Rapport de Projet CoCoMA

Mathieu Berthellemy Élodie Cayez
Marie-Morgane Paumard

10 novembre 2015

Résumé

Ce rapport présente nos réalisations pour le projet de CoCoMA ; nous étayons nos choix de conception d'un SMA cognitif, notamment en détaillant la coordination multi-agents. Nous présentons un SMA décentralisé, qui permet d'obtenir des résultats convenables lors de la simulation. Nous avons doté nos agents d'une préférence pour la fabrication à l'achat car celle-ci est plus complexe et nous permet de confirmer que les comportements sont ceux attendus. Nous avons négligé la résolution de problème subsidiaires tels que la gestion de la capacité, la gestion de la batterie et l'assistance lors de la fabrication, nous contentons de les survoler et d'y apporter des réponses brèves.

1 Introduction

Ce projet, réalisé dans le cadre du cours CoCoMA, a pour objectif de nous donner une expérience du développement d'un système multi-agents. Basé sur le scénario fourni par le *Multi Agent Programming Contest 2015* (MAPC), le projet consiste à concevoir les comportements d'une équipe de quatre agents coursiers se déplaçant dans les rues d'une ville. L'équipe a pour but de maximiser ses recettes en complétant des tâches d'acquisition, de fabrication et de livraison d'objets dans un temps imparti et avant les équipes adverses. Chaque agent dispose de caractéristiques qui lui sont propres, comme sa vitesse, ses chemins autorisés, sa batterie, sa capacité et ses compétences. Les objets peuvent être achetés, fabriqués, donnés, stockés, récupérés, abandonnés et livrés.

Comme conseillé par nos encadrants, nous avons commencé par réaliser un modèle simple mais satisfaisant, que nous avons raffiné au fil de nos expériences. Nous avons distingué deux axes principaux pour l'organisation du travail : conception d'un modèle traduisant les consignes exposées ci-dessus, en optant pour une coordination décentralisée, et optimisation des décisions prises par les agents, par essais successifs.

Nous allons donc organiser ce rapport de la même manière : nous présenterons dans un premier temps notre modèle de base puis nous en soulignerons les aspects principaux, notamment ceux liés à la coordination et au partage de ressource. Dans un second temps, nous détaillerons les améliorations réalisées. La troisième partie propose une évaluation de notre système multi-agent (SMA). Nous terminerons par une conclusion sur notre travail en général, ainsi qu'une ouverture sur de possibles améliorations de notre SMA.

2 Modèle de base

Nous avons jeté notre dévolu sur un algorithme décentralisé, de manière exploiter au mieux les possibilités offertes par le paradigme multi-agent. Au début du programme, les agents initialisent leurs positions (section 2.1). Lorsqu'une tâche est proposée, tous les agents reçoivent un message contenant les type et nombre d'objet à apporter et le lieu de livraison. À chaque tâche proposée, chaque agent calcule s'il est à même de l'effectuer efficacement (détails en section 2.2) et, le cas échéant, quelles sous tâches il s'attribue (détails en section 2.3).

2.1 Initialisation

Comme les propriétés de longitude et latitude ont été désactivées, les agents doivent être dans un emplacement pour pouvoir avoir un notion de leur position. Lors du lancement du programme, les agents sont donc envoyés vers des buts (nous détaillons nos choix en section 4.1). Lorsqu'un agent arrive à sa destination, il envoie à tous les autres qu'il est prêt et attend de recevoir une réponse des trois autres agents. Une fois l'ensemble des réponses reçues, les agents consultent s'ils ont reçu une tâche et la traitent (acceptation et distribution).

Remarques Nous supposons que les agents ne reçoivent qu'une seule tâche à la fois : tant qu'ils n'ont pas déterminé leurs attitudes (accepter et traiter ou refuser la tâche), ils refusent systématiquement toutes les tâches suivantes. Nous supposons que les agents reçoivent les tâches en même temps et dans le même ordre : il n'y a ainsi pas d'inter-blocage.

2.2 Acceptation d'une tâche

Pour qu'une tâche soit acceptée, il est nécessaire que les recettes soient supérieures à l'estimation des coûts. Idéalement, il faudrait qu'elle soit estimée réalisable tant d'un point de vue de capacité que de batterie.

Estimation du rendement Nous définissons la fonction d'estimation du rendement d'une tâche. Soit T la durée moyenne pour se déplacer entre deux points. Dès lors :

- La livraison de l'objet coûte T.
- L'achat de l'objet coûte $T + \lambda * p * n$ où p le prix et n le nombre d'objet à acheter. On note λ le coefficient représentant le poids du prix des achats par rapport au temps.
- La fabrication de l'objet coûte $T * (n_{materiau} + 1) + \lambda * c$ où $n_{materiau}$ le nombre de matériau différents pour fabriquer l'objet et c le coût d'utilisation de l'atelier. La durée correspond au nombre maximum d'objet à apporter plus au trajet du fabriquant. On remarque qu'il s'agit du même λ que précédemment.
- Les recettes de la tâches sont notées $\lambda * r$ où r est la recette.

Par exemple, l'achat et la livraison d'un objet ont un rendement R de :

$$R = -2T - \lambda * p * n + \lambda * r$$

Une tâche est acceptée si le rendement R est strictement positif. La valuation de λ est explicitée en section 4.2.

2.3 Distribution des tâches

À chaque tâche acceptée, chaque agent adopte le comportement décrit ci-dessous. Pour plus de lisibilité, nous mettons en gras les instructions.

1. **Annoncer à tous** si l'objet est possédé, et en quelle quantité ;
2. **Attendre** jusqu'à avoir reçu des messages de la part de tous les agents ;
3. **Si** l'objet est possédé, **estimer** si l'on est le meilleur agent `best_owner`, auquel cas y **aller** ;
4. **Sinon**, personne n'a l'objet, **évaluer** si l'objet peut être décomposé :
   — **Si** l'objet ne peut pas être fabriqué, **estimer** si l'on est le meilleur agent `best_buyer`, auquel cas **aller** l'acheter ;
   — **Sinon**, l'objet peut être fabriqué, **estimer** si l'on est le meilleur fabriquant `best_crafter`. Ce dernier se chargera de **créer** et d' **annoncer à tous** une nouvelle tâche, consistant à demander les matériaux et la livraison en sa position. Il s'attribuera la tâche de fabrication qu'il activera lorsqu'il possèdera tous les matériaux nécessaires.

Revenons sur les étapes pouvant prêter à confusion :

Attendre Chaque agent attend de recevoir tous les messages envoyés par ses confrères. Cette attente est limité temporellement. Détails en section 3.1.

Estimer Les estimations de `best_owner`, `best_buyer` et `best_crafter` correspondent à l'estimation des fonctions de coût, détaillées en section 3.2.

Remarquons qu'il y a unicité pour chacune de ses fonctions de coût : tous les agents obtiennent le même résultat. Il y a également unicité du meilleur coût : en supposant que deux agents obtiennent le même coût, ils sont ordonnés de manière lexicographique et seul le premier sera considéré comme le meilleur.

Notons que si les agents possèdent ensemble la quantité demandée d'un objet, sans qu'aucun agent n'en possède suffisamment, seul le meilleur agent reste considéré.

Évaluation de la décomposition Les caractéristiques des objets, et plus précisément celles concernant sa fabrication, sont connues de tous les agents : tous obtiendront donc la même évaluation.

Terminaison La fonction étant potentiellement récursive, détaillons pourquoi la distribution termine toujours.

— Du point de vue de l'agent, l'agent n'est occupé que jusqu'au calcul d'une des trois fonctions de coût ; s'il s'avère être le `best_*`, il effectuera la tâche qui lui est confiée et aura ensuite terminé.
— Du point de vue global, si un `best_crafter` est élicité, on détermine si la tâche enfant est acceptée. Si elle n'est pas acceptée, les agents ont terminé et attendent l'énonciation d'une nouvelle tâche. Sinon, on boucle. Néanmoins, comme les objets ne peuvent pas être décomposés *ad vitam*

æternam, il viendra toujour un point où l'objet ne sera plus décomposable et que le programme terminera.

3 Détails et améliorations du modèle de base

3.1 Coordination

L'ensemble des agents forme une clique. On suppose que les agents ne peuvent pas tomber en panne. Les canaux de communication sont supposé sûrs et *first in, first out*, mais asynchrone : il est possible qu'un message mette très longtemps pour parvenir à son destinataire. Ainsi, les agents n'ont pas connaissance du moment où le message envoyé a été ou sera reçu, et l'attente est potentiellement très longue en étape 2.

À défaut de pouvoir rendre notre système plus rapide, nous l'avons rendu plus sûr, en implémentant un système de synchronisation, selon lequel les agents se synchronisent et s'attendent mutuellement à certains moments critiques — comme lors de la mise à jour des connaissances. Nous avons opté pour une propagation dans une topologie en anneau adjointe à un système de jetons : chaque agent ajoute à son voisin un jeton qui fini par lui revenir. Le critère d'arrêt est validé lorsque le nombre de jeton possédé par un agent égale le nombre d'agents.

3.2 Fonctions de coût

Nous disposons de trois fonctions de coût : `get_best_owner`, `get_best_buyer` et `get_best_crafter`.

`get_best_owner` La fonction `get_best_owner` est calculée de la sorte : pour chaque agent disposant de l'objet en quantité suffisante, nous calculons la distance par rapport au lieu de livraison ou à l'atelier (selon si l'objet possédé est un matériau ou un objet fini livrable).

`get_best_buyer` Pour chaque magasin, on calcule la distance de l'agent par rapport aux magasins possibles, plus la distance du magasin concerné par rapport au lieu de livraison (ou à l'atelier), le tout pondéré par la vitesse de l'agent et le nombre de tâches qui lui ont déjà été attribuées. On sélectionne la plus petite des valeurs obtenues. Le `best_owner` est l'agent possédant la plus petite valeur ; il effectuera l'achat ainsi que la livraison.

`get_best_crafter` La fonction `get_best_crafter` est calculée de la même façon que `get_best_buyer`, à la différence que l'on compare les distances par rapport aux ateliers, et non plus aux magasins. Nous prenons également en compte les compétences de fabrication des agents.

Résistance aux pannes est possible de réévaluer les plans générés à la sortie d'un agent du système, auquel cas les fonctions de coût renvoient zéro.

3.3 Engagement vis-à-vis d'une ressource

Nous avons implémenté la gestion de l'engagement vis-à-vis d'une ressource : lorsqu'un agent déclare posséder certaines ressources, alors il s'engage à les livrer et elles ne seront donc plus disponibles, notamment dans le cas où l'agent s'interrompt pour effectuer une tâche plus prioritaires.

3.4 Gestion des tâches et priorité

Nous avons limité autant que possible le nombre de tâches proposées simultanément : lors de la phase d'initialisation, toute autre tâche que la première est refusée. Si une tâche survient lors de la réalisation d'une tâche, les agents effectuant des actions ne la considéreront que plus tard : si le meilleur agent est non-occupé, il commencera immédiatement, tandis que si le meilleur agent est occupé, il commencera après avoir fini sa mission. De même, dans le cas où une tâche est décomposée en deux sous-tâches et qu'un même agent est le meilleur pour réaliser les sous-tâches, il effectuera les tâches l'une après l'autre, dans l'ordre de réception.

4 Optimisations

La section précédente décrit un modèle permettant la résolution d'une tâche. Nous avons alors procédé à des modifications légères et des jeux de tests en simulation sur une tâche complexe, de manière à améliorer notre algorithme.

4.1 Initialisation

Nous avons comparé deux méthodes d'initialisation : envoyer l'ensemble des agents à un emplacement précis, indépendamment de la première tâche qui n'est pas nécessairement formulée ou reçue au temps 0, ou répartir arbitrairement les agents vers les emplacements.

Nous avons opté pour la seconde méthode qui nous permet de positionner les agents à des points éloignés les uns des autres, nous offrant différentes valeurs pour les fonctions de coût. Remarquons que la première méthode est bien plus efficace dans le cas de la résolution d'une tâche précise et que nous connaissons à l'avance (ce qui est le cas dans nos jeux de test) ; toutefois nos simulations nous permettent de supposer qu'une telle répartition serait moins efficace si les agents doivent effectuer une série de tâches.

4.2 Acceptation

La diversité des tâches étudiées ne nous permet pas d'affirmer avoir trouver le λ optimal, néanmoins nous pouvons borner λ : trop faible, on refusera toutes les tâches ; trop élevé, on acceptera des tâches lointaines pour lesquelles les chances de succès vis-à-vis de l'équipe adverse sont faibles.

Nous avons choisi $\lambda = 10$.

5 Évaluation des performances

L'algorithme que nous avons présenté nous permet de résoudre les tâches suivantes :

- Une fois le travail accepté, Si un agent possède l'objet demandé en quantité suffisante, il est capable d'effectuer la livraison. Si plusieurs agents possèdent l'objet, l'algorithme est capable de déterminer l'agent le plus près de l'endroit où la livraison s'effectue, et de l'y envoyer.
- Si aucun agent ne possède l'objet demandé, mais qu'il peut le fabriquer et dispose de tous les matériaux/objets nécessaires, il est capable de se déplacer à l'atelier le plus proche et de fabriquer l'objet. Si plusieurs agents sont dans le même cas, l'algorithme attribue cette tâche à l'agent le plus proche d'un atelier.

Il reste cependant des tâches présentant des problèmes :

- Lorsqu'un agent ne possède pas l'objet demandé et que cet objet ne peut pas être fabriqué, l'algorithme lui attribue la tâche d'aller acheter cet objet. L'agent parvient à se rendre au magasin le plus proche, mais n'achète pas l'objet une fois là bas.
- Nos agents présentent des difficultés à assister la fabrication d'un objet. En effet, même une fois arrivé à l'atelier et en possession des pièces nécessaires, l'agent assistant ne parvient pas à se synchroniser avec l'agent déjà présent.

Des tests plus précis sont donc nécessaires pour apporter une solution à ces problèmes.

6 Conclusions et perspectives

Le travail effectué sur ce projet nous a permis de nous essayer à la programmation orientée agents. Si le modèle s'éloigne de la réalité physique des SMA (réseaux de communication sujets aux erreurs et aux défaillances), ce premier projet de CoCoMA fut une bonne introduction aux différentes problématiques rencontrées lors de la création d'un système multi-agent : gestion des communications, prévention des inter-blocages, génération de plans ...

Nous avons proposé un motif général permettant de résoudre des tâches proposées, reposant sur le choix du meilleur agent qu'il s'agisse de livrer, fabriquer ou acheter un objet. Dans la présente version, le meilleur agent est celui répondant aux meilleurs critères de compétence et de proximité. Par exemple, le meilleur fabriquant est l'agent qui sera à même de livrer l'objet au plus tôt et étant capable de le fabriquer.

Nous nous sommes heurtés à des difficultés de prise en main des langages proposés ; la documentation est minimaliste ce qui nous a conduit à devoir procéder à de nombreux tests pour s'assurer de certaines propriétés. Qui plus est, certaines modifications apportées à la version originale (suppression des coordonnées et longitude et latitude) ont induit un décalage entre les réalités de l'exercice et les informations fournies dans la documentation.

Perspectives Le projet proposé étant particulièrement riche et complet, nous disposons de très nombreux axes d'amélioration, tant pour intégrer la diversité des possibilités offertes que pour offrir une meilleure résolution.

Privilégiant la mise en place d'un système simple, fiable et aussi efficace que possible, nous avons mis de côté certains aspects du projet. Celui-ci offrait en effet des contraintes que nous avons négligées :

Capacité Les agents sont dotés d'une capacité maximale d'objets à transporter, capacité que nous ne prenons pas en compte : il est possible que des agents s'attribuent des tâches qui ne sont pas réalisables. Une résolution possible de se problème serait de modifier la fonction d'acceptation des tâches en ajoutant un terme d'estimation de la capacité nécessaire : si un agent ne peut effectuer la tâche, il informe les autres de ne pas le prendre en compte lors de la résolution de la distribution. Si aucun agent ne peut effectuer la tâche, celle-ci est refusée. Une résolution plus efficace consisterait à employer les emplacements de stockages et à mettre en place des échanges d'objets entre les agents.

Batterie De manière similaire, nous n'avons pas pris en compte la batterie. Si l'énergie vient à manquer, les agents se retrouvent bloqués. Une première solution serait de refuser la tâche. Idéalement, il faudrait permettre aux agents d'aller recharger leurs batteries en milieu de course ou lorsqu'ils sont inactifs.

Assistance Si nous pouvons employer plusieurs agents pour mener à bien une tâche, chacun effectue une sous-tâche qui lui est propre : il n'y a pas d'assistance dans le sens où deux agents s'entraident. Le recrutement d'agents assistants permettrait une résolution de meilleure qualité et plus rapide. De plus, plusieurs agents possédants une même ressource pourraient la mettre en commun, et ainsi répondre rapidement aux demandes des clients.

Pour proposer une résolution plus acérée, nous pourrions proposer des fonctions de coût plus spécifiques. Pour ce faire, il suffit de rajouter des termes aux fonctions de coût et de les pondérer. On peut ainsi ordonner les tâches si plusieurs tâches sont disponibles en même temps selon le rendement espéré et l'occupation des agents. Outre ces termes, il est envisageable d'ajouter des termes modélisant l'état de la batterie ou de la capacité, ou encore de prendre en compte le nombre d'objets déjà possédés. La juste pondération sera calculée par essais successifs ou, idéalement, par un algorithme d'apprentissage dont le but est de tester les fonctions de coût pour un très grand nombre de tâches et de les optimiser de sorte à maximiser la réussite.

Lors de l'initialisation, nous ne conservons que la première tâche proposée par le système, rejetant les autres d'office. Si cette simplification convient à la résolution du projet de CoCoMA, il est envisageable que des tâches proposées a posteriori soient plus rentables : nous pourrions retenir l'ensemble des tâches, leur attribuer un numéro selon l'ordre de réception (les canaux de communication étant supposés *first in first out*, les tâches parviennent à chaque agent dans le même ordre), estimer le rendement de chacune, établir un plan pour la plus rentable ; les agents non affectés se partageront alors les tâches restantes.

Concernant les problèmes de synchronisation, nous avons songé à mettre en œuvre une horloge de Lamport. Une autre amélioration serait de considérer l'ensemble des agents comme un système ouvert et de pouvoir reconfigurer à la volée la distribution de tâche — notamment en cas de défaillance d'un agent.