Eviter un obstacle avec des robots lumineux : synthèse d'algorithmes

Karine Altisen,[1] Anaïs Durand,[2] Pascal Lafourcade[2] et Oussama Nahnah[2]

[1] *Université de Genève*
[2] *Universite Clermont Auvergne, Clermont Auvergne INP, CNRS, LIMOS*

Concevoir des algorithmes distribués pour essaims de robots et prouver leur validité est une tâche complexe, fortement combinatoire et sujette aux erreurs. Les tentatives d'automatisation sont restées limitées. Nous proposons une méthode partiellement automatisée pour synthétiser des algorithmes d'exploration perpétuelle d'une grille finie avec obstacle pour des robots lumineux myopes avec chiralité. Cette approche est illustrée par un outil de génération d'algorithmes et de validation par simulation sur des petites grilles. Les résultats sont généralisés à de plus grandes grilles par preuve.

Mots-clefs : Robots lumineux, exploration perpétuelle, grille finie, obstacle, synthèse d'algorithmes.

1 Introduction

Nous nous intéressons au problème de l'*exploration perpétuelle d'une grille finie 2D* par un essaim de *robots* mobiles autonomes *lumineux*. Ces robots sont équipés de capteurs de visibilité dont la portée est limitée, d'actionneurs de mouvement et de lumières pouvant changer de couleur. Chaque nœud de la grille doit être visité infiniment souvent par au moins un robot. Les capacités des robots sont très faibles. En plus de la portée limitée de leurs capteurs, ils ne disposent que d'un nombre constant de couleurs pour leurs lumières. Ils n'ont aucune autre mémoire persistante ou moyen de communiquer. Ils peuvent seulement observer les positions relatives et les couleurs des robots dans leur voisinage proche. Les robots ne s'accordent pas sur un système de coordonnées commun (en particulier, l'orientation Nord-Sud et Est-Ouest), mais ils partagent une *chiralité commune* : leur système de coordonnées est arbitraire mais l'orientation de ses axes est commune à tous les robots. Enfin, les robots réalisent des *rondes synchrones* et opèrent dans le modèle *Regarder-Calculer-Se déplacer*.[†]

Synthèse pour essaims de robots. Concevoir des algorithmes sous de telles hypothèses est une tâche complexe et sujette aux erreurs. Prouver qu'ils sont corrects est d'autant plus complexe qu'il faut faire face à la combinatoire importante du problème et aux nombreuses hypothèses. Si la conception et la preuve sont généralement réalisées « à la main », des outils d'assistance par ordinateur ont aussi été utilisés.

Un assistant de preuve comme *Rocq* permet de formaliser les preuves d'un algorithme et de certifier leur correction. En particulier, le framework *Pactole* [6] est dédié aux algorithmes pour essaims de robots. Cette approche n'est pas automatique : l'utilisateur doit apporter une preuve complète que l'outil valide ; l'assistant aide seulement à énumérer les cas et éviter les erreurs. De plus, à notre connaissance, l'exploration perpétuelle et le modèle des robots lumineux ne sont pas pris en charge par Pactole.

Des approches basées sur le *model-checking*, comme [3], ou sur la *synthèse de contrôleurs*, comme [4], ont également été proposées. Ces approches nécessitent soit de complètement définir le contexte d'exécution (nombre de robots, nombre de couleurs, topologie du graphe, *etc.*), soit de proposer des solutions paramétriques souvent restreintes pour cause d'indécidabilité. Ces méthodes font également face à l'explosion combinatoire due à la très grande taille de l'espace de recherche, ce qui restreint généralement les résultats à des algorithmes synthétisés pour des petits graphes [4]. De plus, à notre connaissance, dans les environnements discrets, seul le modèle des robots *amnésiques* (sans couleur) a été considéré dans ce domaine ; le modèle des robots lumineux ajoute des couleurs qui augmentent rapidement l'explosion combinatoire.

†. Nous utilisons le modèle de calcul de [5]. Pour plus de détails, voir aussi [2].

Contributions. Pour faire face à ces difficultés, nous proposons une nouvelle approche qui combine des étapes automatiques et « à la main ». Nous illustrons notre méthode en nous intéressant au problème de l'exploration perpétuelle d'une grille finie avec obstacle. Bien que le problème de l'exploration perpétuelle d'une grille finie ait déjà été étudié, y compris pour des robots lumineux synchrones avec chiralité [5], aucune de ces études n'a considéré de grilles avec obstacles. Les obstacles considérés ici sont des trous ou des poteaux que les robots doivent contourner pour pouvoir poursuivre leur exploration.

Notre méthode consiste à étendre des algorithmes existants qui réalisent l'exploration perpétuelle d'une grille, mais sans obstacle. La conception des nouveaux algorithmes est guidée par un scénario fourni par le concepteur qui propose des motifs de mouvements pour les robots *(i)*. Les algorithmes sont ensuite synthétisés automatiquement par un outil *(ii)* et testés par simulation sur des petites grilles *(iii)*. Les résultats de simulation sont ensuite généralisés par une preuve « à la main », par induction sur la taille des grilles *(iv)*. Enfin, notre outil permet d'analyser et de classer les nombreux algorithmes synthétisés selon différents critères *(v)*. Cette méthode est générique et associée à un outil qui automatise les étapes de *ii*, *iii* et *v* et dont le code est disponible ici [1]. Nous l'avons appliquée à deux cas d'étude en étendant deux algorithmes proposés dans [5].

Plan. Dans la suite du papier, nous expliquons les différentes étapes de notre méthode en l'illustrant grâce à nos cas d'étude dans les Sections 2 à 6. Nous concluons en Section 7. Par manque de place, tous les détails de notre méthode et les résultats de nos cas d'étude ne peuvent être présentés ici, mais sont disponibles sur le site de notre outil [1] dans une version longue de notre article [2].

2 Établir un scénario

Notre but est d'étendre un algorithme de base $\mathcal{A}_b$ initialement conçu pour résoudre le problème de l'exploration perpétuelle d'une grille finie *sans obstacle* avec un essaim de robots lumineux, afin que les algorithme générés fonctionnent dans une grille contenant un obstacle[‡]. Pour éviter des situations impossibles (voir [1]), nous supposons que l'obstacle est placé suffisamment loin des bords de la grille, aussi appelés *murs*. Plus précisément, nous considérons les grilles respectant le prédicat suivant : $ObsAuMilieu(i, j, C, \mathcal{L})$ est satisfait dans une grille de taille $C \times \mathcal{L}$ si et seulement si elle ne contient qu'un seul obstacle (trou ou poteau) positionné en (i, j) tel que $v \leq i \leq C + 1 - v$ et $v \leq j \leq \mathcal{L} + 1 - v$ où $v = (n + 1) \times \phi$, n est le nombre de robots et ϕ est la portée des capteurs.

Nous considérons comme cas d'étude deux algorithmes proposés dans [5]. Le premier, noté ici $\mathcal{A}_b^1$, utilise deux robots dont la visibilité est limitée à un pas et trois couleurs. Le second, noté ici $\mathcal{A}_b^2$, utilise également deux robots mais leur visibilité est de deux pas et ils utilisent seulement deux couleurs. Nous voulons synthétiser des nouvelles règles (sans changer les règles de $\mathcal{A}_b^1$ et $\mathcal{A}_b^2$) pour permettre aux robots de contourner l'obstacle et obtenir de nouveaux algorithmes $\mathcal{A}_e$ satisfaisant l'exploration perpétuelle d'une grille finie avec obstacle. Les résultats présentés ici supposent que l'obstacle est un poteau qu'il faut contourner et qui bloque la vue des robots. Nous nous attendons à ce que $\mathcal{A}_e$ se comporte comme $\mathcal{A}_b$ lorsque les robots se trouvent loin de l'obstacle.

Bien sûr, ce problème peut avoir beaucoup trop de solutions pour être résolu automatiquement par un outil. Par exemple, il existe 490 règles différentes pouvant être utilisées pour étendre $\mathcal{A}_b^1$, soit 2^{490} extensions possibles qui doivent être considérées (même si un grand nombre d'entre elles seront éliminées ensuite car incompatibles). Pour réduire l'espace de recherche, nous utilisons un *scénario* défini par l'utilisateur.

Un scénario est composé d'une liste d'*objectifs*, chaque objectif étant une paire (D, F) de *configurations locales* (c'est-à-dire l'état de la grille restreinte aux robots et aux cases qui leur sont visibles). En partant de D, les robots doivent atteindre F en utilisant les règles de $\mathcal{A}_b$ et/ou de nouvelles règles synthétisées. Plusieurs paramètres définis pour chaque objectif permettent de restreindre les possibilités et de rendre le problème traitable automatiquement : *(a)* Une *enveloppe* rectangulaire dont les robots ne peuvent sortir pendant leur trajet de D à F. *(b)* Un nombre de rondes k maximum pour effectuer le trajet. *(c)* En option, des *noeuds obligatoires* (qu'au moins un robot doit visiter entre D et F) ou *interdits* (que les robots ne doivent pas visiter entre D et F) peuvent être définis.

‡. Se limiter à un seul obstacle constitue une première étape pour circonscrire l'explosion combinatoire.

Tous ces paramètres sont choisis par l'utilisateur en fonction des trajectoires attendues. Par exemple, pour $\mathcal{A}_b^1$, nous avons considéré le scénario suivant. Quand les robots passent près de l'obstacle (au-dessus ou en-dessous), ils continuent leur chemin. Quand ils arrivent face à l'obstacle, soit ils l'évitent en se déplaçant sur la ligne suivante, soit ils font demi-tour, en fonction de leurs couleurs. Définir ces objectifs est réalisé de façon itérative grâce à l'assistance de l'outil, par essai-erreur, jusqu'à obtenir des solutions.

3 Synthèse des algorithmes candidats

Une fois le scénario établi, il est utilisé en entrée de notre outil nommé **RoASt** (*Robot Algorithm Synthesizer*) développé en Rust 1.83.0. Son code est disponible ici [1].

À partir des règles de $\mathcal{A}_b$, **RoASt** réalise une phase de pré-calcul pour des raisons d'optimisation. Pendant cette phase, il synthétise toutes les vues possibles (sans murs) pour un robot, puis toutes les règles possibles à partir de ces vues. Bien sûr, les vues et règles invalides sont éliminées. Enfin, les règles obtenues sont combinées entre elles pour former des *règles parallèles* : une configuration locale à laquelle on associe pour chaque robot le mouvement et/ou changement de couleur qu'il réalise dans cette situation (un robot pouvant également ne rien faire). On obtient ainsi 35 vues, 490 règles et 2449 règles parallèles pour $\mathcal{A}_b^1$ (respectivement, 146 vues, 1298 règles et 5892 règles parallèles pour $\mathcal{A}_b^2$).

Puis, à partir du scénario, **RoASt** calcule pour chaque objectif i du scénario, l'ensemble $\mathbb{R}_i$ des ensembles de règles possibles permettant aux robots de remplir l'objectif en respectant ses contraintes (nombre de rondes, enveloppe, *etc.*). Les ensembles $\mathbb{R}_i$ sont ensuite itérativement combinés entre eux : $\mathbb{R}_1$ est combiné avec $\mathbb{R}_2$, puis le résultat est combiné avec $\mathbb{R}_3$, et ainsi de suite. Certaines combinaisons sont invalides (par exemple, les règles utilisées sont incompatibles) et sont donc éliminées pour garantir que les algorithmes obtenus soient valides par construction. Une fois tous les ensembles $\mathbb{R}_i$ combinés, on obtient un ensemble d'algorithmes candidats. Pour $\mathcal{A}_b^1$, nous obtenons un ensemble $\mathbb{C}_1$ de 5 algorithmes candidats en environ deux secondes.[§] Pour $\mathcal{A}_b^2$, c'est un ensemble $\mathbb{C}_2$ de 373 248 algorithmes candidats qui est obtenu, en environ une minute.

4 Validation par simulation

Les algorithmes candidats dans $\mathbb{C}$ synthétisés à l'étape précédente, sont valides par construction. Cependant, il n'est pas certain qu'ils réalisent l'exploration perpétuelle de la grille. Nous cherchons donc à déterminer un sous-ensemble $\mathbb{A} \subseteq \mathbb{C}$ d'algorithmes qui répondent au problème. Cette étape ne pouvant pas être entièrement automatisée à cause de la nature du problème, nous procédons en deux étapes.

La première étape consiste à simuler chaque algorithme candidat sur un ensemble de grilles de petite taille $\mathbf{G}^{\mathtt{sim}}$ et pour un ensemble donné de configurations initiales $\mathcal{I}^{\mathtt{sim}}$. Si l'algorithme échoue à réaliser l'exploration perpétuelle dans au moins une des simulations, il est rejeté. Les résultats obtenus par simulation sont ensuite étendus par preuve, voir Section 5.

Notez que les grilles et configurations initiales considérées pour la simulation dépendent fortement des hypothèses nécessaires pour la preuve. Ainsi, pour $\mathbb{C}_1$, nous avons utilisé l'ensemble $\mathbf{G}_1^{\mathtt{sim}}$ contenant les 27 grilles de tailles 7×7 à 8×9 et 9×8 respectant le prédicat *ObsAuMilieu*. Les configurations initiales $\mathcal{I}_1^{\mathtt{sim}}$ considérées sont toutes les configurations initiales de $\mathcal{A}_b^1$ transformées de façon à ce qu'un des robots voit l'obstacle. Cela représente 12 configurations initiales soit un total de $12 \times 27 = 324$ simulations à réaliser pour chaque algorithme candidat. Il se trouve que les cinq candidats de $\mathbb{C}_1$ passent avec succès cette étape de validation par simulation en moins de 1s.

Cependant, ce procédé peut être très intensif en calcul, notamment avec un grand nombre d'algorithmes candidats comme pour le cas de $\mathcal{A}_b^2$. Pour obtenir des résultats progressivement, nous avons choisi de réaliser les simulations par sous-ensembles $\mathbb{C}^i$ d'algorithmes candidats. Nous avons configuré l'outil pour que ces sous-ensembles contiennent les algorithmes ayant le même niveau d'énergie utilisée pour réaliser les objectifs (un robot consomme une unité d'énergie pour un mouvement ou un changement de couleur). Les algorithmes avec le plus bas niveau d'énergie sont simulés en premier, puis les algorithmes du niveau

§. Toutes les expérimentations ont été réalisées sur un AMD EPYC 7763 64-core @ 256 × 2.45GHz (2TO de RAM) utilisant Debian 12.11 avec Linux 6.1.0-37-amd64.

suivant, et ainsi de suite. L'utilisateur peut ainsi choisir de ne valider par simulation que certains niveaux. Pour $\mathbb{C}_2$, nous avons choisi de ne simuler que les quatre premiers niveaux (soit 9 911 candidats) et avons obtenu 115 algorithmes validés après environ une heure de calcul.

5 Extension des résultats par preuve

Pour garantir que les algorithmes $\mathbb{A}$ retenus après simulation sont corrects, il faut étendre les résultats à toute configuration initiale et aux grilles de plus grande taille que celles simulées. Cette généralisation est réalisée par une preuve « à la main » en deux étapes.

1) Nous montrons d'abord que pour toute grille $\mathcal{G} \in \mathbf{G}^{\mathtt{sim}}$, les algorithmes $\mathcal{A}_e \in \mathbb{A}$ réalisent l'exploration perpétuelle de $\mathcal{G}$ à partir de toutes leurs configurations initiales (et non pas seulement celles de $\mathcal{I}^{\mathtt{sim}}$). Pour $\mathbb{A}_1$ et $\mathbb{A}_2$, cette preuve est réalisée en montrant que si les robots commencent dans une configuration $\gamma \notin \mathcal{I}^{\mathtt{sim}}$, ils finissent par atteindre une configuration de $\mathcal{I}^{\mathtt{sim}}$ à partir de laquelle ils se comporteront comme dans la simulation. La preuve complète est disponible ici [1].

2) Ensuite, nous étendons ces résultats aux grilles de taille plus grande. Cette preuve est réalisée par induction sur la taille de la grille. Les cas de base sont couverts par la simulation et la preuve du point 1). Le pas d'induction consiste ensuite à ajouter des colonnes ou des lignes et à montrer que l'exploration perpétuelle est toujours satisfaite dans la grille plus grande. Notez que pour $\mathbb{A}_1$ et $\mathbb{A}_2$, les colonnes et les lignes doivent être ajoutées par deux, le comportement des robots dépendant de la parité des dimensions de la grille, voir [1].

6 Classification des algorithmes et analyse

Même si par construction de **RoASt**, tous les algorithmes générés ont des ensembles de règles différents à permutation près, le nombre d'algorithmes générés peut être grand. **RoASt** analyse les résultats obtenus lors de la génération et des simulations afin de classifier les algorithmes selon différents critères. Cela permet une meilleure compréhension du résultat. Nous avons ainsi défini quatre *critères*.

Deux algorithmes $\mathcal{A}$ et $\mathcal{A}'$ sont équivalents selon le critère C_i, $i \in \{1, ..., 4\}$ si :

- C_1 – *Nombre de règles :* $\mathcal{A}$ et $\mathcal{A}'$ ont le même nombre de règles.
- C_2 – *Nombre de rondes dans un cycle :* dans chaque simulation réalisée, le nombre de rondes nécessaires pour réaliser un cycle d'exploration (c'est-à-dire, revenir à une configuration précédemment rencontrée après avoir exploré toute la grille) est le même pour $\mathcal{A}$ et $\mathcal{A}'$.
- C_3 – *Énergie dans un cycle :* dans chaque simulation, l'énergie consommée pour réaliser un cycle d'exploration est le même pour les deux algorithmes.
- C_4 – *Chemin dans un cycle :* dans chaque simulation, à tout moment pendant le cycle d'exploration, les robots occupent la même position dans la grille dans les deux algorithmes.

Les résultats et analyses de ces classifications pour les études de cas se trouvent dans [1]. Par exemple, nous avons constaté deux grandes familles d'algorithmes synthétisés pour $\mathcal{A}_b^2$: ceux qui se comportent comme attendu et ceux qui, à partir de certaines configurations initiales, changent la direction de leur exploration (verticalement colonne par colonne au lieu d'horizontalement ligne par ligne, par exemple).

7 Conclusion

L'outil **RoASt** permet de reproduire aisément ces expériences en modifiant la nature de l'obstacle (en trou) ou en changeant le scénario, voir [1]. Finalement, la méthode proposée s'est montrée plutôt efficace pour nos cas d'étude (par exemple, 115 nouveaux algorithmes en un peu plus d'une heure) et ces résultats permettent d'envisager différentes extensions, par exemple : plusieurs obstacles, grilles triangulaires, voire génération complète d'algorithmes (et non extension d'un algorithme existant).

Cette approche repose sur un scénario fourni par l'utilisateur, scénario qui peut grandement impacter les résultats. En effet, une mauvais scénario peut aboutir à la synthèse d'algorithmes qui ne réalisent pas l'exploration perpétuelle (échec lors de la phase de simulation). Les paramètres des objectifs ont aussi un impact important sur les performances : plus on laisse de liberté aux robots, plus le temps de génération sera grand. Dans des travaux futurs, nous souhaiterions étudier la conception automatique des scénarios.

Références

[1] Karine Altisen, Anaïs Durand, Pascal Lafourcade, and Oussama Nahnah. Code de **RoASt** et annexes, 2026. `https://luminousrobots.github.io/RoASt-Docs`.

[2] Karine Altisen, Anaïs Durand, Pascal Lafourcade, and Oussama Nahnah. Synthesizing algorithms to avoid an obstacle with a swarm of robots, 2026. `https://sancy.iut.uca.fr/~durand/files/roast.pdf`.

[3] Béatrice Bérard, Pascal Lafourcade, Laure Millet, Maria Potop-Butucaru, Yann Thierry-Mieg, and Sébastien Tixeuil. Formal verification of mobile robot protocols. *Distributed Comput.*, 29(6) :459–487, 2016.

[4] François Bonnet, Xavier Défago, Franck Petit, Maria Potop-Butucaru, and Sébastien Tixeuil. Discovering and assessing fine-grained metrics in robot networks protocols. In *SRDS*, pages 50–59, 2014.

[5] Quentin Bramas, Pascal Lafourcade, and Stéphane Devismes. Optimal exclusive perpetual grid exploration by luminous myopic opaque robots with common chirality. In *ICDCN*, pages 76–85, 2021.

[6] Projet Pactole. Framework Pactole. `https://pactole.liris.cnrs.fr/`.