Algorithmes de communication autostabilisants dans un système de robots mobiles

Joyce El Haddad[*,**] — **Serge Haddad**[*]

**LAMSADE*
Université Paris-Dauphine
Place du Maréchal de lattre de Tassigny
F-75775 Paris cedex 16
{elhaddad, haddad}@lamsade.dauphine.fr

***Ecole Supérieure d'Informatique et de GEnie de TELecom*
Département de recherche
1, rue du Port de Valvins
F-77215 Avon cedex
{elhaddad}@esigetel.fr

RÉSUMÉ. *Ce papier traite de la communication dans un système de robots mobiles avec des points de rendez-vous fixes. Nous proposons un algorithme autostabilisant d'ordonnancement des visites des points de rendez-vous assurant qu'après stabilisation, chaque visite aboutit à une communication et un deuxième algorithme autostabilisant, qui établit un routage entre les robots distants.*

ABSTRACT. *This paper deals with communication in autonomous multi-robots system. We propose a self-stabilizing scheduling algorithm that solves the management of visits to the locations ensuring that after stabilization phase, every visit will lead to a communication. Next, we present a second self-stabilizing algorithm, based on the above one, computing the shortest-path for all-pairs of locations.*

MOTS-CLÉS : *systèmes multi-robots, tolérance aux pannes, autostabilisation, réseaux de Petri, chaînes de Markov.*

KEYWORDS: *muli-robots systems, fault tolerance, self-stabilization, Petri nets, Markov chains.*

1. Introduction

L'une des principales motivations du développement de la robotique mobile demeure la substitution de l'être humain en milieu hostile. Les robots mobiles autonomes qui réalisent des tâches sans intervention d'un opérateur sont nécessaires dans plusieurs domaines d'application militaire, d'exploration spatial ou sous-marine . . . Généralement, on cherche à minimiser les risques à l'être humain et à réduire les coûts des opérations. Ainsi, parmi les tendances actuelles en robotique [CAO 97, SEL 00, LUZ 00, DEF 02], on note la mobilité et la coopération en environnement dynamique.

Par exemple, pour certaines tâches telles que l'exploration spatiale, il est plus sûr et même nécessaire d'utiliser un groupe de robots. En effet, les explorateurs (robots) envoyés sur le sol martien se déplacent sur plusieurs kilomètres explorant chacun une zone géologique qui lui est affectée pour la mission. Chacun de ces robots navigue de manière autonome en ramassant des échantillons dans le but des les analyser ou de les rapatrier, mais en groupe et en communiquant ses données à ses voisins afin de permettre la prise de mesures sismiques, météorologiques ou astronomiques.

Dans le but d'obtenir des systèmes robotisés de plus en plus performants, de nombreuses approches furent prises. En effet, la quête visant à la réalisation d'un être synthétique parfait a laissé place à une considération plus pragmatique du problème. Le mythe de la machine capable d'affronter n'importe quel type de problème cède le pas à la réalisation de systèmes tolérants aux pannes temporaires liées à l'environnement qui est méconnu. Dans de tels systèmes, deux types d'algorithme de communications sont nécessaires : l'algorithme de synchronisation entre deux robots voisins afin d'établir une communication point-à-point (temporaire), et l'algorithme de routage permettant l'échange de données entre deux robots (même s'ils sont distants). Dans ce contexte, les protocoles de communications existants [BRA 02, HU 98, PRE 01] supposent que les robots et liens de communications sont fiables. Notre approche consiste à proposer un protocole de communication tolérant les pannes temporaires de certains robots du système. Cette manière de traiter les défaillances est adoptée par les algorithmes autostabilisants. Un tel algorithme garantit que le système retrouve un comportement correct au bout d'un temps fini. Le concept de l'autostabilisation a été introduit par Dijkstra [DIJ 74]. Depuis, de nombreux travaux ont été fait dans ce domaine [DOL 00, SCH 93].

Dans un premier temps, nous proposons un algorithme autostabilisant d'ordonnancement des visites des points de rendez-vous assurant qu'après la phase de stabilisation, chaque visite à un point de rendez-vous aboutit à une communication. Dans un second temps, nous présentons un algorithme autostabilisant, basé sur le premier, qui établit un routage entre les robots distants.

La suite du papier est organisée comme suit. Dans la deuxième section, nous décrivons brièvement un algorithme d'ordonnancement non autostabilisant. Nous utiliserons les réseaux de Petri comme outil pour prouver la correction de cet algorithme. Dans la troisième section, nous décrivons formellement la transformation de cet algorithme en un algorithme autostabilisant dont le comportement se modélise par une

chaîne de Markov à temps discret. Les résultats théoriques dans ce domaine serviront à établir la preuve de stabilisation de l'algorithme. Dans la quatrième section, nous donnons une description informelle de la preuve de stabilisation. Ensuite, nous présentons dans la cinquième section, un deuxième algorithme autostabilisant pour le calcul des plus courts chemins entre un point de rendez-vous fixé et l'ensemble de tous les points de rendez-vous des autres robots. Cet algorithme peut être utilisé pour effectuer le routage. Enfin, nous décrivons la preuve formelle de sa stabilisation dans la dernière section.

2. Algorithme d'ordonnancement non autostabilisant

Dans le contexte des systèmes de robots mobiles, BMR [BRA 02] propose un algorithme d'ordonnancement basé sur une stratégie de déplacement des robots assurant une communication entre robots distants en un temps borné. Nous commençons par la description des hypothèses :

1) Le système est composé d'un ensemble de m robots anonymes (les identités ne seront pas utilisées dans l'algorithme), notés $(r_1, \ldots, r_m)$ et de N points de rendez-vous.

2) A chaque point de rendez-vous (ou location) est associé un unique identifiant numérique. Nous notons $(l_1, \ldots, l_N)$ l'ensemble ordonné des points de rendez-vous du système. Au plus, il y a un point de rendez-vous par paire de robots et la communication ne peut avoir lieu que si les deux robots concernés sont présents au point de rendez-vous.

3) Chaque robot r_i maintient, dans sa mémoire incorruptible, un tableau trié par ordre croissant des identificateurs de ses points de rendez-vous. n_i est la taille de ce tableau et $f(i,j)$ pour $1 \leq i \leq m$ et $0 \leq j \leq n_i - 1$, est l'identificateur associé au $j^{ième}$ point de rendez-vous de r_i.

4) Le réseau est (fortement) connexe : entre deux robots r_i et $r_{i'}$, il existe une séquence de robots $r_i = r_{i_0}, r_{i_1}, \ldots, r_{i_K} = r_{i'}$ tel que pour tout $0 \leq k < K$, r_{i_k} et $r_{i_{k+1}}$ ont un point de rendez-vous en commun.

L'objectif de cet algorithme est de limiter les degrés de liberté du système en imposant la contrainte suivante : le premier robot qui arrive sur le point de rendez-vous attend son homologue avant de continuer son déplacement. Les points de rendez-vous sont arbitrairement numérotés. L'activité d'un robot consiste à parcourir ses points de rendez-vous dans *l'ordre* croissant de leur numéro. Il s'agit de partir du point de rendez-vous ayant le plus petit numéro, de visiter tous les autres points de manière unidirectionnelle et ceci de façon répétitive. Ainsi, chaque point de rendez-vous est visité infiniment souvent.

Les auteurs démontrent que l'ordonnancement produit est exempt de blocage (partiel et total). Dans ce papier, nous utilisons les réseaux de Petri (RdP) comme outil pour proposer une preuve plus rapide et plus simple de correction de cet algorithme. Cette preuve sera à la base de notre version stabilisante de l'algorithme. Pour une

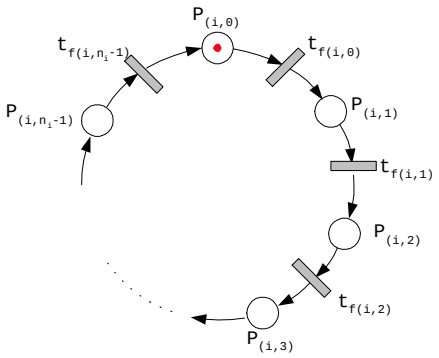

Figure 1. *Le cycle de visite du robot r_i*

bonne compréhension de notre travail, nous supposons une connaissance des concepts de base des réseaux de Petri, sinon nous conseillons l'ouvrage de Reisig [REI 85].

Pour chaque robot r_i, un réseau de Petri modélise le parcours local de ses points de rendez-vous. A partir de ces réseaux locaux, un réseau de Petri produit modélise l'état global du système à tout instant. Le réseau de Petri de la figure 1 représente le comportement d'un robot r_i, et se définit par le tuple $N_i = (P_i, T_i, Pré_i, Post_i, M0_i)$ où :

1) $P_i = \{p_{(i,0)}, \ldots, p_{(i,j)}, \ldots, p_{(i,n_i-1)}\}$ est un ensemble fini et non vide de places représentant l'ensemble des points de rendez-vous. La présence d'un jeton dans la place $p_{(i,j)}$ signifie que r_i est soit en déplacement vers son $j^{ième}$ point de rendez-vous, soit présent sur ce dernier mais "en attente" de son homologue.

2) $T_i = \{t_{f(i,0)}, ..., t_{f(i,j)}, ..., t_{f(i,n_i-1)}\}$ est un ensemble fini et non vide de transitions. La condition de franchissement d'une transition $t_{f(i,j)}$ est que les deux robots, r_i et son homologue, soient présents sur le $j^{ième}$ point de rendez-vous de r_i. Le franchissement de $t_{f(i,j)}$ représente le déplacement des deux robots, chacun vers son point de rendez-vous suivant.

3) $Pré_i$ est la matrice d'incidence arrière définie de $P_i \times T_i$ dans $\{0, 1\}$ comme suit :

$$Pré_i(p,t) = \begin{cases} 1 & \text{si } p = p_{(i,j)} \text{ et } t = t_{f(i,j)}, \\ 0 & \text{sinon.} \end{cases}$$

4) $Post_i$ est la matrice d'incidence avant, définie de $P_i \times T_i$ dans $\{0, 1\}$ comme suit :

$$Post_i(p,t) = \begin{cases} 1 & \text{si } p = p_{(i,(j+1) \bmod n_i)} \text{ et } t = t_{f(i,j)}, \\ 0 & \text{sinon.} \end{cases}$$

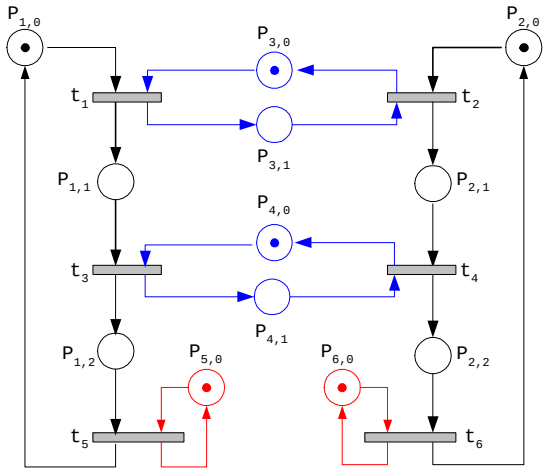

Figure 2. *Le réseau de Petri global d'une instance de l'algorithme*

5) Le marquage initial, noté $M0_i$, définit le nombre de jetons contenus dans toutes les places du réseau. $M0_i(p_{(i,j)})$ définit le marquage initial de la place $p_{(i,j)}$ comme suit :

$$M0_i(p_{(i,j)}) = \begin{cases} 1 & \text{si } j = 0, \\ 0 & \text{sinon.} \end{cases}$$

Le modèle global du système est un réseau de Petri défini par le tuple $N = (P, T, Pré, Post, M0)$ où :

1) $P = \biguplus P_i$, est l'union des toutes les places des réseaux de Petri locaux.

2) $T = \bigcup T_i$, est l'union (non disjointe) de toutes les transitions des réseaux de Petri locaux.

3) $Pré$ et $Post$ sont les matrices d'incidence arrière et avant, définies de $P \times T$ dans $\{0, 1\}$ comme suit :

$$Pré(p,t) = \begin{cases} Pré_i(p,t) & \text{si } p \in P_i \text{ et } t \in T_i, \\ 0 & \text{sinon.} \end{cases}$$

$$Post(p,t) = \begin{cases} Post_i(p,t) & \text{si } p \in P_i \text{ et } t \in T_i, \\ 0 & \text{sinon.} \end{cases}$$

4) $M0$, le marquage initial défini par $M0(p) = M0_i(p)$ si $p \in P_i$.

A titre d'exemple, supposons un système constitué de 6 robots et de 6 points de rendez-vous. En ordonnant les points de rendez-vous de chacun des ces robots, nous obtenons le tableau suivant :

Robots	r_1	r_2	r_3	r_4	r_5	r_6
Points rendez vous	1 3 5	2 4 6	1 2	3 4	5	6

Dans la figure 2, nous présentons le réseau de Petri global correspondant à ce système. Afin de modéliser la synchronisation entre deux robots par rendez-vous, nous utilisons la solution de fusion des transitions. Il est tout à fait intéressant de remarquer que ce graphe appartient à la famille des *graphes d'événements* dans lesquels les transitions ne sont jamais en conflit, puisqu'une place est entrée (et sortie) d'une seule transition. Dans les réseaux de Petri, l'absence de blocage correspond à la vivacité dont la définition est l'existence d'une séquence de franchissement à partir du marquage initial comportant exactement une occurrences de toutes les transitions. Autrement dit, quelque soit l'état du système, toute transition est potentiellement franchissable dans le futur. Pour cette famille de graphes, de nombreuses propriétés comportementales sont caractérisées par des conditions structurelles. Cependant, nous nous limiterons à un seul lemme. Nous rappelons sa preuve, puisqu'elle contient des éléments que nous utiliserons ultérieurement pour la preuve de l'autostabilisation.

Lemme 2.1 (VIVACITÉ D'UN GRAPHE D'ÉVÉNEMENTS).– *Si N est un graphe d'événements tel que tout circuit élémentaire est initialement marqué, alors N est vivant*

Preuve : Remarquons, tout d'abord, que le nombre de jetons d'un circuit élémentaire dans un graphe d'événements est invariant puisqu'il ne peut y avoir de transition entrante ni sortante.

Supposons que tous les circuits sont initialement marqués, alors pour tout marquage M accessible, ils le sont aussi, d'après la remarque précédente. Pour ce marquage M, nous définissons la relation binaire t *aide*$_M$ t' si et seulement s'il existe une place non marquée de sortie de t et d'entrée de t', et la relation t *précède*$_M$ t' comme la fermeture réflexive et transitive de *aide*$_M$. Montrons que *précède*$_M$ est une relation d'ordre partiel. Si tel n'est pas le cas, nous avons deux transitions t et t' telles que t *précède*$_M$ t' et t' *précède*$_M$ t. D'après la définition de *précède*$_M$, cela signifie qu'il existe des chemins de t à t' et de t' à t où aucune place n'est marquée. En les composant, on obtient un circuit dont on peut extraire un circuit élémentaire non marqué, ce qui est impossible.

Toute relation d'ordre partiel sur un ensemble fini peut s'étendre en (au moins) une relation d'ordre total. Soit $t_1, \ldots, t_n$ la liste ordonnée des transitions. Nous affirmons que $t_1 \cdot \ldots \cdot t_n$ est une séquence de franchissement à partir de M. En effet, t_1 est franchissable, puisque toutes ses places entrées sont marquées. Par induction, le franchissement de la séquence $t_1 \cdot \ldots \cdot t_i$ conduit à un nouveau marquage M'. Toutes les places entrées de t_{i+1} sont marquées dans M' soit parce qu'elles le sont dans M, soit comme conséquence du franchissement d'une certaine transition t_j tel que $j \leq i$. Donc, la séquence de franchissement peut être étendue jusqu'à t_{i+1}. Par conséquent, le réseau est vivant. □

La proposition suivante montre la correction de l'algorithme. Sa preuve, découle presque directement du lemme précédent.

Proposition 2.2 *Soit N le graphe représentant l'algorithme pour un réseau donné, alors N est vivant.*

Preuve : Prenons un circuit $\mathcal{C}$ de N. Notons t_k la transition de plus petit identificateur dans $\mathcal{C}$, $p_{(i,j)}$ la place d'entrée de t_k et $t_{k'}$ la transition d'entrée de $p_{(i,j)}$ dans $\mathcal{C}$. Par construction de N, $k = f(i,j)$ et $k' = f(i,(j-1)\,mod\,n_i)$. Ce choix de t_k implique que $k < k'$, alors que f est croissante vis à vis de son second paramètre. Par conséquent, l'unique possibilité pour j est 0. Puisque $p_{(i,0)}$ est initialement marquée, nous avons prouvé que tout circuit de N contient une place initialement marquée. D'après le lemme 2.1 précédent, N est vivant. □

Ce résultat se généralise au cas de rendez-vous n-aire. Ce schéma de rendez-vous mettant en jeu plusieurs robots diminue la flexibilité du système. Pour cela, nous nous restreindrons au schéma de rendez-vous binaire.

3. Algorithme d'ordonnancement autostabilisant

Dans cette section, nous proposons de doter l'algorithme précédent d'un aspect autostabilisant. Pour cela, nous avons besoin des hypothèses additionnelles suivantes :

1) Chaque robot r_i dispose d'un compteur, contenant une valeur de suspension et représenté par la variable $timeout_i$.

2) Le déplacement d'un robot d'un point vers un autre coûte au plus 1ut (unité de temps). Cette hypothèse peut être facilement satisfaite avec un bon choix de l'unité de temps.

3) Chaque robot possède un capteur lui permettant de connaître sa position courante dans le réseau. La valeur de ce capteur est maintenue par la variable $position_i$.

Chaque robot r_i, maintient les variables suivantes :

– $timeout_i$: une variable contenant la valeur d'un délai de suspension, à valeur réelle dans $[0, \ldots, N+1]$.

– $MP_i[0 \ldots n_i - 1]$: un tableau contenant les identifiants de ses points de rendez-vous triés par ordre croissant.

– $position_i$: la position courante du robot r_i, à valeur dans $\{0, \ldots, n_i - 1\} \cup nowhere$. Elle prend soit la valeur $nowhere$ lorsque r_i est en déplacement d'un point de rendez-vous à un autre, soit le numéro (identifiant) du point de rendez-vous sur lequel il se trouve.

Le comportement d'un robot est événementiel : l'occurrence d'un événement déclenche l'exécution d'un code dépendant de l'état courant du robot. Dans notre cas,

deux événements sont possibles : l'*expiration du timeout* et la *détection de l'homologue*. Ce dernier exige la présence des deux robots concernés au point de rendez-vous. Nous supposons que l'arrivée d'un robot à un point de rendez-vous ne constitue pas un événement. Puisqu'un robot réarme son timeout de 1u.t avant de se diriger vers un nouveau point de rendez-vous, son timeout expirera après son arrivée à ce point ce qui lui permet d'exécuter les actions correspondantes à son arrivée. Le point critique dans ce mécanisme est que le déplacement entre deux points de rendez-vous dure exactement 1u.t. L'activité d'un robot r_i vis à vis des événements est gérée par une variable $status_i$ dont la valeur est soit $moving$, soit $waiting$. Quand un robot est dans un état actif ($moving$), il ne peut ni détecter son homologue, ni être détecter par ce dernier. Autrement dit, l'arrivée d'un robot r_i sur un point de rendez-vous, même si son homologue y est déjà, ne déclenche pas automatiquement l'événement *détection de l'homologue*. La phase d'échange de données n'est engagée qu'après expiration du timeout de r_i.

Une description formelle de cet algorithme est donnée dans la figure 3. Le robot courant est r_i. L'algorithme est décrit suivant les quatres activités d'un robot : $SYNC$, $WAIT$, $RECOVER$ et $MISS$. $SYNC$ et $WAIT$ correspondent aux actions du protocole originel. Afin de savoir si l'expiration du timeout d'un robot correspond à son arrivée à un point de rendez-vous, nous utilisons la variable $status_i$. En effet, cette dernière prend la valeur $moving$ quand le robot se dirige vers un nouveau point de rendrez-vous, et la valeur $waiting$ quand son timeout expire dés son arrivée à un point de rendez-vous. Cependant, l'action $WAIT$ est différente de celle de l'algorithme originel, puisque r_i arme son timeout de $(N + 1)$u.t. L'expiration du timeout d'un robot r_i dès son arrivée à un point de rendez-vous, déclenche l'exécution de l'action $WAIT$, même si son homologue y est déjà. Cette action changera le statut de r_i à $waiting$ permettant ainsi, à r_i et à son homologue, d'exécuter simultanément leurs actions $SYNC$ respectives.

L'occurrence d'une panne transitoire déclenche l'exécution de $RECOVER$ qui est la première action exécutée par le robot fautif. Ce cas de figure n'est possible que si le robot au moment de la panne se trouve sur la route, au passage d'un point de rendez-vous vers un autre. Alors, il arme son timeout de 1u.t. et se dirige vers son premier point de rendrez-vous.

L'action qui permettra la stabilisation de l'algorithme est $MISS$. Elle est exécutée soit initialement, soit à l'expiration du timeout d'un robot pendant qu'il attend son homologue. Dans ce cas, le robot r_i réarme son timeout de 1u.t. et choisit soit de rester sur place, soit de se diriger vers son premier point de rendez-vous. Ce choix est généré aléatoirement par la fonction Uniform-choice qui suite à son appel, affecte à son unique paramètre l'une des valeurs de l'ensemble $\{0, 1\}$.

Une execution de cet algorithme se présente sous la forme d'une séquence infinie $\{t_n, A_n\}_{n\in\mathbb{N}}$ où $\{t_n\}$ est une suite strictement croissante et chaque A_n est un ensemble non vide d'actions déclenchées à l'instant t_n (au plus deux actions par robot, au cas où il exécute $WAIT$ et tout de suite après $SYNC$). Avec cette formalisation, nous définissons une exécution stabilisante.

Constantes : $N, n_i, MP_i[0 \ldots n_i - 1]$;
Timer **:** $timeout_i \in [0 \ldots N + 1]$;
Sensor **:** $position_i \in \{0, \ldots, n_i - 1\} \cup \{nowhere\}$;
Variables **:** $status_i \in \{waiting, moving\}$;
$choice_i \in \{0, 1\}$;

DÉTECTION DE L'HOMOLOGUE // $SYNC$
// à l'arrivée de l'un des deux robots concernés, r_i ou son homologue, alors que
// l'autre l'attend
// Nécessairement $status_i$ est waiting
Échange des messages ;
Réarme($timeout_i$,1) ;
$status_i = moving$;
Aller à $MP_i[(position_i + 1) \bmod n_i]$;

EXPIRATION DU TIMEOUT
Si $(position_i != nowhere)$ **Et** $(status_i == moving)$ **Alors** // $WAIT$
// r_i vient d'arriver au point de rendez-vous
Réarme($timeout_i$,$N + 1$) ;
$status_i = waiting$;
Finsi

Si $(position_i == nowhere)$ **Alors** // $RECOVER$
// recouvrement d'une panne survenue alors que r_i était entre deux points de
// rendez-vous
Réarme($timeout_i$,1) ;
$status_i = moving$;
Aller à $MP_i[0]$;
Finsi

Si $(position_i != nowhere)$ **Et** $(status_i == waiting)$ **Alors** // $MISS$
// expiration du timeout de r_i pendant qu'il attend son homologue
Uniform-Choice($choice_i$) ;
Case ($choice_i$)
0 : Réarme($timeout_i$,1) ;
1 : Réarme($timeout_i$,1) ;
$status_i = moving$;
Aller à $MP_i[0]$;
Fincase
Finsi

Figure 3. *Programme du robot r_i*

Définition 3.1 *Une exécution* $\{t_n, A_n\}_{n\in\mathbb{N}}$ *de l'algorithme est stabilisante si le nombre d'occurrences des actions* $RECOVER$ *et* $MISS$ *est fini.*

En d'autres termes, après un temps fini l'algorithme aura le même comportement que l'algorithme originel. Rappelons que l'action $RECOVER$ n'est exécutée qu'une seule fois au plus par chaque robot. La section suivante est consacrée à la description informelle de la proposition suivante :

Proposition 3.2 *Partant d'un état quelconque, la probabilité qu'une exécution se stabilise est égale à* 1.

4. Preuve informelle

Nous nous restreindrons sans perte de généralité à l'état initial obtenu après que chaque robot ait exécuté au moins une action. Ceci nous évite de prendre en compte l'action $RECOVER$.

Nous supposons d'abord que le temps d'exécution du code est instantané et négligeable par rapport à celui du déplacement des robots. Ainsi, la seule source de non déterminisme dans notre algorithme est le choix uniforme de l'action $MISS$, puisque tout déplacement d'un point vers un autre prend exactement 1 u.t. Par conséquent, la sémantique probabiliste de notre algorithme est définie par une chaîne de Markov que nous décrivons dans la suite.

L'état d'un robot est défini par son statut, l'identifiant de son point de rendez-vous courant et par son timeout divisé en une valeur entière et une valeur résiduelle comprise entre 0 et 1. De la sorte, un état de la chaîne de Markov est le produit des état locaux de tous les robots. Notons, que l'ensemble des états est infini puisque le domaine de valeurs résiduelles est un intervalle de $\mathbb{R}$. Si nous extrayons ces valeurs résiduelles, nous obtenons une relation d'équivalence entre les états basée sur les positions relatives de ces valeurs. Ainsi, nous réduisons notre chaîne à une chaîne de Markov finie respectant les conditions d'agrégation forte.

Puis, nous donnons une condition suffisante de stabilité sur les états du système. Dans toute exécution partante d'un état satisfaisant ladite condition, aucune occurrence d'action $MISS$ n'est possible. A cette fin, nous remarquons que si un marquage M est sans blocage, alors la relation $aide_M$ introduite dans le lemme 2.1 définit un graphe sans cycles. La longueur d'un chemin étant le nombre d'arcs le constituant, nous définissons $level_M(t)$ comme la longueur du plus long chemin d'extrémité t. La condition suffisante s'exprime alors ainsi : un état e est stable si son marquage $M(e)$ est sans blocage et si pour tout robot dont le statut est $waiting$ la valeur entière de son timeout est supérieure à $level_M(t)$.

Nous montrons ensuite qu'à partir d'une classe d'états initiaux quelconque, il existe un chemin allant de cette classe vers la classe d'états stables dans la chaîne de Markov agrégée (la relation d'équivalence étant plus fine que la relation de stabi-

lité). Le seul point non trivial est que pour deux états équivalents quelconques s et s', s est stable si et seulement si s' est stable puisque la définition de la stabilité ne prend pas en compte les valeurs résiduelles. Par agrégation, tout chemin dont l'état final est stable correspond, dans la chaîne finie agrégée, à un chemin dont l'état final est l'ensemble des états stables.

Pour trouver ce chemin particulier dans la chaîne de Markov, nous fixons l'issue du choix aléatoire à chaque exécution de l'action $MISS$. Supposons que l'état initial, noté e, ne soit pas stable. Nous examinons, selon le marquage de e, les deux cas possibles : soit $M(e)$ est sans blocage, soit $M(e)$ est avec blocage. Dans le premier cas, seules les contraintes temporelles peuvent être construites à l'expiration d'un timeout. Nous imposons au robot de rester sur place. Nous prouvons qu'une fois la contrainte de temps est satisfaite chez un robot, elle le reste tout le temps. Ainsi, l'état atteint après que chacun des robots ait exécuté au moins une action $SYNC$, est stable. Dans le second cas, nous simulons le comportement de l'algorithme originel jusqu'au point où chaque robot se trouve seul sur un point de rendez-vous et qu'il ait exécuté au moins une fois l'action $MISS$. A cet instant, tous les timeouts ont des valeurs inférieure ou égale à 1 et nous imposons à chaque robot de choisir la seconde alternative durant l'exécution de son action $MISS$. Toutes ses actions s'exécuteront en moins de 1u.t. Après la dernière exécution d'une action $MISS$, chacun des robots est sur le chemin de son premier point de rendez-vous. Dans un tel état, noté e', le marquage $M(e')$ correspond au marquage initial du réseau N de l'algorithme originel. Par conséquent, $M(e')$ est sans blocage et la suite du chemin est complétée par le chemin construit dans le cas précédent. □

Une description formelle de cette preuve apparaît dans [HAD 03].

5. Routage autostabilisant

Maintenant que nous avons produit un ordonnancement des déplacements des robots, nous nous intéressons au routage dans ce système. Nous décrivons un algorithme autostabilisant responsable de la génération et de la maintenance des tables de routage des robots. La table de routage d'un robot doit contenir, pour être la plus précise possible, une entrée pour tous les couples point de rendez-vous du robot et la destination. En effet, le chemin le plus court vers une destination dépend de la position du robot au moment où il décide d'émettre un message. L'algorithme que nous proposons construit, à partir d'un point de rendez-vous fixé d'un robot, les plus courts chemins à l'ensemble des points de rendez-vous des autres robots. Une entrée dans la table de routage d'un robot est indicée par un couple (point de rendez-vous, identité du robot destination). Cette entrée indique au robot lorsqu'il arrive au point de rendez-vous correspondant s'il doit transmettre, ou non, à son homologue les messages de la destination correspondante. Plus précisément, une entrée dans la table de routage d'un robot est constituée de 4 champs : le premier est la position du robot émetteur du message, le second est l'identité du robot destination, le troisième est la distance minimale séparant ce dernier de la position contenue dans le premier champs. Le qua-

trième champs est le point de rendez-vous suivant sur le plus court chemin vers la destination. A titre d'exemple, si l'on considère le réseau de la figure 2, les entrées de la table de routage de r_1 vers la destination r_6 sont les suivantes :

Source	Destination	Distance	Suivant
$r_1.1$	r_6	6	$r_3.1$
$r_1.3$	r_6	5	$r_4.3$
$r_1.5$	r_6	7	$r_1.1$

Dans cette table, si le robot r_1 désire envoyer un message au robot r_6, il choisira de l'envoyer soit via r_3 s'il est aux points de rendez-vous 1 ou 5, soit via r_4 s'il est au point de rendez-vous 3. Remarquons que l'algorithme trouve le plus court chemin entre les points de rendez-vous d'un robot r_i et l'un quelconque du robot destination.

Pour effectuer le calcul des distances des plus courts chemins, nous construisons un graphe de communication à partir des ordonnancements. Puisque chaque point de rendez-vous est partagé par deux robots, nous doublons les points de rendez-vous pour constituer les noeuds du graphe. Ainsi, un noeud de ce graphe est un couple (r_i, mp) dont le premier élément est l'identité d'un robot et le second élément est l'identifiant d'un de ses points de rendez-vous. Dans ce graphe orienté, deux types d'arcs existent, le premier type d'arc relie le noeud (r_i, mp) au noeud (r_i, mp') si et seulement s'il existe un j tel que $mp = MP_i[j]$ et $mp' = MP[(j+1) \bmod n_i]$. Le second type d'arc relie le noeud (r_i, mp) au noeud (r_j, mp). Ce dédoublement des points de rendez-vous du système, augmente le nombre des liens du graphe de communication. Ainsi, un arc du premier type représentant le déplacement d'un point de rendez-vous à un autre, augmente la distance de 1. Un arc du second type représentant l'échange de message entre un robot r_i et son homologue r_j, augmente aussi la distance de 1. La valeur de cette distance est virtuelle puisqu'en réalité l'échange de message sur un point de rendez-vous est instantané.

Pour simplifier la présentation, nous nous restreignons à une destination particulière, la généralisation étant immediate. Nous appellerons r_0, le robot à atteindre. En plus des constantes et des variables précédentes, chaque robot possède une identité unique r_i placée dans sa mémoire incorruptible et les nouvelles variables suivantes nécessaires pour la construction et le maintien de sa table de routage :

– $distance_i[0 \ldots n_i - 1]$: un élément $distance_i[j]$ de ce tableau est l'estimation courante de r_i de la distance minimale séparant son point de rendez-vous $MP_i[j]$ de r_0. L'intervalle de valeur de cette variable est de $[0 \ldots 2N - 1]$, où N est le nombre total des points de rendez-vous dans le système.

– $sortie_i[0 \ldots n_i - 1]$: tableau de booléens indicé par les points de rendez-vous de r_i. Quand $sortie_i[j]$ est à $vrai$ et que r_i arrive au point de rendez-vous, il passera les messages à destination de r_0 à son homologue via ce point de rendez-vous commun. Notons qu'un robot peut avoir plusieurs attribues $sortie$ positionnées à vrai.

Dans la suite, nous utilisons la technique de composition équitable introduite par Dolev et Herman [DOL 99]. L'idée consiste à composer deux algorithmes autostabi-

Variables
$distance_i[1 \ldots n_i] \in [0 \ldots 2N]$;
$sortie_i[1 \ldots n_i]$;

RÉPÉTER PÉRIODIQUEMENT
Si $(r_i == r_0)$ **Alors**
Pour $j = 0$ **à** $n_0 - 1$ **faire** $distance_0[j] = 0$;
$sortie_0[j] = faux$;
Finpour

Sinon
Sur réception de $< d >$ au point de rendez-vous $MP_i[j]$ **faire**
$d = d + 1$;
$d' = 2N$;
Pour $k = 1$ **à** $n_i - 1$ **faire**
Si $(sortie_i[(j + k) \bmod n_i] == vrai)$ **et** $(distance_i[(j + k) \bmod n_i] < d' - k)$
Alors $d' = distance_i[(j + k) \mod n_i] + k$;
Finsi
Finpour
Si $(d' < d)$ **Alors**
$distance_i[j] := d'$;
$sortie_i[j] := faux$;
Sinon
$distance_i[j] := d$;
$sortie_i[j] := vrai$;
Finsi
Finsi

Figure 4. *Programme d'un robot* r_i

lisants, l'un d'eux utilisant le comportement stabilisé de l'autre comme postulat de départ. Nous appliquons cette technique, aux algorithmes d'ordonnancement et de routage. Chaque robot exécute en parallèle, une action de chacun de ces deux algorithmes. L'algorithme de routage, suppose que celui d'ordonnancement s'est stabilisé et que chaque visite à un point de rendez-vous aboutit à une communication point-à-point durant laquelle les deux robots concernés échangent leur estimation de la distance les séparant du robot destination. Dans le cas contraire, si l'algorithme d'ordonnancement n'est pas stable alors l'algorithme de routage ne garantit pas la correction des entrées des tables de routages.

Le code de l'algorithme est donné dans la figure 4. Si ce code est exécuté par le robot destination r_0, alors ce dernier fixe les distances de tous ses points de rendez-vous à 0 et marque toutes ces variables $sortie$ à $faux$. Sinon, à la réception d'une distance d sur un point de rendez-vous $MP_i[j]$, un robot normal r_i détermine la plus courte distance en passant par un autre point de rendez-vous $MP_i[k]$ dont la variable $sortie_i[k]$ est positionnée à $vrai$. Si cette plus courte distance est meilleure que celle proposée par son homologue, il l'accepte et positionne sa variable $sortie_i[j]$ à $faux$. Sinon, il accepte la distance de son homologue et positionne sa variable $sortie_i[j]$ à $vrai$.

6. Preuve de stabilisation

Rappelons que pour calculer les distances des plus courts chemins, nous avons construit un graphe $\mathcal{G}$ de communication dont les noeuds sont des couples d'éléments (r_i, mp). Dans la suite, nous définissons un sous-graphe $\mathcal{SG}$ comme un graphe dont les noeuds sont les mêmes que ceux de $\mathcal{G}$ et possédant les deux types d'arcs, ceux qui relient un noeud (r_i, mp) à un noeud (r_i, mp') si et seulement s'il existe un j tel que $mp = MP_i[j]$ et $mp' = MP[(j+1) \bmod n_i]$ et ceux qui relient un noeud (r_i, mp) à un noeud (r_j, mp) si et seulement s'il existe un j tel que $mp = MP_i[j]$ et $sortie_i[j] = vrai$. Dans cette section, nous prouvons que le sous-graphe orienté se stabilise en formant une forêt d'arborescences dont les racines sont les points de rendez-vous du robot destination. Dans la suite, nous formalisons cet argument pour prouver la correction de l'algorithme. La structure d'une arborescence est déduite des variables $distance_i[j]$ et $sortie_i[j]$ pour tout les noeuds $(r_i, MP_i[j])$ appartenant à l'arborescence.

Nous introduisons quelques concepts associés à une exécution. Une séquence d'exécution se divise en rondes successives. Chaque ronde débute à la fin de la ronde précédente et se termine lorsque tous les robots ont réalisé au moins un tour complet de leurs points de rendez-vous. La première ronde débute après que l'algorithme d'ordonnancement précédent soit stabilisé et que le robot destination r_0 ait fixé les distances de tous ses points de rendez-vous à 0.

Étant donné une configuration, nous appelons "valeur fausse" une distance dans une variable qui ne correspond pas à la distance réelle. La $ppvf$ correspond à la plus petite valeur fausse de la configuration. S'il n'existe pas de valeur fausse nous posons $ppvf = \infty$. Le lemme suivant établit la propriété clef de cet algorithme.

Lemme 6.1 *Pour tout $k \geq 0$ et pour toute configuration qui suit les premières k rondes, nous avons :*

1- $ppvf \geq k$

2- l'ensemble des noeuds à distance inférieure ou égale à k est organisé en une forêt d'arborescence des plus courts chemins.

Preuve : Notons $ppvf^k$, la $ppvf$ à la fin de la $k^{ième}$ ronde et $ppvf^0$ la $ppvf$ initiale. Nous allons prouver le lemme par récurrence sur k.

Pour $k = 1$, d'une part les distances dans toutes les variables sont strictement positives, et par conséquent, $ppvf^0 \geq 0$. D'autre part, avant la première ronde, le robot destination r_0 a toutes ses variables distances, et sorties correctement positionnées et ne changerons pas au cours de ladite ronde. Par conséquent, toutes les points de rendez-vous du robot destination constituent chacun la racine d'une arborescence.

Supposons que les propriétés sont valides pour la $k^{ième}$ ronde et montrons qu'elles le restent pour la $(k+1)^{ième}$ ronde. En effet, au cours de la $(k+1)^{ième}$ ronde, la distance d'un noeud $(r_i, MP_i[j])$ lorsqu'elle est recalculée est soit exacte, soit supérieure ou égale à $ppvf^k + 1$. Ceci achève la preuve du premier point.

Supposons la deuxième propriété du lemme établie à la $k^{ième}$ ronde et examinons la $(k+1)^{ième}$ ronde. Au cours de cette ronde, la $ppvf$ est supérieure ou égale à k. Donc les noeuds à distances inférieure ou égale à (k conservent leurs attribues $distance$ et $sortie$ inchangés. Un noeud $(r_i, MP_i[j])$ voisin d'un noeud de distance égale à k, recalcule sa distance au cours de cette ronde qui prendra la valeur minimale entre $d+1$ et la plus courte distance en ne passant que par des arcs du premier type $distance_i[(j+x) \bmod n_i]+x$. Cette dernière est soit exacte et correspond à un noeud à distance k dans le graphe de communication, soit une valeur fausse $\geq k$. Dans les deux cas, $(r_i, MP_i[j])$ évaluera sa distance à $k+1$ et choisira son père d'une manière correcte rejoignant l'une des arborescences de la forêt. □

Le corollaire suivant découle immédiatement du lemme précédent.

Corollaire 6.2 *Au bout de $2N$ ronde, le routage est stabilisé dans tout le système*

7. Conclusion

Dans cet article, nous avons étendu le domaine d'application de l'autostabilisation, jusque là réservée aux systèmes distribués, au systèmes de robots mobiles. Dans un premier temps, nous avons présenté un algorithme d'ordonnancement autostabilisant sous un démon probabiliste, basé sur une stratégie de visites des points de rendez-vous assurant qu'après la phase de stabilisation, chaque visite aboutit à une communication. Notre particularité est d'avoir utilisé les réseaux de Petri comme modèle formel pour vérifier la correction de l'algorithme.

Dans un deuxième temps, nous avons présenté un deuxième algorithme autostabilisant, basé sur le premier, qui construit à partir d'un point de rendez-vous fixé d'un robot, les plus courts chemins à l'ensemble des points de rendez-vous des autres robots.

Notre recherche se focalise actuellement sur la transformation de l'algorithme d'ordonnancement en un algorithme autostabilisant sous un démon quelconque et la

réalisation d'un algorithme autostabilisant permettant l'ajout dynamique d'un robot au système.

8. Bibliographie

[BRA 02] BRACKA P., MIDONNET S., ROUSSEL G., « Routage dans un réseau de robots », *Quatrièmes Rencontres Francophones sur les aspects Algorithmiques des Télécommunications AlgoTel*, Mèze, France, mai2002, p. 17-24.

[CAO 97] CAO Y. U., FUKUNAGA A., KAHNG A., « Cooperative Mobile Robotics : Antecedents and Directions », *Autonomous Robots*, vol. 4, March 1997, p. 7-23.

[DEF 02] DEFAGO X., KONAGAYA A., « Circle Formation for Oblivious Anonymous Mobile Robots with No Common Sens of Orientation », *Workshop on self stabilizing, POMC'02*, 2002, p. 97-104.

[DIJ 74] DIJKSTRA E., « Self-stabilizing systems in spite of distributed control », *Communications of the ACM*, vol. 17, n° 11, 1974, p. 643–644.

[DOL 99] DOLEV S., HERMAN T., « Parallel composition of stabilizing algorithms », *WSS'99*, 1999, p. 25-32.

[DOL 00] DOLEV S., *Self-stabilization*, MIT, 2000.

[FEL 68] FELLER W., *An introduction to probability theory and its applications*, vol. I, John Wiley & Sons, Paris, 1968, third edition.

[HAD 03] HADDAD J. E., HADDAD S., « Self-Stabilizing Scheduling Algorithm for Cooperating Robots », *ACS/IEEE International Conference on Computer Systems and Applications, AICCSA'03*, Tunisia, July, 2003.

[HU 98] HU H., KELLY I., KEATING D., VINAGRE D., « Coodination of multiple mobile robots via communication », *Proceedings of SPIE. Mobile Robots XIII and Intelligent Transportation Systems*, Boston, Massachusetts, Novembre 1998, p. 94–103.

[KEM 60] KEMENY J., SNELL J., *Finite Markov Chains*, Van Nostrand, Princeton, NJ, 1960.

[LUZ 00] LUZEAUX D., « Autonomous small robots for military applications », *Conference on Unnamed Groud Vehicle Technology*, USA, April 2000.

[PAR 97] PARK V., CORSON M., « A Highly Adaptive Distributed routing Algorithm for Mobile Wireless Networks », *Proceedings IEEE INFOCOM,The Conference on Computer Communications, Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, Japan, April 1997, p. 1405–1413.

[PRE 01] PRENCIPE G., « Corda : Distributed Coordination of a Set of Autonomous Mobile Robots », *European Research Seminar on Advances in Distributed Systems, Ersads*, Italy May 2001.

[REI 85] REISIG W., *Petri Nets : an Introduction*, Springer Verlag, 1985.

[SCH 93] SCHNEIDER M., « Self-Stabilization », *ACM Symposium Computing Surveys*, vol. 25, 1993, p. 45–67.

[SEL 00] SELLEM P., LUZEAUX D., « Répartition de la perception dans un système distribué de robots autonomes », *Journées Francophones d'Intelligence Artificielle Distribuée et Système multi-agents*, France, Octobre 2000.