Construction et représentation de graphes

Dominique Rossin
rossin@liafa.jussieu.fr

1 Introduction

Le but de ce projet est de développer un langage de description de graphes ainsi que la visualisation de ceux-ci.

Les graphes seront décrits par un langage où les variables seront des graphes et les fonctions seront des opérations entre graphes telles que la mise en parallèle, la mise en série ou encore la construction par module.

La partie visualisation sera prise en charge en utilisant un algorithme gravitationnel de manière à avoir un rendu agréable à l'oeil.

1.1 Description du projet

Ce projet se découpe donc en deux parties totalement distinctes à savoir:

- L'écriture de l'analyseur lexical et de l'analyseur syntaxique correspondant au langage de construction de graphe
- Le calcul de la position des sommets du graphe selon l'algorithme gravitationnel ainsi que sa représentation graphique.

Une interface graphique minimaliste pourra se composer uniquement:

- D'une fenêtre principale divisée en deux parties:
  - En haut, un espace de dessin de graphes
  - En bas, un champ de texte où l'on pourra rentrer des instructions de commande

2 Langage de description

2.1 Grammaire

Dans cette partie, nous expliquons brièvement le langage de description de graphes.

```
Programme ::= [Instruction]*

Instruction ::=
                SETG Identificateur = ExpressionGraphe;
                | AFFICHE ExpressionGraphe;
                | SET Identificateur = Expression;
                | PRINT Expression;
                | ;

ListeExpressionGraphe ::=
                ExpressionGraphe [, ExpressionGraphe]*

ExpressionGraphe ::=
                Kn(ExpressionEntiere)
                | Wn(ExpressionEntiere)
```

```
                | Pn(ExpressionEntiere)
                | Knp(ExpressionEntiere,ExpressionEntiere)
                | DUP(ExpressionGraphe)
                | Parallele(ListeExpressionGraphe)
                | Serie(ListeExpressionGraphe)
                | Module(ExpressionGraphe,ExpressionEntiere,ExpressionGraphe)
                | Identificateur

ExpressionEntiere ::=
                ProduitEntier + ExpressionEntiere
                | ProduitEntier - ExpressionEntiere
                | ProduitEntier

ProduitEntier ::=
               FacteurEntier * ProduitEntier
               | FacteurEntier / ProduitEntier
               | FacteurEntier

FacteurEntier ::=
               Entier
               | (ExpressionEntiere)
               | Identificateur
```

2.2 Description des commandes

- SETG ident = ExpressionGraphe; : Stocke le graphe ExpressionGraphe dans la variable ident
- AFFICHE ExpressionGraphe; : Dessine le graphe suivant l'algorithme donné plus loin
- SET ident = Expr; : Stocke l'entier Expr dans la variable ident
- PRINT expr; : Affiche l'expression
- Kn(exprE) : Représente le graphe complet à exprE sommets c'est à dire le graphe ayant exprE sommets et toutes les arêtes possibles.
- Wn(exprE) : Représente le graphe ayant un cycle de taille (exprE-1) dont tous les sommets sont reliés à un autre sommet central. (Une roue avec des rayons)
- Pn(exprE) : Représente la ligne à exprE sommets
- Knp(exprE1,exprE2) : Représente le graphe formé de deux ensembles de sommets de taille respective exprE1 et exprE2. Chaque sommet d'un ensemble est relié à tous les sommets de l'autre.
- DUP(exprG) : Duplique le graphe exprG (sommet et arêtes)
- Parallele(ListeExp) : Met les graphes en parallèle c'est-à-dire construit le graphe formé de l'union de tous les autres graphes
- Serie(ListeExpG) : Met les graphes en série c'est-à-dire construit le graphe qui est l'union de tous les graphes et où les sommets d'un graphe sont reliés à tous les sommets du graphe le précédant dans la liste et du graphe le suivant dans la liste.

- Module(expG,exp,expG2) : Remplace le `exp` ième sommet du graphe expG par le graphe expG2. Si le sommet était relié à un autre sommet alors tous les sommets du graphe expG2 seront reliés à lui.

3 Représentation graphique

Le problème de trouver une représentation pour le graphe qui soit le plus agréable et le plus compréhensible a conduit à de nombreux algorithmes. Une classe d'algorithmes est dite à ressort. En pratique, on met des ressorts entre chaque couple de sommets dont la longueur au repos et la force dépendent de la distance entre ces sommets dans le graphe. Ensuite, on laisse évoluer ce système pour essayer de trouver un point d'équilibre qui sera notre représentation.

Dans cette partie, nous utiliserons l'algorithme de Fruchterman et Reingold (Graph drawing by force directed placement) pour placer les sommets du graphe. Cet algorithme travaille dans une fenêtre de taille W (largeur) par L (hauteur).

Nous donnons ici une version en pseudo code de cet algorithme telle qu'elle est présentée dans l'article original.

Cet algorithme prend en entrée un graphe G représenté par ses sommets V et son ensemble d'arêtes E.

```
area := W*L;
G = (V,E) // Les sommets ont des positions initiales aléatoires
k := √(area/|V|);
function f_a(z) := begin return z^2/k end;
function f_r(z) := begin return k^2/z end;

for i := 1 to iterations do begin
   // Calcul des forces repulsives
   for v in V do begin
       // chaque sommet a deux vector .pos (position) et .disp (deplacement)
       v.disp := 0;
       for u in V do
          if (u # v) then begin // u relie a v
          Δ := v.pos - u.pos; // Δ est le vecteur difference
          v.disp := v.disp + (Δ / |Δ|) f_r(|Δ|);
       end
   end

   // Calcul des forces attractives
   for e in E do begin
      // Chaque arete est une paire ordonnee de sommets .v et .u
      Δ := e.v.pos − e.u.pos;
      e.v.disp := e.v.disp - (Δ/|Δ|) f_a|Δ|;
      e.u.disp := e.u.disp + (Δ/|Δ|) f_a|Δ|;
   end

   // Limiter le deplacement maximal a la temperature t
   // et interdire de sortir de l ecran
   for v in V do begin
      v.pos := v.pos + (v.disp/|v.disp|) min(v.disp,t);
      v.pos.x := min(W/2,max(-W/2,v.pos.x));
      v.pos.y := min(L/2,max(-L/2,v.pos.y))
   end
```

```
    // reduit la temperature
    t := cool(t)
end
```

Pour la fonction de température (paramètre t, fonction cool) on peut prendre comme valeur initiale W/10 et comme évolution (cool(t) = W/(10 * (1+K*numeroIteration)))

Vous pouvez faire des essais avec d'autres fonctions pour voir les différences en terme de vitesse et de rendu.

4 Extensions

4.1 Choix des fonctions f_a et f_r

Vous pouvez grâce à un menu où encore mieux en rajoutant des commandes à votre langage prévoir de définir d'autres fonctions permettant le calcul des forces d'attraction et de répulsion pour la phase de calcul des positions des sommets du graphe.

4.2 Placement semi-automatique

On peut imaginer que l'utililsateur ait envie de placer des sommets à des endroits précis. Pour cela, il lui suffirait de déplacer le sommet à l'aide de la souris à l'écran. Il faut alors tenir compte de cette position des sommets lors du calcul des forces d'attraction.

4.3 Sauvegarde du graphe en format latex

Le programme devra exporter le graphe dans un format comme pstricks.