Calcul de Valeurs d'Arguments : Manuel Utilisateur

Edouard Delasalles
Thomas Gerald
Alexis Martin
Thibault Mouton

May 2015

1 Vue d'ensemble

L'interface graphique de l'application est divisée en trois parties (voir figure 1) :

(1) Le menu
(2) La partie algorithme
(3) La partie graphe

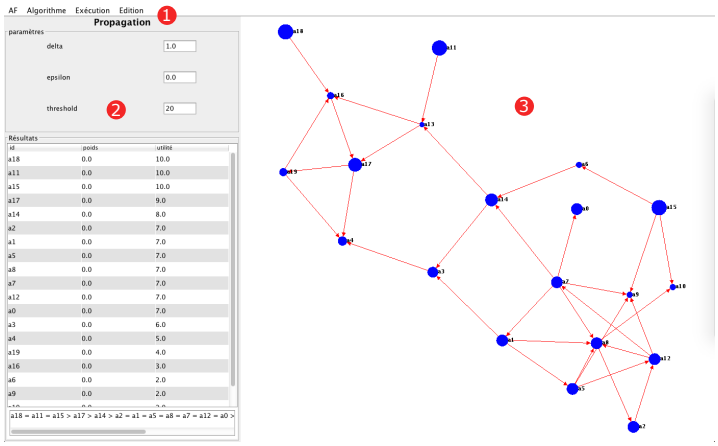

FIGURE 1 – Vue d'ensemble de l'IHM

1.1 Menu

Le menu sert à sélectionner des actions à réaliser. Dans la suite, les lettres entre parenthèses font références à la figure 2.

Le menu `AF` (a) sert à charger et sauvegarder des réseaux d'argumentations. Les formats supportés sont `.dot`, `.dgs`, `.apx` et `.tgf`. Quand un réseau est chargé, il devient actif. Il est affiché dans la partie (3) de l'IHM (figure 1). Cliquer sur le bouton `Enregistrer` enregistrera le réseau actif.

Le menu `Algorithme` (b) sert à sélectionner un algorithme. Cliquer sur un des noms d'algorithmes rendra celui-ci actif, et son nom, ainsi que ses paramètres, apparaîtront alors en haut de la partie (2) de l'IHM (figure 1).

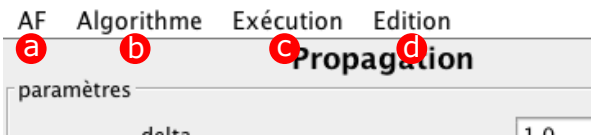

FIGURE 2 – Menu

Le menu `Exécution` (c) sert à exécuter les algorithmes. Il possède deux options :

`Lancer` : exécute l'algorithme actif sur le réseau d'argumentation actif. Il faut donc avoir sélectionner au préalable un algorithme et un réseau pour exécuter cette option.

`Batterie de tests` : permet d'exécuter une campagne de tests (c.f. 2).

Le menu `Édition` (d) sert à faire passer le graphe actif en mode éditable. Il est ainsi possible de manipuler graphiquement les noeuds, d'en ajouter et d'en supprimer (c.f. 1.3).

1.2 Partie Algorithme

Cette partie de l'IHM contient les informations relatives à l'algorithme actif, qui peut être modifié via le menu `Algorithme` (c.f. 1.1). Elle est divisée en trois sous-parties (figure 3).

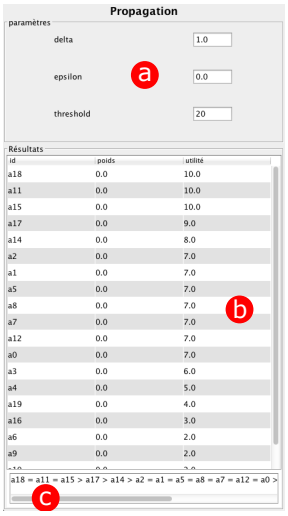

FIGURE 3 – Partie Algorithme

La partie (a) contient les paramètres de l'algorithme actifs. Ces paramètres sont dynamiques, ils reflètent les valeurs qui seront utilisés lors de la prochaine exécution avec l'option `Lancer` du menu `Exécution`. Il suffit de modifier leurs valeurs dans les champs de textes associés et de cliquer sur `Lancer` pour voir l'effet sur le résultat de l'algorithme.

La partie (b) liste tous les arguments du réseau actif, ainsi que leurs poids et leurs utilités. Tous les algorithmes que nous avons implémentés utilisent des poids initiaux sur les arguments. En particuliers, les algorithmes utilisant la méthode des points fixes sont très sensibles aux poids initiaux. Ces poids initiaux peuvent donc être modifiés en double cliquant sur la case correspondante (deuxième colonne du tableau). Les utilités (troisième colonne), quant à elles, ne sont pas éditables, et sont mises à jour après chaque exécution. Ces valeurs sont spécifiques à chaque algorithme, et représentent l'ordre de préférence entre les arguments. Quand l'algorithme n'utilise pas de valeur numérique pour représenter les préférences, la valeur indique la position de l'argument dans la liste de préférence. Ce sont les valeurs des utilités qui définissent la taille des noeuds dans la partie graphe.

La partie (c) affiche sous forme de chaîne de caractères le résultat de l'algorithme. $a = b$ signifie que l'argument a à la même "importance" ou est "équivalent" à l'argument b. $a > b$ signifie $a \succ b$, c'est à dire que a est "plus important" ou "préféré" à b.

1.3 Partie Graphe

Cette partie affiche sous forme de graphe le réseau d'argumentation actif (indice (3) sur la figure 1). Les noeuds représentent les arguments du réseau d'argumentation actif, labélisés par leur id. Il est possible de déplacer la caméra avec les touches (si la commande ne marche pas du premier coût, cliquez sur la fenêtre du graphe et recommencez) :

- Zoom + : Page ↑ (fn + ↑)
- Zoom - : Page ↓ (fn + ↓)
- Déplacer : ← ↓ ↑ →

En mode édition (c.f. 1.1), il est possible de modifier le graphe actif. Cela se fait via les commandes suivantes :

- Sélectionner un noeud : Clique gauche sur un noeud
- Sélection multiple : Ctrl + clique gauche (cmd + clique gauche)
- Ajouter un noeud : Clique droit
- Ajouter une arrête : Sélectionner un ou plusieurs noeuds sources, puis maintenir Ctrl + A (cmd + A) et clique gauche sur le noeud destination
- Rafraîchir la vue : Ctrl + R (cmd + R)
- Annuler une action : Ctrl + Z (cmd + Z)
- Rétablir une annulation : Ctrl + Y (cmd + Y)

2 Effectuer une campagne de test

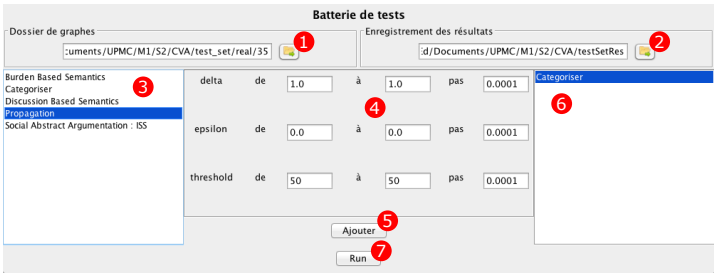

FIGURE 4 – Batterie de tests

L'option `Batterie de tests` du menu `Exécution` permet de lancer une campagne de test. Elle ouvre une nouvelle fenêtre (figure 4) permettant de configurer une campagne de tests.

Lancer une campagne de tests se fait selon le schéma suivant :

(1) Choisissez le répertoire contenant les réseaux d'argumentations à tester. Les fichiers peuvent être dans des formats différents, mais doivent être supportés (c.f. 1.1).
(2) Indiquez le répertoire dans lequel enregistrer les résultats.
(3) Sélectionnez un algorithme dans la liste.
(4) Choisissez les valeurs des paramètres à tester
(5) Appuyez sur le bouton `Ajouter`. Le nom de l'algorithme apparaît alors dans le zone (6).
(7) Appuyez sur le bouton `Run`

Les résultats seront enregistrés au format `csv` à l'endroit spécifié. Un fichier CSV sera créé pour chaque algorithme ajouté dans la zone (6). La figure 5 montre un exemple de fichier généré par la procédure.

Il est recommandé de ne pas avoir de graphe chargé lors du lancement d'une campagne de tests. En effet, l'affiche du graphe consomme beaucoup de ressources CPU. Il est donc plus intéressant de garder ces ressources pour l'exécution des algorithmes.

Graphe	Nombre d'arguments	Nombre de relations	epsilon	xhi	Temps (ms)	Temps (h:m:s.ms)	Résultats
real_0_5.apx	5000	7364	0.001	0.1	348	0:0:0.348	a_9 = a_15 = a_24 = a_29 = a
real_10_15.apx	15002	21899	0.001	0.1	1094	0:0:1.94	a_7 = e_7 = c_38 = e_40 = a_
real_16_20.apx	20000	29267	0.001	0.1	1443	0:0:1.443	a_9 = e_31 = c_36 = a_39 = a
real_19_35.apx	35000	51339	0.001	0.1	2979	0:0:2.979	a_7 = a_18 = e_18 = e_24 = e
real_3_35.apx	95000	138757	0.001	0.1	8229	0:0:8.229	a_30 = e_30 = a_48 = e_52 =
real_6_10.apx	10005	14658	0.001	0.1	663	0:0:0.663	e_10 = e_26 = a_30 = e_51 =
real_8_50.apx	50000	73174	0.001	0.1	4293	0:0:4.293	e_18 = a_23 = a_28 = a_41 =
real_9_15.apx	15000	21883	0.001	0.1	1047	0:0:1.47	c_57 = e_58 = a_59 = c_60 =

FIGURE 5 – Exemple de résultats sous forme d'un fichier CSV

3 Implémenter son propre algorithme

Il est possible d'implémenter son propre algorithme de calcul de valeurs d'arguments. Pour cela, il est nécessaire d'avoir le code source de l'application, récupérable depuis la page du projet sur GitHub (`https://github.com/Alexis-Martin/CVA/releases/latest`). Un exemple simplifié d'un algorithme customisé est présenté sur la figure 6.

Pour implémenter son propre algorithme, il faut créer une classe héritant de `AbstractAlgorithm`, qui implémente elle même l'interface `Algorithme`, dans le paquet `cva.algo.implem`. Un constructeur et 3 méthodes doivent être implémenté pour respecter l'interface et la classe abstraite.

3.1 Méthodes et constructeur

Le constructeur doit être vide, et doit appeler le constructeur `super` avec en paramètre une chaîne de caractères représentant le nom de votre algorithme. Ce nom sera utilisé dans le menu `Algorithme` de l'IHM pour pouvoir sélectionner votre algorithme.

Les trois méthodes à implémenté ont pour prototype :

```
public void init();
public void run();
public void end();
```

Ces trois méthodes sont appelées dans cette ordre quand l'algorithme est lancé via l'interface graphique. Il faut implémenter ces méthodes, mais elle peuvent être vide.

3.2 Paramètres

Si votre algorithme possède des paramètres et que vous voulez pouvoir les modifier à la voler dans l'interface graphique, il vous faudra utiliser les objets `Parameter` du paquet `cva.algo` . Ce type d'objet contient un nom (`String`), une valeur (`Object`) et une description (`String`). Il peut être instancié de la manière suivante avec le constructeur `Parameter p = new Parameter(String name, Object val)`.

Une fois le paramètre `p` créée, il faut l'ajouter à l'algorithme. Pour cela il faut utiliser la méthode `addParameter(Parameter p)`. Il est ensuite possible de récupérer la valeur de ce paramètre, qui à pu être modifié dans l'interface graphique, avec la méthode `public Parameter getParam(String name);`.

3.3 Réseau d'argumentation

La classe `AbstractAlgorithm` a en attribut un objet de type `ArgumentationFramework`, du paquet `cva.af`. Cet objet est passé par l'IHM via la méthode `public void setGraph(ArgumentationFramework af)` de l'interface `Algorithm`, que vous pouvez redéfinir, en prenant soin d'appeler `super.setGraph(af);` au début.

Vous pourrez ensuite manipuler ce réseau d'argumentation dans votre algorithme, en l'appelant avec la méthode `public ArgumentationFramework super.getGraph()`, et le manipuler grâce aux différentes méthodes de l'interface `ArgumentationFramework`.

Il est en particulier possible de manipuler des attributs customisés sur chacun des éléments du réseau (le réseau lui même, ses arguments, et les relations entre ses arguments). Ils sont accessibles en appelant les méthodes suivantes sur un objet de type `ArgumentationFramework`, `Argument` ou `Relation` (du paquet `cva.af`) :

- `public void setAttr(String attribute, Object value);`
- `public Object getAttr(String attribute);`
- `public void removeAttr(String attribute);`

3.4 Poids et utilités

Nous distinguons deux types de valeurs pour les arguments d'un réseau : leurs poids et leurs utilités. Les poids servent à initialiser l'algorithme et les utilités servent à classer les arguments. Il est possible de spécifier des poids par défauts en implémentant la méthode `public double getDefaultInitUtility();` et en renvoyant simplement un `double`. Ces valeurs seront alors affichées dans l'IHM et éditable par l'utilisateur. Ces valeurs sont ensuite accessibles grâce aux méthodes `public double getWeight();` et `public void setWeight(double w);` de l'interface `Argument`.

Les utilités sont aussi affichées dans l'IHM, mais ne peuvent pas être modifiées. Ce sont les sorties d'un algorithme. Ces valeurs qui déterminent l'ordonnancement des arguments, ainsi que la taille de leurs noeuds associé dans l'IHM. ces utilités sont modifiables grâce aux méthodes `public double getUtility();` et `public void setUtility(double u);` de l'interface `Argument`. Il est donc intéressant de mettre à jour les utilités des arguments du réseau dans la méthode `end()` afin d'avoir un retour visuel dans l'IHM.

```java
package algo.implem;

import af.Argument;
import af.ArgumentationFramework;
import algo.AbstractAlgorithm;

public class MonAlgo extends AbstractAlgorithm {

    public MonAlgo(){
        super("Mon super Algo");

        //Ajout d'un parametre
        this.addParam("threshold", -1, "Le nombre maximum d'iterations");

        //...
    }

    @Override
    public void init() {
        //Pré-traitement
        for(Argument arg : this.getGraph().getArguments()){
            System.out.println(arg.getId());
            //...
        }
    }

    @Override
    public void run() {
        //Boucle principale
        for(int i =0; i< (int) this.getParam("threshold").getValue(); i++){
            //...
        }
    }

    @Override
    public void end() {
        //On met à jour les utilités des arguments
        //en fonction de notre algorithme
        //pour l'affichage graphique
        for(Argument arg : this.getGraph().getArguments()){
            arg.setUtility(0);
        }
    }

    //Redéfinition de la methode d'ajout d'un réseau à l'algo courrant
    @Override
    public void setGraph(ArgumentationFramework af){
        super.setGraph(af);

        //On met à jour la valeure par défaut du paramètre threshold
        this.getParam("threshold").setValue(af.getArguments().size());
    }
}
```

FIGURE 6 – Code d'un algorithme customisé