Implantation d'algorithmes autostabilisants dans *Sinalgo*

Stéphane Devismes

Université Joseph Fourier, Grenoble I

9 octobre 2018

Plan

Plan

Principe

Principe

1. Chaque processus envoie son état courant régulièrement à tous ses voisins.

Principe

1. Chaque processus envoie son état courant régulièrement à tous ses voisins.
2. Chaque processus stocke le dernier état connu de chacun de ses voisins.

Principe

1. Chaque processus envoie son état courant régulièrement à tous ses voisins.
2. Chaque processus stocke le dernier état connu de chacun de ses voisins.
3. Tous les variables sont initialisés avec une valeur choisie au hasard dans leur domaine de définition.

Principe

1. Chaque processus envoie son état courant régulièrement à tous ses voisins.
2. Chaque processus stocke le dernier état connu de chacun de ses voisins.
3. Tous les variables sont initialisés avec une valeur choisie au hasard dans leur domaine de définition.
4. Régulièrement :
   - 4.1 Pour tout voisin v, tester la réception d'un message contenant l'état de v.
     - 4.1.1 Le cas échéant, mise à jour du dernier état connu de v.

Principe

1. Chaque processus envoie son état courant régulièrement à tous ses voisins.
2. Chaque processus stocke le dernier état connu de chacun de ses voisins.
3. Tous les variables sont initialisés avec une valeur choisie au hasard dans leur domaine de définition.
4. Régulièrement :
   - 4.1 Pour tout voisin v, tester la réception d'un message contenant l'état de v.
     - 4.1.1 Le cas échéant, mise à jour du dernier état connu de v.
   - 4.2 Le processus inspecte son état et les derniers états connus de chacun de ses voisins pour voir si une règle s'applique.

Principe

1. Chaque processus envoie son état courant régulièrement à tous ses voisins.
2. Chaque processus stocke le dernier état connu de chacun de ses voisins.
3. Tous les variables sont initialisés avec une valeur choisie au hasard dans leur domaine de définition.
4. Régulièrement :
   - 4.1 Pour tout voisin v, tester la réception d'un message contenant l'état de v.
     - 4.1.1 Le cas échéant, mise à jour du dernier état connu de v.
   - 4.2 Le processus inspecte son état et les derniers états connus de chacun de ses voisins pour voir si une règle s'applique.
     - 4.2.1 Le cas échéant, le processus applique la règle activable.

Ressources

`http://www-verimag.imag.fr/~devismes/WWW/enseignements.html`

1. Sources *Sinalgo*
2. Lien Tutorial *Sinalgo*
3. TP d'installation et d'initiation
4. `FifoRandom2.java`

Rappel : fichiers principaux

Un algorithme $=$ un répertoire dans `projects`

(copie du répertoire `template`)

Rappel : fichiers principaux

Un algorithme = un répertoire dans `projects`

(copie du répertoire `template`)

Contenu

1. `CustomGlobal.java` : accès à la configuration globale (utilité : détection de terminaison)
2. `description.txt` : description de l'algorithme qui sera affichée dans la fenêtre « menu »
3. `Config.xml` : paramètres de simulation
4. Répertoires : `images, models, nodes`

Exemple : Ensemble Indépendant Maximal

On crée un répertoire `MIS` copie du répertoire `template` dans le répertoire `projects`

Puis, on remplit le fichier `Config.xml`

Plan

`Config.xml` : champs principaux (1/7)

Le réseau est déployé sur un plan rectangulaire 2D

```
<dimensions value="2" />
```

Taille du rectangle

```
<dimX value="1000" />

<dimY value="1000" />
```

`Config.xml` : champs principaux (2/7)

Processus asynchrone ? Pour simplifier non (seuls les liens le seront)

```
<asynchronousMode value="false" />
```

Processus mobile ? non

```
<mobility value="false" />
```

Pas d'interférence sur la ligne

```
<interference value="false" />
```

Liens bidirectionnels

```
<edgeType
     value="sinalgo.nodes.edges.BidirectionalEdge" />
```

`Config.xml` : champs principaux (3/7)

Transmission FIFO asynchrone :

```
<DefaultMessageTransmissionModel value="FifoRandom2" />
```

Il faut ajouter dans la balise dans `<Custom>`

```
<FifoRandom distribution="Exponential" lambda="10"/>
```

Un message mets en moyenne 10 unités de temps à arriver

`Config.xml` : champs principaux (4/7)

Deux processus sont voisins s'ils sont à moins d'une certaine distance (portée) :

```
<DefaultConnectivityModel value="UDG" />
```

Il faut ajouter dans deux balises dans `<Custum>` pour spécifier la portée

```
<GeometricNodeCollection rMax="150"/>
<UDG rMax="150"/>
```

Portée 150 pixels

`Config.xml` : champs principaux (5/7)

Topologie :

```
<DefaultDistributionModel value="PositionFile" />
```

Ici, issue d'un fichier (sinon, `Circle`, `Line2D`, `Grid2D`)

Pas d'interférence et pas de mobilité

```
<DefaultInterferenceModel value="NoInterference" />
<DefaultMobilityModel value="NoMobility" />
```

`Config.xml` : champs principaux (6/7)

Fiabilité des liens :

```
<DefaultReliabilityModel value="LossyDelivery" />
```

Il faut ajouter dans une balise dans `<Custum>` pour spécifier le taux de pertes

```
<LossyDelivery DropRate="0.4"/>
```

Ici 40% de pertes

`Config.xml` : champs principaux (7/7)

Où trouver le code des processus :

```
<DefaultNodeImplementation value="MIS:MISNode" />
```

Ici dans le fichier `MISNode.java` du répertoire `MIS`

Dans `<Custom>`, taille minimum des processus à affichage :

```
<Node defaultSize="20" />
```

Plan

Nouveau type : `ValeurS.java` dans `MIS`

```
//Nouveau type pour coller à l'algorithme
public enum ValeurS {
     Dominant,domine;

     public static ValeurS RandomInit(){
          return ((int) (Math.random() * 10000.0))%2
                  == 1 ?
                  ValeurS.Dominant:ValeurS.domine;
        }
}
```

Stockage de l'état des voisins : `EtatVoisin.java` dans `MIS/nodes/nodeImplementations`

```
package projects.MIS;

public class EtatVoisin{
     public  int ID;
     public ValeurS S;

     public EtatVoisin(){
          ID=(int) (Math.random() * 10000.0);
          S=ValeurS.RandomInit();
     }
}
```

Un seul type de message : `MISMessage.java` dans `MIS/nodes/messages`

```
package projects.MIS.nodes.messages;

import projects.MIS.ValeurS;
import sinalgo.nodes.messages.Message;

public class MISMessage extends Message {
public int ID;
public ValeurS S;

public MISMessage(int ID, ValeurS S){
     this.ID=ID;
     this.S=S;
}

public Message clone() {
     return new MISMessage(ID,S);
}
}
```

Code des processus : `MISNode.java` dans `MIS/nodes/nodeImplementations`, import

```
package projects.MIS.nodes.nodeImplementations;

import java.awt.Color;
import java.awt.Graphics;
import java.util.Iterator;
import projects.MIS.EtatVoisin;
import projects.MIS.ValeurS;
import projects.MIS.nodes.messages.MISMessage;
import projects.MIS.nodes.timers.initTimer;
import projects.MIS.nodes.timers.waitTimer;
import sinalgo.configuration.WrongConfigurationException;
import sinalgo.gui.transformation.PositionTransformation;
import sinalgo.nodes.Node;
import sinalgo.nodes.edges.Edge;
import sinalgo.nodes.messages.Inbox;
import sinalgo.nodes.messages.Message;
import sinalgo.tools.Tools;
```

Code des processus : `MISNode.java` dans `MIS/nodes/nodeImplementations`, fonctions obligatoires

```
public class MISNode extends Node {
    public void preStep() {}
    // ATTENTION lorsque init est appelé,
    // les liens de communications n'existent pas encore
    // il faut attendre une unité de temps,
    // avant que les connections soient réalisées
    // nous utilisons donc un minuteur pour déclencher
    // la véritable initialisation ``start'' après 1 unité de temps
    public void init() {}
    public void start(){}

    //  handleMessages est appelée régulièrement même si
    //  aucun message n'a été reçu
    public void handleMessages(Inbox inbox)  {}
    public void postStep() {}
    public void checkRequirements() throws
          WrongConfigurationException {}
    // affichage du noeud
    public void draw(Graphics g,
          PositionTransformation pt, boolean highlight){}
}
```

Code des processus : `MISNode.java` dans `MIS/nodes/nodeImplementations`, variables du noeud

```
// état du noeud initialisé au hasard,
// mais dans son domaine de définition
 private ValeurS S=ValeurS.RandomInit();

 // pour stocker le dernier état connu de chaque voisin
 private EtatVoisin Voisins[];
```

Code des processus : `MISNode.java` dans `MIS/nodes/nodeImplementations`, boîte à outils

```
// getIndex retourne le numéro de canal correspondant à ID
// si l'identité n'existe pas, on retourne 0 :
// pas de soucis avec les tableaux !
private int getIndex(int ID){
     int j=0;
     for(Edge e: this.outgoingConnections){
          if (e.endNode.ID==ID) return j;
          j++;
     }
     return 0;
}
// getVoisin retourne l'ID correspondant au numéro ``indice''
// si l'indice n'existe pas, on retourne l'identité du noeud
private int getVoisin(int indice){
     if (indice >= this.nbVoisins() || indice < 0)
          return this.ID;
     Iterator<Edge> iter=this.outgoingConnections.iterator();
     for(int j=0;j<indice;j++) iter.next();
     return iter.next().endNode.ID;
}
private int nbVoisins(){
     if (this.outgoingConnections == null) return 0;
     return this.outgoingConnections.size();
}
```

Code des processus : `MISNode.java` dans `MIS/nodes/nodeImplementations`, problème de l'initialisation

```
public void init() {
     (new initTimer()).startRelative(1,this);
}
```

`initTimer()` est défini dans `initTimer.java` dans `MIS/nodes/timers`

```
package projects.MIS.nodes.timers;

import projects.MIS.nodes.nodeImplementations.MISNode;
import sinalgo.nodes.timers.Timer;

public class initTimer extends Timer {
     public void fire() {
          MISNode n= (MISNode) this.node;
          n.start();
     }
}
```

Code des processus : `MISNode.java` dans `MIS/nodes/nodeImplementations`, démarrage

```
public void start(){
     if(this.nbVoisins()> 0){
          // création et initialisation
          // du tableau où seront stockés
          // les derniers états connus de chaque voisin
          this.Voisins=new EtatVoisin[this.nbVoisins()];
          for(int i=0;i<this.nbVoisins();i++)
                this.Voisins[i]=new EtatVoisin();
     }
     (new waitTimer()).startRelative(20, this);
}
```

`waitTimer()` est défini dans `waitTimer.java` dans `MIS/nodes/timers` : régulièrement, le processus envoie son état à ses voisins.

```
package projects.MIS.nodes.timers;
import projects.MIS.nodes.nodeImplementations.MISNode;
import sinalgo.nodes.timers.Timer;

public class waitTimer extends Timer {
     public void fire() {
          MISNode n= (MISNode) this.node;
          n.envoi();
     }
}
```

Code des processus : `MISNode.java` dans `MIS/nodes/nodeImplementations`, fonction `envoi`

```
// Un noeud envoie régulièrement son état à ses voisins
public void envoi(){
      // broadcast envoie à tous les voisins
      this.broadcast(new MISMessage(this.ID,this.S));
      // waitTimer appelle de nouveau envoi
      // dans 20 unités de temps
      (new waitTimer()).startRelative(20, this);
}
```

Code des processus : `MISNode.java` dans `MIS/nodes/nodeImplementations`, l'algorithme

```
// Garde de l'action Leave
private boolean Leave(){
     if(this.S==ValeurS.domine) return false;
     for(int i=0;i<this.nbVoisins();i++)
          if(this.Voisins[i].S==ValeurS.Dominant &&
               Voisins[i].ID<this.ID) return true;
     return false;
}
// Garde de l'action Join
private boolean Join(){
     if(this.S==ValeurS.Dominant) return false;
     for(int i=0;i<this.nbVoisins();i++)
          if(this.Voisins[i].S==ValeurS.Dominant &&
              Voisins[i].ID<this.ID) return false;
     return true;
}
// Gère la partie "action" des deux règles
private void Actions(){
     if (this.Leave()) this.S=ValeurS.domine;
     if (this.Join()) this.S=ValeurS.Dominant;
}
```

Code des processus : `MISNode.java` dans `MIS/nodes/nodeImplementations`, réception de messages

```
// la fonction ci-dessous gère la réception de messages
// elle est appelée régulièrement même si
// aucun message n'a été reçu
public void handleMessages(Inbox inbox)  {
// on parcourt la liste de tous les messages reçus
while(inbox.hasNext()){
     Message m=inbox.next();
     if(m instanceof MISMessage){
          MISMessage msg=(MISMessage) m;
          // on met à jour l'état du voisin
          this.Voisins[this.getIndex(msg.ID)].S=msg.S;
          this.Voisins[this.getIndex(msg.ID)].ID=msg.ID;
          }
     }
// l'état des voisins peut avoir changé
// on recalcule donc l'état du noeud
this.Actions();
}
```

Code des processus : `MISNode.java` dans `MIS/nodes/nodeImplementations`, affichage

```
// la couleur du noeud dépend de son état
private Color Couleur(){
     if(this.S==ValeurS.Dominant) return Color.red;
     return Color.yellow;
}

 // affichage du noeud
 public void draw(Graphics g,
        PositionTransformation pt, boolean highlight){
 this.setColor(this.Couleur());
 String text = ""+this.ID;
 super.drawNodeAsDiskWithText(g, pt,
        highlight, text, 20, Color.black);
 }
```

Code des processus : `MISNode.java` dans `MIS/nodes/nodeImplementations`, gestion dynamicité

```
public void neighborhoodChange() {
     if(this.nbVoisins()> 0){
          this.Voisins=new EtatVoisin[this.nbVoisins()];
          for(int i=0;i<this.nbVoisins();i++)
              this.Voisins[i]=new EtatVoisin();
     }
}
```

Plan

Boîte à outils (suite)

```
private static int nbNoeuds(){
          return Tools.getNodeList().size();
     }

private Node Successeur(){
          int succ=(this.ID%this.nbNoeuds())+1;
          for(Edge e : this.outgoingConnections)
               if(e.endNode.ID==succ)
                    return e.endNode;
          return null;
     }


// Envoie d'un message à un processus particulier
// ici le successeur du processus dans l'anneau
this.send(message, this.Successeur());
```