Sujet des TP d'Algorithmique Distribuée

Sébastien Ratel

manque possibilité de mettre memoire sur noeuds, sur agents, et. trous noirs sur le reseau
-> possibilité de lesajouter ensuite

MAS (pour Mobile Agent Simulator) est un environnement de visualisation d'algorithmes distribués pour le modèle des agents mobiles.

1 - mettre memoire sur agents

1 Installation de MAS

Commencez par télécharger le simulateur à l'adresse
`https://github.com/sebastienratel/Mobile-Agent-Simulator`.

Comme décrit dans le README, les dépendances peuvent être installées à l'aide de la commande
`pip install .`

La documentation du simulateur peut être construite via la commande
`python setup.py build_sphinx`

2 Utilisation de MAS

Pour utiliser simplement le simulateur, vous n'aurez à priori besoin que de

- modifier l'affectation de `G` et de `sim` dans la fonction main de app.py;
- ajouter des fonctions au script `agent_algorithms.py` du module `agent`.

2.1 Que mettre dans le main ?

La première chose à faire va être de choisir la topologie dans laquelle vous voulez travailler. Le script `graph_generator.py` du module `graph` permet de générer des chemins, arbres binaires, arbres, grilles, lignes, graphes aléatoires ou des graphes complets. Les graphes peuvent aussi être créés à la main, mais tout ceci est expliqué dans la documentation.

Il s'agit ensuite de choisir le modèle dans lequel vous souhaitez travailler. Ceci se fait à la création d'une Simulation. La Simulation contiendra les agents mobiles (1 par défaut) et l'algorithme qu'ils doivent exécuter, la topologie, ainsi que le modèle. Dans son état initial, le simulateur vous permet de définir les points suivants du modèle:

- Réseau nommé/anonyme;
- Agents nommés/anonymes;
- Réseau synchrone/asynchrone;
- Agent avec/sans latence.

2.2 Ecriture d'algorithmes

Le code des agents mobiles devra être écrit dans le script `agent_algorithms.py` du module `agent`. Ce code se présentera sous la forme d'une fonction ayant pour seul paramètre un `agent`. Les agents mobiles disposent des primitives suivantes:

- `available_ports`: pour récupérer la liste des ports disponibles;
- `become`: pour changer de status;
- `get_id`: pour connaître son identifiant;
- `get_moves_nb`: pour connaître le nombre de mouvement effectués depuis le début;
- `get_port_back`: pour connaître le port retour vers le sommet précédent;
- `get_position_id`: pour connaître l'identifiant du nœud courant;
- `get_sim_step`: pour connaître le temps global;
- `move_along`: pour se déplacer le long d'une arête;
- `position_contains_mate`: pour savoir si le nœud courant contient un autre agent;
- `status`: pour connaître son statut courant; et
- `wait`: pour ne rien faire (éventuellement utile pour la lisibilité du code).

3 Fonctionnement de MAS

Le principe du simulateur est le suivant: Les agents font des requêtes à la simulation qui se comporte comme un contrôleur. La simulation décide d'accéder ou non aux requêtes selon le modèle. Lorsque la requête est légitime vis-à-vis du modèle, la simulation fait appel à l'AgentManager qui joue le rôle d'une base de données contenant toutes les informations à propos des agents: leur identifiant; leur position; leur éventuelle mémoire; etc.

Par exemple, la simulation répondra None à une requête d'identifiant lorsque le modèle est anonyme. Si le modèle est nommé, alors la simulation fera un appel à l'AgentManager qui connaît l'identifiant des sommets, et elle relaiera la réponse à l'agent.

3.1 Fonctionnalités manquantes

Dans l'état actuel du simulateur, vous ne pouvez pas implémenter d'algorithmes qui nécessitent:

- de la mémoire dans les agents;
- de la mémoire sur les nœuds;
- des jetons.

Il n'est pas non plus possible d'ajouter des trous noirs dans la topologie.

4 Projet

Il faudra implémenter un maximum d'algorithmes vus en cous. Les algorithmes marqués par une pastille verte peuvent être implémentés avec la version initiale du simulateur. Les algorithmes marqués par une pastille orange ne peuvent pas être implémentés avec la version initiale du simulateur. Il vous faudra donc entrer dans le code de celui-ci pour ajouter les fonctionnalités manquantes décrites en section 3.1.

Dans l'idéal, à l'issue de la deuxième séance, tous les algorithmes marqués par une pastille vertes doivent être implémentés. Les deux séances suivantes seront alors consacrées à l'ajout de fonctionnalités au simulateur pour implémenter les algorithmes marqués par une pastille orange.

Conseil: Lorsque vous ajoutez une fonctionnalité au simulateur, programmez dans la foulée les algorithmes qui la nécessite avant d'ajouter d'autres fonctionnalités.

Lors de la dernière séance, vous devrez présenter votre travail.