résumé de l'article: Optimal probalistic ring exploration by semi-synchronous oblivious robot

Stéphane Devismes, Frank Petit, and Sébastien Tixieul

1 Introduction

Cet article s'intéresse au modèle de Suzuki et Yamashita pour le problème de l'exploration avec stop d'un anneau anonyme et non orienté par des robots autonomes et en abordera 2 points en particulier. Dans un premier temps, on s'intéressera à l'impossibilité de résoudre ce problème sous certaines conditions. Puis on détaillera un algorithme permettant de réduire de façon significative le nombre de robots nécessaires à la résolution de ce problème, ainsi que la preuve de cet algorithme.

L'exploration consiste à visiter toutes les positions au moins une fois et le stop correspond au fait qu'une fois certain que l'exploration a été faite, aucun robot ne bougera.

Antérieurement, les meilleurs résultats proposaient un algorithme déterministe qui résolvait le problème de l'exploration dans un modèle asynchrone avec un nombre de robots relatif à la taille de l'anneau. Pour passer à un meilleur algorithme, on affaiblit les contraintes sur le modèle (d'asynchrone à semi-synchrone) et sur le problème en lui-même (de l'exploration déterministe à l'exploration probabiliste).

L'algorithme présenté ici lui permet de résoudre le problème de l'exploration avec un nombre de robots $k = 4$, et en enlevant la contrainte que k et n doivent être premier entre eux. Il a aussi comme propriété d'utiliser un tirage aléatoire quand il est bloqué à cause d'une symétrie, mais c'est un algorithme qui termine presque sûrement.

suivant le modèle de Suzuki et Yamashita, les robots sont tous identiques, ne peuvent pas communiquer entre eux et suivent des chemins contraints. Ici ils sont sur un anneau. Les robots sont oublieux ce qui signifie qu'ils ne gardent pas en mémoire leurs actions d'une étape à l'autre.

L'article présente aussi des résultats d'impossibilité pour conforter son choix dans les contraintes : si k et n ne sont pas premiers entre eux, alors il n'est pas possible dans un modèle synchrone de finir une exploration, même avec l'aide d'un algorithme probabiliste. Si $k \leq 3$ alors il est également impossible de finir une exploration et ce dans le cas semi-synchrone.

2 Définitions

L'espace considéré dans cet article est un anneau à n positions, et on ne peut passer d'un nœud qu'à un nœud voisin. L'anneau est non orienté et anonyme, ainsi un robot ne peut pas différencier sa position d'une autre ni si un de ses voisins est à sa droite ou à sa gauche.

Ci-après, il est donné à titre indicatif un identifiant à chaque nœuds pour des questions de représentation et de clarté.

Une configuration pour un nœud u_i donné est représentée par une suite de nombres composée des nombres de robots sur chaque nœud, en partant de u_i et en y revenant dans chacun des sens.

Ce nombre de robot est appelé la multiplicité du nœud. Si plus d'un robot est sur un nœud, on appelle cela une tour. Deux configurations sont dites équivalentes si ce sont les mêmes à une rotation ou une symétrie près. Par exemple, si x_i représente la multiplicité du nœud u_i, les configurations <$x_0, x_1, ..., x_n$> et <$x_i, x_{i+1}, ..., x_{i+n-1}$> sont équivalentes par une rotation de i, et les configurations <$x_0, x_1, ..., x_n$> et <$x_n, x_{n-1}, ..., x_0$> le sont par symmétrie.

Les robots se déplacent (de manière probabiliste ou déterministe) en fonction d'une destination qu'ils calculent de manière autonome grâce à une perception de l'espace faite précédemment. Ce cycle, défini comme le cycle `Look-Compute-Move`, répété indéfiniment est le programme local du robot, et l'ensemble de ces programmes est appellé un `protocole`.

Il existe plusieurs façons d'entrelacer les différentes phases des cycles entre des robots. Pour les différencier, il existe plusieurs modèles. On en considèrera deux :

- Le modèle `semi-synchrone` décrit les cycles `Look-Compute-Move` comme atomiques : entre deux moments où un robot peut commencer un cycle, tout autre robot activé aura effectué son cycle en entier.
- Le modèle `asynchrone`, lui, permet aux autres robots d'effectuer un nombre fini mais non borné d'actions entre deux actions du cycle d'un robot, mais l'action `Look` est instantanée, et l'action `Move` est atomique.

Il est possible d'instancier des contraintes sur les exécutions pour que certains choix soient pris. on appelle l'ensemble de ces contraintes un démon. À noter que les algorithmes doivent fonctionner pour tout *démon*.

L'article définit le problème de l'exploration de la manière suivante :

Un protocole P résout le problème de l'exploration de façon déterministe (resp. probabiliste) si et seulement si pour tout calcul c de P partant d'une configuration sans tour, (1) c termine en un temps fini (resp. avec une probabilité de 1); (2) chaque nœud est visité par au moins un robot pendant c.

Les configurations initiales sont donc celles sans tours. On exclut donc les cas où $k > n$ car il y a forcément une tour, et si $k = n$ la configuration initiale permet à elle seule de faire l'exploration. Dorénavant, $k < n$.

3 Impossibilité

L'article présente dans un premier temps deux résultats d'impossibilité.

3.1 L'exploration ne peut être réalisée en asynchrone si k divise n

Si k divise n, il n'existe pas d'algorithmes pouvant résoudre le problème de l'exploration dans le modèle asynchrone. Puisque k divise n, il existe une configuration où tous les robots sont à une distance égale les uns des autres, et le *démon* fait en sorte que tous les robots bougent dans le même sens d'une case, donc on revient à une configuration équivalente à celle de départ, et ce cycle peut se répéter à l'infini. Cette preuve justifie l'utilisation du modèle semi-synchrone par la suite.

3.2 L'exploration ne peut être réalisée en semi-synchrone si $k \leq 3$

Pour montrer cette impossibilité il est pris en compte que toute configuration terminale contient au moins une tour. En effet les robots on besoin d'une configuration où ils ne bougent plus pour résoudre l'exploration avec stop, et il faut donc que cette configuration ne puisse pas être une des configurations initiale. Donc cela élimine les cas où $k = 0$ ou $k = 1$ car on ne peut pas former de tour.

Il existe forcément une séquence de $n - k + 1$ configurations contenant une tour de moins de k robots indistinguables deux à deux pour tout protocole d'exploration. En effet pour être sûr que tout nœud a été exploré, on part d'une configuration où on a k nœuds occupés par des robots, on forme une tour (1 pas au minimum) puis on explore les nœuds non encore explorés ($n - k$ pas). considérons un pas β β' lors de cette phase, si on a $\beta = \beta$', aucun nœud n'est exploré, et sinon, la seule façon d'explorer un nœud de plus de façon certaine est de passer d'une tour dans β à une tour de moins de k robots dans β'.

Dans le cas où $k = 2$, il n'est pas possible de construire une tour de moins de k donc il est impossible d'avoir un algorithme résolvant notre problème d'exploration.

Il reste le cas où $k = 3$: comme il doit y avoir une tour, il n'y a que $\lfloor n/2 \rfloor$ configurations distinctes possibles. de plus, on a montré ci-dessus qu'il devait y avoir au moins $n - k + 1$ configurations distinctes, et en recoupant ces deux contraintes, on en ressort l'inéquation $\lfloor n/2 \rfloor \geq n-k+1$ donc $n \leq 4$.

Il reste donc à étudier le cas où $k = 3$ et $n = 4$. Pour ce cas un ordonnanceur séquentiel ne suffit pas. Il est utilisé un ordonnanceur non-séquentiel et un raisonnement par cas pour prouver l'impossibilité d'un algorithme. Pour chaque cas, soit toute configuration terminale peut être obtenue à partir d'une exécution correcte du protocole qui ne visite pas tous les nœuds, soit le démon peut faire en sorte que le protocole ne termine jamais. À chaque pas, On se retrouverait sur une configuration équivalente à celle juste avant, et donc on recommencerait le même cycle.

4 Résultats positifs

Dorénavant, on prendra $n > 4$ et $k = 4$.

Pour explorer l'anneau, on veut dans un premier temps créer un motif ne faisant pas partie des configurations initiales possibles, puis propager un seul élément de ce motif qui fera le tour de l'anneau. Le motif doit donc contenir une tour pour ne pas appartenir aux configurations initiales. De plus, la création de cette tour ne doit pas être différente pour deux configurations, donc on veut se ramener à une configuration initiale distinguable de toutes les autres et toujours atteignable. Pour ce faire, on définit un segment comme toute suite de nœuds non vide avec deux nœuds vide aux extrémités. Sa taille est le nombre de robots à l'intérieur. Un x-segment est un segment de taille x. La configuration toujours distinguable de toutes les autres est celle où il y a un 4-segment.

L'espace entre deux segments est appeler un trou. Sa taille est le nombre de nœuds vides le composant.

Le motif utilisé, nommé *flèche*, sera le suivant : X-...-_-...-T-X avec X représentant un robot seul, T une tour, _ un nœud vide et ... la possibilité de nœuds vide. si le trou intérieur est composé d'un seul nœud, c'est une flèche primaire, et si ce trou a une taille de $n - 3$, c'est une flèche terminale.

L'algorithme se découpe en 3 étapes : premièrement, partant de toute configuration initiale sans tour, on construit un 4-segment. Puis à partir de ce segment, l'algorithme construit de manière probabiliste une flèche primaire. Pour finir, on parcourt l'anneau en faisant grandir la flèche jusqu'à ce qu'elle soit terminale.

Lors de la première étape, il n'y a jamais de tour construite, donc il n'y a pas de création de flèche. C'est aussi la partie dont l'exactitude est la plus compliquée à prouver. En effet pour la création de la flèche, seuls les robots à l'intérieur du 4-segment peuvent être activés, et ils essayent l'un et l'autre de bouger sur l'autre pour créer une tour, donc ils réussiront cette tâche avec une probabilité de 1. En effet, il y a toujours une probabilité que les deux robots interieurs au 4-segment décident de bouger au même moment, mais il y a une probabilité de 0 que ça soit toujours le cas. Donc on est sûr que la flèche se formera à un moment donné. Quant à la dernière phase, il est facile de prouver qu'elle est correcte.

La partie de l'algorithme correspondant à la création d'un 4-segment est séparée en différents cas en fonction de la taille de l'anneau.

- Si la configuration contient juste un unique segment de taille maximale, on rapproche les robots isolés vers ce segment, et en ne bougeant que les isolés on ne peut pas créer de tour.
- Si $n = 6$ ou $n = 8$, l'algorithme propose de se référer à des automates représentant les évolutions de l'algorithme pour comprendre comment on arrive à une configuration contenant un 4-segment sans avoir créé de tour et en partant de n'importe laquelle des configurations initiales possibles, à des équivalences près. Comme le nombre de possibilités est fini, on prouve qu'on ne crée pas de tour lors de la création du 4-segment.
- Si $n = 7$, si on est voisin du plus grand trou, on bouge vers ce trou. Il n'y a donc pas de tour créée car on bouge vers des nœuds vides.
- Si $n > 8$, si on a une configuration avec deux 2-segment ou quatre robots isolés, l'algorithme arrive éventuellement à passer à une configuration à un 4-segment en regardant une fois encore toutes les possibilités de configuration par rapport à la distance entre les robots. On fait ensuite en sorte de réussir à faire ce 4-segment en ne bougeant que vers des nœuds vides. Donc aucune tour n'est créée.

Ainsi en prouvant qu'on ne crée jamais de tour, on arrive d'une configuration sans tour à une configuration contenant un 4-segment, donc on peut créer une flèche puis explorer l'anneau.

L'algorithme présenté dans l'article permet donc l'exploration avec stop d'un anneau de taille $n \geq 4$ pour un nombre de robots $k = 4$, avec une probabilité de 1.