M4102C - Algorithmique distribuée

Election, puis tranformation de graphes

TD2

1 Élection

L'élection est le problème qui consiste à distinguer un nœud parmi les autres, à partir d'une initialisation uniforme (tout le monde démarre dans le même état). Dans les exercices suivants, on supposera parfois que les nœuds ont des identifiants uniques, et parfois non (réseau anonyme). De même, on supposera parfois un réseau synchrone (communication en rondes) et parfois asynchrone (les messages prennent un temps arbitraire, mais quand même *fini*).

1.1 Élection dans les arbres

On suppose ici un réseau *asynchrone* dont la topologie est un arbre *sans racine*. Les nœuds ont des identifiants uniques (mais arbitraires). Les nœuds savent combien de voisins ils ont. Ils sont aussi capables d'identifier le lien sur lequel arrive un message et de choisir le lien sur lequel ils envoient un message. Tous les nœuds démarrent simultanément.

1. Concevoir un algorithme d'élection qui fonctionne dans ce contexte. Vous pouvez utiliser le format de votre choix pour décrire l'algorithme (pseudo code ou explications textuelles), du moment que c'est clair et précis.
2. Combien de messages votre algorithme a-t-il utilisé (au plus) lorsqu'un nœud apprend qu'il est élu ? (On ignorera la partie qui consiste à informer les autres nœuds du résultat.)

1.2 Borne inférieure sur le nombre de messages ?

Soit l'affirmation suivante : "Il n'existe aucun algorithme d'élection *universel* (= qui marche dans toute topologie) qui utilise toujours moins de m messages (m = nombre d'arêtes)."

Cela est-il vrai ou faux ? Si faux, pouvez-vous penser à un algorithme moins coûteux ? Vous pouvez, pour réfléchir, supposer qu'il y a des identifiants ou non, et que le réseau est synchrone ou non.

1.3 Élection probabiliste

On considère un réseau *synchrone* dont la topologie est un graphe *complet*. Voici un algorithme simple d'élection probabiliste :

Algo 1 : A chaque ronde, les nœuds non-éliminés tirent à pile ou face. Ceux qui tirent pile envoient un message à tout le monde et les autres s'auto-éliminent. On répète cette étape tant qu'il y a au moins deux nœuds en lice. Si un seul noeud a envoyé un message, il est élu.

1. Cet algorithme réussit-il toujours ou peut-il échouer ? Pourquoi ?
2. Le temps d'exécution est-il borné ? est-il fini ? peut-il être infini ? (*)
3. Quelle est la probabilité de succès si $n = 2$, si $n = 3$?

(*) *Fini* signifie que quelque chose se produira tôt ou tard (mais potentiellement arbitrairement tard), tandis que *borné* implique qu'il existe une date qui ne sera pas dépassée.

On considère maintenant un autre algorithme basé sur le premier.

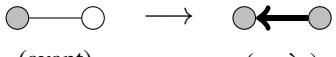
Algo 2 : Lancer l'algorithme 1. Tant qu'il échoue, le relancer.

1. Cet algorithme réussit-il toujours ? Pourquoi ?
2. Combien de tirages cela prendra-t-il en moyenne pour réussir l'élection avec $n = 2$?

2 Transformation de graphes

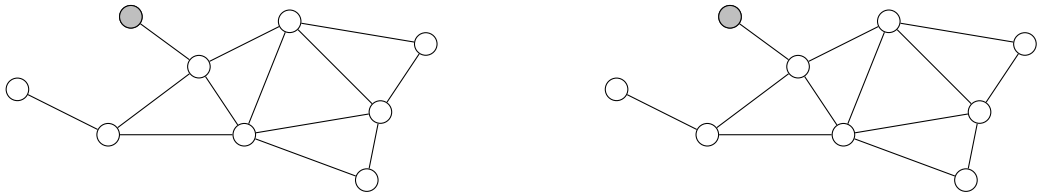
Il s'agit d'un modèle abstrait pour l'algorithmique distribuée. Ici on ignore la façon dont les nœuds communiquent réellement (messages ou autres) en décrivant les algorithmes par des *règles de transformation* sur des nœuds voisins. Par exemple, la règle r_1 ci-dessous nous dit que si un nœud gris interagit avec un nœud blanc, ce dernier devient gris et leur lien commun pointe vers le premier.

Règle r_1 :



Comment l'algorithme choisit où se font les interactions ? Il ne choisit pas ! Vous pouvez imaginer une sorte de jeux entre votre algorithme et une autre entité qui, à chaque étape, choisit une arête et demande à l'algorithme d'y appliquer une règle (si cela est possible). L'ordre dans lequel les arêtes sont choisies est quelconque, du moment que chaque arête est choisie de temps en temps.

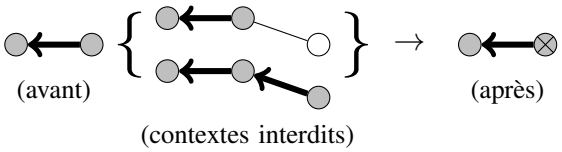
1. Éxecutez l'algorithme qui consiste en la règle r_1 sur le graphe de gauche. Faites-le jusqu'à ce que l'état du système soit stable (plus rien ne peux changer).



2. Que fait cet algorithme ?

Deuxième règle. On ajoute maintenant une seconde règle à l'algorithme, plus complexe :

Règle r_2 :



Explication : Les parties entre accolades indiquent les cas où la règle ne doit pas être appliquée. Ainsi, cette règle transforme ●←● en ●←⊗ à condition que le second nœud n'ait aucun voisin blanc (via une arête normale) et aucun voisin gris (via une arête orientée sur lui-même). Attention : l'état ⊗ est différent de l'état ●.

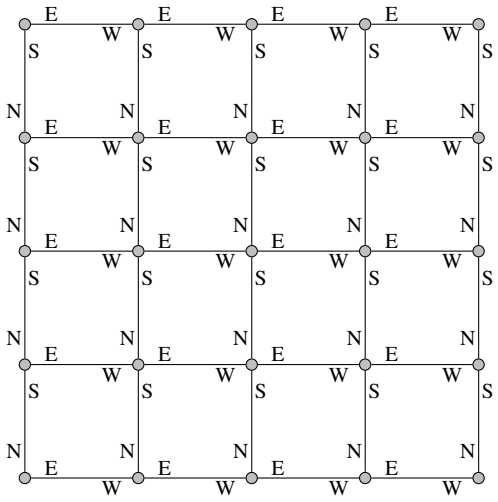
3 (en binôme) Exécutez l'algorithme qui consiste en les deux règles r_1 et r_2 de la manière suivante. L'un de vous fait la sélection des arêtes (en se comportant plutôt comme un adversaire) et l'autre y applique r_1 si possible, sinon r_2 si possible, sinon rien. Répétez jusqu'à ce que l'état soit stable (plus rien ne peux changer).

4 Que fait cet algorithme ?

3 Complément (selon état d'avancement)

3.1 Diffusion dans une grille

On considère une topologie en grille carrée (voir dessin, pour $n = 25$). On considère un processus de diffusion à partir du nœud le plus en haut à gauche. Les nœuds savent qu'ils sont dans une grille, mais ils ne savent pas où dans la grille (sauf l'émetteur initial).



1. Combien de messages seront échangés si l'on utilise l'algorithme de diffusion basique vu la semaine dernière ? ("La première fois que je reçois le message, je le retransmets à tous mes voisins avec `sendAll()`.") Le fait que le réseau soit synchrone ou asynchrone a-t-il un impact sur ce nombre ? Si oui, lequel ?
2. On suppose maintenant que les nœuds peuvent utiliser les primitives `sendN()`, `sendS()`, `sendE()`, `sendW()` plutôt que `sendAll()`. En revanche, il ne savent toujours pas où ils sont ni même s'ils ont un voisin dans une direction donnée (ce qui peut induire des messages perdus dans le vide).

   → Écrivez un algorithme qui tient compte de la topologie en grille. Combien de messages sont envoyés (en incluant les messages envoyés dans le vide) ?
3. On suppose maintenant que les nœuds peuvent connaître la provenance d'un message `msg` en utilisant `msg.getOrigin()` dont la réponse est parmi `{N,S,E,W}`. Ils peuvent aussi tester s'ils ont un voisin dans une direction donnée (ex : `hasNeighbor(N)`).

   → Écrire un algorithme qui utilise ces informations pour n'envoyer que $n - 1$ messages.

   Peut-on espérer faire mieux ? Et dans une autre topologie ? Pourquoi ?