColor);

    CGContextMoveToPoint(context, p0.x, p0.y - 1);
    CGContextAddLineToPoint(context, p0.x, p0.y - 3 * penWidth);
    CGContextStrokePath(context);
    int robotSize = 24;

    CGContextSetFillColorWithColor(context, xFillColor.CGColor);
    CGContextFillEllipseInRect(context, CGRectMake(p0.x - robotSize / 2, p0.y - robotSize -
     3 * penWidth, robotSize, robotSize));

    CGContextStrokePath(context);
}

- (void)drawRobot:(NSString *)title;
{
    [self drawRobot:title withColor:[UIColor colorWithRed:0.3 green:0.6 blue:0.75 alpha
    :1]];
}

- (void)drawRobot:(NSString *)title withColor:(UIColor *)pColor
{
```

```
    p0 = CGPointMake(20, 40);
    penColor = pColor;
    if (!label) {
        label = [[UILabel alloc] init];
        label.frame = CGRectMake(12, 10, 20, 20);
        CGRect frame;
        frame = self.frame;
        frame.origin.y += 2;
        label.frame = frame;
        label.font = [UIFont fontWithName:@"Helvetica Light" size:14];
        label.textAlignment = NSTextAlignmentCenter;
        [self addSubview:label];
    }
    label.text = title;

    xFillColor = penColor;
    [self setNeedsDisplay];
}

- (void)drawRobotAt:(CGPoint)point withColor:(UIColor *)pColor;
{
    p0 = CGPointMake(20, 40);
    penColor = pColor;
    xFillColor = penColor;
    [self setNeedsDisplay];
}

- (void)moveRobotTo:(CGPoint)point
{
    CGRect frame = self.frame;
    frame.origin.x = point.x - 20;
    frame.origin.y = point.y - 40;
    self.frame = frame;
}

@end

//
//  SegmentView.h
//

#import <UIKit/UIKit.h>

@interface SegmentView : UIView

@property (nonatomic) UIColor *penColor;
@property (nonatomic) int maxVal;

- (void)drawSegmentAt:(CGFloat)y withMargin:(CGFloat)margin;

@end

//
//  SegmentView.m
//

#import "SegmentView.h"

@implementation SegmentView {
    CGFloat yLine;
    CGFloat marginLR;
}

- (id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        self.backgroundColor = [UIColor clearColor];
    }
    return self;
}

- (void)drawRect:(CGRect)rect
{
```

```
    self.penColor = [UIColor grayColor];
    float x0 = marginLR, x1;
    float y = yLine;

    x1 = self.frame.size.width - x0;

    CGFloat x_length = x1 - x0;
    float chunk = x_length / self.maxVal;

    CGContextRef context = UIGraphicsGetCurrentContext();
    CGContextSetLineWidth(context, 1.0);
    CGColorSpaceRef colorspace = CGColorSpaceCreateDeviceRGB();
    CGFloat components[] = {0.0, 0.0, 1.0, 1.0};
    CGColorRef color = CGColorCreate(colorspace, components);
    CGContextSetStrokeColorWithColor(context, self.penColor.CGColor);

    CGContextMoveToPoint(context, x0, y);
    CGContextAddLineToPoint(context, x1, y);

    CGContextMoveToPoint(context, x0, y - 5);
    CGContextAddLineToPoint(context, x0, y + 5);

    for (int i = 1; i < self.maxVal; i++) {
        CGFloat x = x0 + i * chunk;
        CGContextMoveToPoint(context, x, y);
        CGContextAddLineToPoint(context, x, y + 3);
    }

    CGContextMoveToPoint(context, x1, y - 5);
    CGContextAddLineToPoint(context, x1, y + 5);

    CGContextStrokePath(context);

    CGColorSpaceRelease(colorspace);
    CGColorRelease(color);

    for (int i = 0; i <= self.maxVal; i++) {
        [self addLabel:[NSString stringWithFormat:@"%d", i] at:x0 + chunk * i];
    }

}

- (void)addLabel:(NSString *)nb at:(CGFloat)x
{
    float y = yLine;
    UILabel *l = [[UILabel alloc] init];
    l.frame = CGRectMake(x - 20 / 2, y, 20, 24);
    l.text = nb;
    l.textAlignment = NSTextAlignmentCenter;
    l.font = [UIFont fontWithName:@"Helvetica Light" size:10];
    [self addSubview:l];
}

- (void)drawSegmentAt:(CGFloat)y withMargin:(CGFloat)margin
{
    yLine = y;
    marginLR = margin;
    [self setNeedsDisplay];
}

@end

//
//  TrajectoryView.h
//

#import <UIKit/UIKit.h>

@interface TrajectoryView : UIView

- (void)drawLineAtY:(CGFloat)y fromX1:(CGFloat)x1 toX2:(CGFloat)x2 toX3:(CGFloat)x3
colorType:(int)colorType;
- (void)drawLineAtY:(CGFloat)y fromX1:(CGFloat)x1 toX2:(CGFloat)x2 colorType:(int)colorType
;
```

```
@end

//
//  TrajectoryView.m
//

#import "TrajectoryView.h"

@implementation TrajectoryView {
    CGPoint p0;
    CGPoint pStart, pReturn, pEnd, pEnd2;
    CGFloat yPlus;
    UIColor *penColor;
}

- (id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        self.backgroundColor = [UIColor clearColor];
    }
    return self;
}

- (void)drawRect:(CGRect)rect
{
    CGContextRef context = UIGraphicsGetCurrentContext();
    int ropeWidth = 3;
    CGContextSetLineWidth(context, ropeWidth);
    CGContextSetStrokeColorWithColor(context, penColor.CGColor);
    CGContextMoveToPoint(context, pStart.x, pStart.y);
    CGContextAddLineToPoint(context, pEnd.x, pEnd.y);

    CGContextMoveToPoint(context, pEnd.x, pStart.y + ropeWidth + 1);
    CGContextAddLineToPoint(context, pEnd2.x, pStart.y + ropeWidth + 1);

    CGContextStrokePath(context);
    CGContextSetStrokeColorWithColor(context, [UIColor blueColor].CGColor);

    if (pEnd.x < pEnd2.x) {
        CGContextMoveToPoint(context, pEnd2.x - 1.5, pStart.y + ropeWidth  + 2.5);
        CGContextAddLineToPoint(context, pEnd2.x - 1.5, pStart.y + ropeWidth + 2.5 + yPlus)
        ;
    } else {
        CGContextMoveToPoint(context, pEnd2.x + 1.5, pStart.y + ropeWidth  + 2.5);
        CGContextAddLineToPoint(context, pEnd2.x + 1.5, pStart.y + ropeWidth + 2.5 + yPlus)
        ;
    }
    CGContextStrokePath(context);
}

- (void)drawLineAtY:(CGFloat)y fromX1:(CGFloat)x1 toX2:(CGFloat)x2 colorType:(int)colorType
{
    [self drawLineAtY:y fromX1:x1 toX2:x2 toX3:x2 colorType:colorType];
}

- (void)drawLineAtY:(CGFloat)y fromX1:(CGFloat)x1 toX2:(CGFloat)x2 toX3:(CGFloat)x3
colorType:(int)colorType
{
    pStart = CGPointMake(x1, y);
    pEnd = CGPointMake(x2, y);
    pEnd2 = CGPointMake(x3, y);
    yPlus = 0;
    if (colorType > 0) {
        yPlus = colorType * 3;
    }
    switch (colorType) {
        case -1:
            penColor = [UIColor redColor];
            break;
        case 0:
            penColor = [UIColor colorWithRed:0 green:0.8 blue:0 alpha:1];
            break;
        default:
```

```
            penColor = [UIColor blueColor];
            break;
    }
    [self setNeedsDisplay];
}

@end

//
//  Trace.h
//

#import <Foundation/Foundation.h>

@interface Trace : NSObject

@property (nonatomic) double temps;
@property (nonatomic) double minTemps;
@property (nonatomic) double maxTemps;
@property (nonatomic) BOOL possible;

@property (nonatomic) int iRobotCh;
@property (nonatomic) int statePrev;
@property (nonatomic) int state;

+ (Trace *)traceWithTemps:(double)temps;
+ (Trace *)traceWithTemps:(double)temps minTemps:(double)minTemps maxTemps:(double)maxTemps
 possible:(BOOL)possible;
+ (Trace *)traceWithTemps:(double)temps iRobotCh:(int)iRobotCh statePrev:(int)statePrev
state:(int)state;

@end

//
//  Trace.m
//

#import "Trace.h"

@implementation Trace

+ (Trace *)traceWithTemps:(double)temps
{
    Trace *trace = [[Trace alloc] init];
    trace.temps = temps;
    return trace;
}

+ (Trace *)traceWithTemps:(double)temps minTemps:(double)minTemps maxTemps:(double)maxTemps
 possible:(BOOL)possible
{
    Trace *trace = [[Trace alloc] init];
    trace.temps = temps;
    trace.minTemps = minTemps;
    trace.maxTemps = maxTemps;
    trace.possible = possible;
    return trace;
}

+ (Trace *)traceWithTemps:(double)temps iRobotCh:(int)iRobotCh statePrev:(int)statePrev
state:(int)state
{
    Trace *trace = [[Trace alloc] init];
    trace.temps = temps;
    trace.iRobotCh = iRobotCh;
    trace.statePrev = statePrev;
    trace.state = state;
    return trace;
}

@end
```

Bibliographie

[1] Eric Aaron, Evangelos Kranakis et Danny Krizanc : On the complexity of the multi-robot, multi-depot map visitation problem. *In IEEE 8th International Conference on Mobile Adhoc and Sensor Systems, MASS 2011, Valencia, Spain, October 17-22, 2011*, pages 795–800. IEEE, 2011.

[2] Noa Agmon, Chien-Liang Fok, Yehuda Emaliah, Peter Stone, Christine Julien et Sriram Vishwanath : On coordination in practical multi-robot patrol. *In IEEE International Conference on Robotics and Automation, ICRA 2012, 14-18 May, 2012, St. Paul, Minnesota, USA*, pages 650–656. IEEE, 2012.

[3] Susanne Albers et Monika Rauch Henzinger : Exploring unknown environments. *In* Frank Thomson Leighton et Peter W. Shor, éditeurs : *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 416–425. ACM, 1997.

[4] Steve Alpern et Shmuel Gal : *The theory of search games and rendezvous*, volume 55. Springer, 2003.

[5] Julian Anaya, Jérémie Chalopin, Jurek Czyzowicz, Arnaud Labourel, Andrzej Pelc et Yann Vaxès : Collecting information by power-aware mobile agents. *In* MarcosK. Aguilera, éditeur : *Distributed Computing*, volume 7611 de *Lecture Notes in Computer Science*, pages 46–60. Springer Berlin Heidelberg, 2012.

[6] Ricardo A. Baeza-Yates, Joseph C. Culberson et Gregory J. E. Rawlins : Searching in the plane. *Inf. Comput.*, 106(2):234–252, 1993.

[7] Ricardo A. Baeza-Yates et René Schott : Parallel searching in the plane. *Comput. Geom.*, 5:143–154, 1995.

[8] Anatole Beck : On the linear search problem. *Israel Journal of Mathematics*, 2(4):221–228, 1964.

[9] Michael A. Bender, Antonio Fernández, Dana Ron, Amit Sahai et Salil P. Vadhan : The power of a pebble : Exploring and mapping directed graphs. *Inf. Comput.*, 176(1):1–21, 2002.

[10] Lélia Blin, Janna Burman et Nicolas Nisse : Perpetual Graph Searching. Rapport technique RR-7897, INRIA, 2012.

[11] Anthony Bonato et Boting Yang : Graph searching and related problems. *In* Panos M. Pardalos, Ding-Zhu Du et Ronald L. Graham, éditeurs : *Handbook of Combinatorial Optimization*, pages 1511–1558. Springer New York, 2013.

[12] W. Burgard, M. Moors, D. Fox, R. Simmons et S. Thrun : Collaborative multi-robot exploration. *In IEEE International Conference on Robotics and*

Automation, ICRA 2000, volume 1, pages 476–481 vol.1, 2000.

[13] W. Burgard, M. Moors, C. Stachniss et F.E. Schneider : Coordinated multi-robot exploration. *Robotics, IEEE Transactions on*, 21(3):376–386, June 2005.

[14] Wolfram Burgard, Mark Moors et Frank Schneider : Collaborative exploration of unknown environments with teams of mobile robots. *In* Michael Beetz, Joachim Hertzberg, Malik Ghallab et MarthaE. Pollack, éditeurs : *Advances in Plan-Based Control of Robotic Agents*, volume 2466 de *Lecture Notes in Computer Science*, pages 52–70. Springer Berlin Heidelberg, 2002.

[15] Ke Chen, Adrian Dumitrescu et Anirban Ghosh : On fence patrolling by mobile agents. *In Proceedings of the 25th Canadian Conference on Computational Geometry, CCCG 2013, Waterloo, Ontario, Canada, August 8-10, 2013*. Carleton University, Ottawa, Canada, 2013.

[16] Timothy H. Chung, Geoffrey A. Hollinger et Volkan Isler : Search and pursuit-evasion in mobile robotics - A survey. *Auton. Robots*, 31(4):299–316, 2011.

[17] Jurek Czyzowicz, Leszek Gasieniec, Konstantinos Georgiou, Evangelos Kranakis et Fraser MacQuarrie : The beachcombers' problem : Walking and searching with mobile robots. *In* Magnús M. Halldórsson, éditeur : *Structural Information and Communication Complexity - 21st International Colloquium,*

SIROCCO 2014, Takayama, Japan, July 23-25, 2014. Proceedings, volume 8576 de *Lecture Notes in Computer Science*, pages 23–36. Springer, 2014.

[18] Jurek Czyzowicz, Leszek Gasieniec, Adrian Kosowski et Evangelos Kranakis : Boundary patrolling by mobile agents with distinct maximal speeds. *In* Camil Demetrescu et Magnús M. Halldórsson, éditeurs : *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, volume 6942 de *Lecture Notes in Computer Science*, pages 701–712. Springer, 2011.

[19] Jurek Czyzowicz, Leszek Gasieniec, Adrian Kosowski, Evangelos Kranakis, Oscar Morales Ponce et Eduardo Pacheco : Position discovery for a system of bouncing robots. *In* Marcos K. Aguilera, éditeur : *Distributed Computing - 26th International Symposium, DISC 2012, Salvador, Brazil, October 16-18, 2012. Proceedings*, volume 7611 de *Lecture Notes in Computer Science*, pages 341–355. Springer, 2012.

[20] Jurek Czyzowicz, David Ilcinkas, Arnaud Labourel et Andrzej Pelc : Worst-case optimal exploration of terrains with obstacles. *Inf. Comput.*, 225:16–28, 2013.

[21] Jurek Czyzowicz, Evangelos Kranakis et Eduardo Pacheco : Localization for a system of colliding robots. *In* Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska et David Peleg, éditeurs : *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July*

8-12, 2013, Proceedings, Part II, volume 7966 de *Lecture Notes in Computer Science*, pages 508–519. Springer, 2013.

[22] Paola Flocchini, David Ilcinkas, Andrzej Pelc et Nicola Santoro : Computing without communicating : Ring exploration by asynchronous oblivious robots. *In* Eduardo Tovar, Philippas Tsigas et Hacène Fouchal, éditeurs : *Principles of Distributed Systems*, volume 4878 de *Lecture Notes in Computer Science*, pages 105–118. Springer Berlin Heidelberg, 2007.

[23] Fedor V. Fomin, Petr A. Golovach et Pawel Pralat : Cops and robber with constraints. *SIAM J. Discrete Math.*, 26(2):571–590, 2012.

[24] Fedor V. Fomin et Dimitrios M. Thilikos : An annotated bibliography on guaranteed graph searching. *Theor. Comput. Sci.*, 399(3):236–245, 2008.

[25] Tom Friedetzky, Leszek Gasieniec, Thomas Gorry et Russell Martin : Observe and remain silent (communication-less agent location discovery). *In* Branislav Rovan, Vladimiro Sassone et Peter Widmayer, éditeurs : *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, volume 7464 de *Lecture Notes in Computer Science*, pages 407–418. Springer, 2012.

[26] Alan M. Frieze, Michael Krivelevich et Po-Shen Loh : Variations on cops and robbers. *Journal of Graph Theory*, 69(4):383–402, 2012.

[27] Akitoshi Kawamura et Yusuke Kobayashi : Fence patrolling by mobile agents with distinct speeds. *In* Kun-Mao Chao, Tsan-sheng Hsu et Der-Tsai Lee, édi-

teurs : *Algorithms and Computation - 23rd International Symposium, ISAAC 2012, Taipei, Taiwan, December 19-21, 2012. Proceedings*, volume 7676 de *Lecture Notes in Computer Science*, pages 598–608. Springer, 2012.

[28] Jon M. Kleinberg : On-line search in a simple polygon. *In* Daniel Dominic Sleator, éditeur : *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia.*, pages 8–15. ACM/SIAM, 1994.

[29] Evangelos Kranakis et Danny Krizanc : An algorithmic theory of mobile agents. *In* Ugo Montanari, Donald Sannella et Roberto Bruni, éditeurs : *Trustworthy Global Computing*, volume 4661 de *Lecture Notes in Computer Science*, pages 86–97. Springer Berlin Heidelberg, 2007.

[30] Frédéric Lessard : Exploration optimale d'un segment de droite par deux agents mobiles. Mémoire de D.E.A., Université du Québec en Outaouais, 2013.

[31] Joseph S.B. Mitchell : Geometric shortest paths and network optimization. *In Handbook of Computational Geometry*, pages 633–701. Elsevier Science Publishers B.V. North-Holland, 1998.

[32] Giuseppe Oriolo, Giovanni Ulivi et Marilena Vendittelli : Real-time map building and navigation for autonomous robots in unknown environments. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 28(3):316–333, 1998.

[33] Fabio Pasqualetti, Joseph W. Durham et Francesco Bullo : Cooperative patrolling via weighted tours : Performance analysis and distributed algorithms.

IEEE Transactions on Robotics, 28(5):1181–1188, 2012.

[34] David PORTUGAL et Rui P. ROCHA : A survey on multi-robot patrolling algorithms. *In* Luis M. CAMARINHA-MATOS, éditeur : *Technological Innovation for Sustainability - Second IFIP WG 5.5/SOCOLNET Doctoral Conference on Computing, Electrical and Industrial Systems, DoCEIS 2011, Costa de Caparica, Portugal, February 21-23, 2011. Proceedings*, volume 349 de *IFIP Advances in Information and Communication Technology*, pages 139–146. Springer, 2011.

[35] David PORTUGAL et Rui P. ROCHA : Multi-robot patrolling algorithms : examining performance and scalability. *Advanced Robotics*, 27(5):325–336, 2013.

[36] Mélanie ROY : Exploration optimale d'un arbre par un essaim d'agents mobiles. Mémoire de D.E.A., Université du Québec en Outaouais, 2012.

[37] Masafumi YAMASHITA et Tsunehiko KAMEDA : Computing on anonymous networks : Part i-characterizing the solvable cases. *IEEE Trans. Parallel Distrib. Syst.*, 7(1):69–89, 1996.

[38] Vladimir YANOVSKI, Israel A. WAGNER et Alfred M. BRUCKSTEIN : A distributed ant algorithm for efficiently patrolling a network. *Algorithmica*, 37(3):165–186, 2003.