

Visualisation d'arbres de grandes tailles

Rapport de PSTL

Érika Baëna
erika.baena@etu.upmc.fr

Diana Malabard
diana.malabard@etu.upmc.fr

Antoine Genitrini (encadrant)
antoine.genitrini@lip6.fr

May 8, 2014

Contents

1	État des lieux	2
2	Étude préliminaire	3
2.1	TikZ	3
2.1.1	500 nœuds	3
2.1.2	10000 nœuds	4
2.2	Asymptote	4
2.2.1	500 nœuds	4
2.2.2	10000 nœuds	4
2.3	NetworkX accompagné de Matplotlib	5
2.3.1	500 nœuds	5
2.3.2	10000 nœuds	5
2.4	Conclusion	5
3	Choix d'implémentation	6
3.1	Fonctionnement général	6
3.2	Plus en détails	6
3.2.1	Structure d'arbre	6
3.2.2	Fonctionnement des algorithmes de parsing	7
3.2.3	Fonctionnement de l'algorithme de calcul de coordonnées	7
3.2.4	Fonctionnement de la génération du code	7
3.3	Complexité	8
4	Conclusion	9
4.1	Bilan	9
4.2	Pour la suite	9
A	Images obtenues lors de l'étude préliminaire	10
B	Extraits de l'implémentation	16

Abstract

Des outils existent actuellement pour représenter des arbres de grande taille de façon efficace. Citons par exemple GraphViz. L'inconvénient d'un tel outil est qu'il ne prend pas en compte l'ordre des fils. Ceci pose problème lorsque l'on souhaite représenter des arbres dont l'ordre des fils est primordial : les arbres de recherche.

Ce projet consiste à fournir une alternative à Graphviz, afin de pouvoir visualiser n'importe quel type d'arbre, toujours de manière efficace, mais en conservant l'ordre des fils. Ce problème possède plusieurs problématiques. Tout d'abord, nous voulons que l'affichage d'un arbre soit faite de manière élégante. Ensuite, il faut que le calcul de la mise en page de l'arbre soit rapide.

Pour ce faire, nous avons donc dû étudier les algorithmes déjà existants pour la mise en page élégante des arbres de grande taille. Sachant ces algorithmes, nous avons conçu un algorithme permettant cette mise en page. Enfin, nous avons implémenté cet algorithme, de telle façon qu'il puisse être utilisé avec différentes sorties. Nous avons ici choisi de considérer trois sorties possibles : Tikz, pour pouvoir générer automatiquement un pdf ou intégrer le code à un document LaTeX ; Asymptote, une alternative à Tikz ; et NetworkX, pour pouvoir générer une image de l'arbre que l'on pourra ultérieurement insérer dans n'importe quel document.

Chapter 1

État des lieux

Les articles que nous avons étudié concernaient avant tout la visualisation des arbres binaires de grande taille sans prendre en compte l'ordre des fils. Il en ressort néanmoins plusieurs principes :

1. Les arêtes de l'arbre ne doivent pas s'intersecter.
2. Les nœuds de même profondeur doivent être dessinés sur la même ligne horizontale.
3. Les arbres doivent être dessinés de la manière la plus compacte possible.
4. Un nœud parent doit être centré par rapport à ses fils.
5. Un sous-arbre doit être dessiné de la même façon, peu importe où il est placé dans l'arbre.
6. Les nœuds fils d'un nœud père doivent être espacés de manière homogène.

Un des enjeux soulevés par l'ensemble de ces principes est que les nœuds ne doivent pas se chevaucher, tout comme les arbres en eux-même.

Il existe plusieurs algorithmes permettant de dessiner des arbres de grande taille, mais tous ne respectent pas tous les principes décrits ci-dessus.

Par exemple, l'algorithme le plus simple, l'algorithme de Knuth, ne respecte que les deux premiers principes. Cet algorithme décrit déjà une idée de "slot disponible". Mais ce sont Charles Wetherell et Alfred Shannon, en 1979, qui introduiront l'utilisation d'un tableau qui associera à chaque profondeur le prochain slot disponible. Il arrivent ainsi à respecter tous les principes, à l'exception des principes 4 et 5. Ils introduisent malgré tout également l'idée de parcourir l'arbre de bas en haut, plutôt que l'inverse, ceci afin de centrer facilement un père selon ses fils.

Le principal problème est alors : Comment respecter tous ces principes et traiter le chevauchement d'arbres sans perdre en complexité ? C'est l'algorithme The Mods and the Rockers qui répondra à cette question. Au lieu de reparcourir les sous-arbres pour les décaler afin d'éviter tout chevauchement, on raisonne en deux passes de l'arbre. Lors de la première passe, on mémorise un "modifier" qui indiquera le décalage qui devra être appliqué sur chaque sous-arbre lors de la deuxième passe. Il existe enfin des algorithmes dont le but est surtout d'optimiser les concepts et algorithmes existants. Citons le concept de contour d'arbre, qui permet de ne parcourir que ce contour et donc de ne pas rentrer au cœur de l'arbre. Ceci est un gain conséquent puisque nous traitons ici des arbres de grande taille, et donc potentiellement très larges.

Nous allons donc voir à présent comment nous nous sommes inspirées de ces algorithmes afin de généraliser les concepts aux arbres n-aires, et en conservant l'ordre des fils.

Chapter 2

Étude préliminaire

Le but de cette étude préliminaire est de trouver un outil adapté à la représentation d'arbres de grande taille. Étudions les performances de TikZ, Asymptote et NetworkX pour la génération d'un cas particulier d'arbres: les chaînes.

2.1 TikZ

TikZ est un package L^AT_EX permettant la création de graphiques.

On va utiliser le code Python suivant pour générer du code TikZ décrivant un arbre linéaire d'une taille passée en paramètre.

```
1 import sys
3 nbIte = int(sys.argv[1])
4 if (sys.argv[2] == "true"):
5     labels = True
6 else:
7     labels = False
8 i = 0
9
10 fileName = "testTikz%d" % (nbIte,)
11
12 if (labels):
13     fileName += ".tex"
14 else:
15     fileName += ".tex"
16
17 fichierTest = open(fileName, "w")
18
19 if (labels):
20     fichierTest.write("\\node (a%d) at (0,%d) {$%d$};\n" % (i, i, i))
21     i += 1
22     while i < nbIte:
23         fichierTest.write("\\node (a%d) at (0,%d) {$%d$};\n" % (i, i, i))
24         fichierTest.write("\\draw (a%d) — (a%d);\n" % (i-1, i))
25         i += 1
26 else:
27     i += 1
28     while i < nbIte:
29         fichierTest.write("\\draw (0,%d) — (0,%d);\n" % (i-1, i))
30         i += 1
31
32 fichierTest.close()
```

2.1.1 500 nœuds

On utilise le code précédent pour générer un arbre linéaire de taille 500. On insère le code obtenu dans un fichier L^AT_EX pour voir le résultat. Le fichier L^AT_EX compile et le résultat est aux figures A.1 et A.2.

2.1.2 10000 nœuds

On utilise maintenant le même code mais pour avoir un arbre de 10000 nœuds. Le fichier L^AT_EX ne compile plus. On obtient l'erreur **dimension too large** à la ligne:

```
\node (a576) at (0,576) {$576$};
```

Si l'arbre est trop grand, essayons de réduire sa taille: on diminue l'échelle, on diminue la distance entre deux points et on supprime les labels. L'erreur persiste au même endroit. TikZ limite notre arbre à 575 nœuds à la verticale. Peut-être la limite serait-elle différente si les nœuds étaient répartis autrement.

2.2 Asymptote

Asymptote est un langage de description de dessins vectoriels. Un package permet de le compiler dans un fichier L^AT_EX mais le code asymptote peut aussi être autonome.

On va utiliser le code Python suivant pour générer du code Asymptote décrivant un arbre linéaire d'une taille passée en paramètre.

```
1 import sys
3 nbIte = int(sys.argv[1])
4 if (sys.argv[2] == "true"):
5     labels = True
6 else:
7     labels = False
9 i = 0
11 fileName = "testAsymptote%d" % (nbIte,)
12 if (labels):
13     fileName += ".tex"
14 else:
15     fileName += ".tex"
17 fichierTest = open(fileName, "w")
18 fichierTest.write("\\begin{asy}\n")
19 fichierTest.write("size(20cm,20cm);\n")
21 if (labels):
22     fichierTest.write("label(\"a%d\", (0, %d), E);\n" % (i, i))
23 i += 1
25 while i < nbIte:
26     if (labels):
27         fichierTest.write("label(\"a%d\", (0, %d), E);\n" % (i, i))
28         fichierTest.write("draw((0, %d) — (0, %d));\n" % ((i-1), i))
29         i += 1
31 fichierTest.write("\\end{asy}\n")
33 fichierTest.close()
```

2.2.1 500 nœuds

On commence doucement en générant un arbre de 500 nœuds. On insère le code obtenu dans un fichier L^AT_EX comme précédemment. Le fichier compile et le résultat obtenu est celui des figures A.3 et A.4.

2.2.2 10000 nœuds

On recommence en mettant la barre à 10000 nœuds. Le fichier compile sans problème et le résultat est celui des figures

2.3 NetworkX accompagné de Matplotlib

NetworkX est une bibliothèque Python pour l'étude des graphes, conçue pour fonctionner sur des grands graphes.

Matplotlib est aussi une bibliothèque Python mais qui permet quant à elle de générer une image 2D dans différents formats de sortie possible comme par exemple un png, un pdf ou un svg.

On va utiliser le code Python suivant pour générer du code NetworkX décrivant un arbre linéaire d'une taille passée en paramètre.

```
import matplotlib.pyplot as plt
2 import networkx as nx
import sys
4
nbIte = int(sys.argv[1])
longueur_arete = float(sys.argv[2])
6 G = nx.path_graph(nbIte)
8 pos={x: (5, x*longueur_arete) for x in G.nodes()}
nx.draw(G, pos, node_size=5, with_labels=False)
10 plt.savefig("networkx_%d_nodes.png" % nbIte)
```

2.3.1 500 nœuds

De même que précédemment, on génère d'abord un arbre de 500 nœuds. On choisit le format de sortie png. Le résultat est visible aux figures

2.3.2 10000 nœuds

Passons maintenant à 10000 nœuds. Le résultat est aux figures

2.4 Conclusion

TikZ permet une représentation claire d'un arbre avec ses labels. En effet, même avec 500 nœuds, les labels sont lisibles si on zoome suffisamment. Cependant, une limite a rapidement été atteinte. TikZ serait préférable pour la représentation de petits arbres avec (ou sans!) labels.

Asymptote permet de représenter de grands arbres. Son point faible est la représentation des labels. Cependant, les labels sont compilés avec L^AT_EX_{ce} qui peut permettre d'avoir des labels un peu plus évolués qu'une chaîne de caractères, une fois la taille des labels maîtrisée.

NetworkX et Matplotlib permettent plusieurs formats de sortie différents. Cela pourrait être utile aux non-utilisateurs de L^AT_EX. Cependant, l'affichage des labels n'est pas non plus très optimal.

Notons tout de même que cette étude préliminaire ne prend pas en compte le temps de calcul des coordonnées, ce qui est le cœur de notre projet et qui est expliqué dans le chapitre suivant.

Chapter 3

Choix d'implémentation

3.1 Fonctionnement général

L'utilisateur fournit un fichier et précise son type. Il peut aussi choisir d'afficher des labels sur les nœuds ainsi que le format de sortie souhaité. Par défaut, l'application génère un png sans labels. L'application fonctionne ensuite en trois étapes:

1. Parsing du fichier d'entrée selon le type indiqué pour obtenir une représentation d'arbre selon notre structure interne.
2. Calcul des coordonnées de chaque nœud.
3. Génération d'une image selon le type de sortie choisie.

```
erika@erika-K53SD:~/Documents/Cours/M1S2/PSTL/Implementation$ python3 treeDisplay.py -h
usage: treeDisplay.py [-h] [-L] [-O {tikz,asy,png,pdf,eps}] [-N NAME]
                    {str,arb,xml,dot} src

positional arguments:
  {str,arb,xml,dot}  The type of the given file.
  src                The file which contains a tree description.

optional arguments:
  -h, --help            show this help message and exit
  -L, --labels          Print labels on the output tree, deprecated with
                        NetworkX and Asymptote
  -O {tikz,asy,png,pdf,eps}, --output {tikz,asy,png,pdf,eps}
                        The type of the output file
  -N NAME, --name NAME The name of the output file
erika@erika-K53SD:~/Documents/Cours/M1S2/PSTL/Implementation$
```

3.2 Plus en détails

3.2.1 Structure d'arbre

```
Éë
2 This file is part of TreeDisplay.

4     TreeDisplay is free software: you can redistribute it and/or modify
5     it under the terms of the GNU General Public License as published by
6     the Free Software Foundation, either version 3 of the License, or
7     (at your option) any later version.

8
9     FooBar is distributed in the hope that it will be useful,
10    but WITHOUT ANY WARRANTY; without even the implied warranty of
11    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12    GNU General Public License for more details.

13
14    You should have received a copy of the GNU General Public License
15    along with FooBar. If not, see <http://www.gnu.org/licenses/>.
```

3.2.2 Fonctionnement des algorithmes de parsing

Mots bien parenthésés Le parser de mots bien parenthésés (cf. B) respecte la grammaire suivante:

```
ARBRE : '(' ID NOEUDS ')'  
NOEUDS : ARBRE NOEUDS | e  
ID : [a-zA-Z1-9]* | e
```

Dot Le parser dot parse une sous-partie du langage dot, à savoir:

```
DOT : STRICT GRAPH ID '{' SEQINST '}'  
STRICT : strict | e  
GRAPH = diagraph | graph  
SEQINST : INST SEQINST | e  
INST : REF [label = "ID"] ';' | REF LINK REF  
LINK : -- | ->  
REF : [0-9]*  
ID : [a-zA-Z1-9]* | e
```

XML Le parser XML utilise la librairie XML de Python. Il respecte la grammaire suivante:

```
XML : <?xml version="1.0"?><tree> NOEUDS </tree>  
NOEUDS : NOEUD NOEUDS | e  
NOEUD : <node type=TAG id=ID> NOEUDS </noeud> | <leaf type=TAG id=ID />  
TAG : "Leaf" | "BinNode"
```

3.2.3 Fonctionnement de l'algorithme de calcul de coordonnées

On décide que la distance minimale entre deux nœuds est de 1 unité.

L'ordonnée d'un nœud est triviale: c'est sa profondeur.

L'abscisse d'un nœud est un peu plus complexe et demande donc plus de réflexion. Pour un nœud donné, on commence par placer ses fils. On centre ensuite ce nœud au milieu de ses fils en faisant la moyenne de l'abscisse de ses deux fils extrêmes. Si un nœud n'a pas de fils, on le place à 1 de son frère gauche. D'un point de vue de l'architecture, on a donc besoin d'une structure qui, à profondeur p mémorise la prochaine place disponible (ou au choix la dernière place utilisée). Si un père a des fils, on le centre au milieu de ses fils. Par ce calcul, on peut se retrouver avec un nœud qui est trop proche de son frère gauche [Mettre un exemple]. Pour cela, on compare la position calculée avec la première position disponible et on prend le max. Si la position calculée n'est pas celle retenue, le père n'est plus centré au milieu de ses fils. On mémorise donc le décalage effectué pour ce père pour ensuite l'appliquer à ses sous-arbre dans un second temps.

1. Pour un nœud donné, on commence par placer ses fils.
2. On centre ensuite ce nœud au milieu de ses fils.
3. Si un nœud collisionne avec son frère gauche, on le décale vers la droite et on mémorise ce décalage car il faudra ensuite décaler ses sous-arbres. On applique ce décalage dans une seconde passe pour des raisons de complexité. En effet, si on faisait chaque décalage lorsqu'il se présentait, le décalage serait quadratique alors que dans le cas choisi, on est linéaire.

3.2.4 Fonctionnement de la génération du code

TikZ

Asymptote

Autre

3.3 Complexité

On suppose que la taille des labels est bornée. On note n le nombre de nœuds dans l'arbre. Dans ce cas:

1. Le parsing d'un fichier est en $O(n)$.
2. Le calcul des coordonnées est en $O(n)$ (on effectue 2 passes sur l'arbre).
3. La génération du fichier de sortie est en $O(n)$.

On a donc une complexité générale en $O(n)$ où n est le nombre de nœud de l'arbre.

Notons que nous avons supposé que la taille des labels était bornée. Or nous n'avons aucune prise sur la taille des labels du fichier qui nous est passé en entrée. Dans ce cas, même si on borne la taille des labels pour l'affichage, la complexité est dominée par le parsing du fichier d'entrée car on doit lire tous les caractères du fichier.

Chapter 4

Conclusion

4.1 Bilan

L'application `treeDisplay` permet de calculer les coordonnées des nœuds d'un arbre en temps linéaire par rapport à ce nombre de nœuds.

4.2 Pour la suite

La structure choisie permet aussi de représenter des graphes. On peut donc envisager par la suite d'implémenter un algorithme de calcul de coordonnées pour les graphes. Les modules de parsing et de génération sont pleinement réutilisables.

La complexité en mémoire est actuellement égale à celle en temps. Cependant, on utilisant une table de hachage pour les sous-arbres, on peut devenir sous-linéaire. L'idée consiste à calculer des coordonnées relatives et lorsqu'un arbre comporte deux (ou plus) sous-arbres identiques. Le sous-arbre en question n'est sauvegarder en mémoire qu'une seule fois et la deuxième fois on ne sauvegarde qu'une référence. On calcule ensuite les coordonnées absolues lors de la génération.

Appendix A

Images obtenues lors de l'étude préliminaire

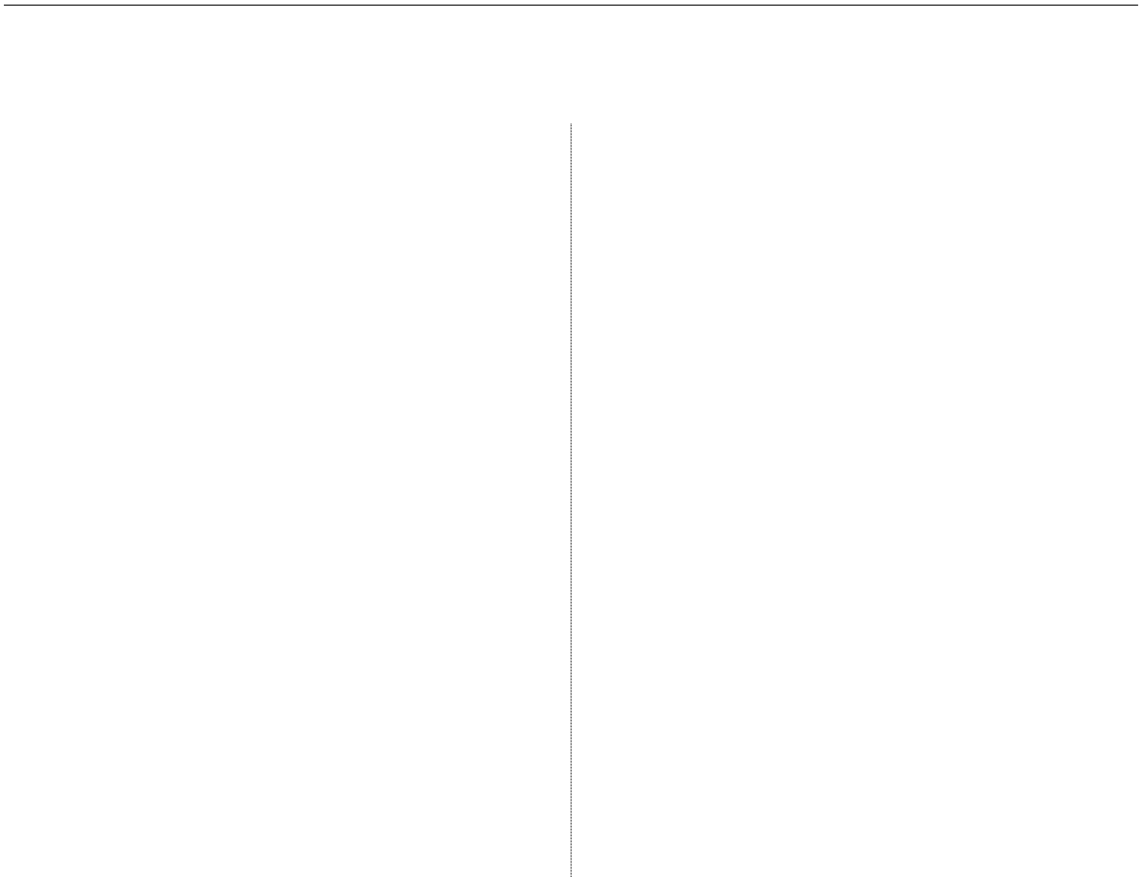


Figure A.1: Arbre linéaire de taille 500 avec labels obtenu avec TikZ.

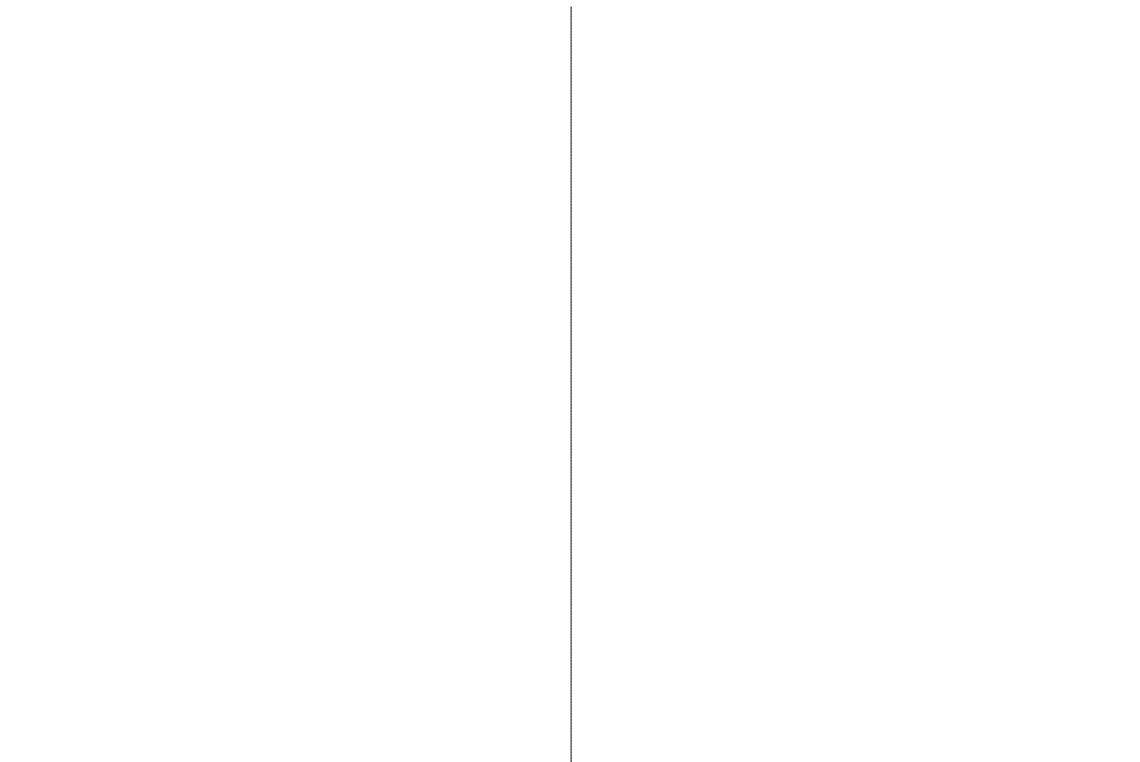


Figure A.2: Arbre linéaire de taille 500 sans labels obtenu avec TikZ.



Figure A.3: Arbre linéaire de taille 500 avec labels obtenu avec Asymptote.

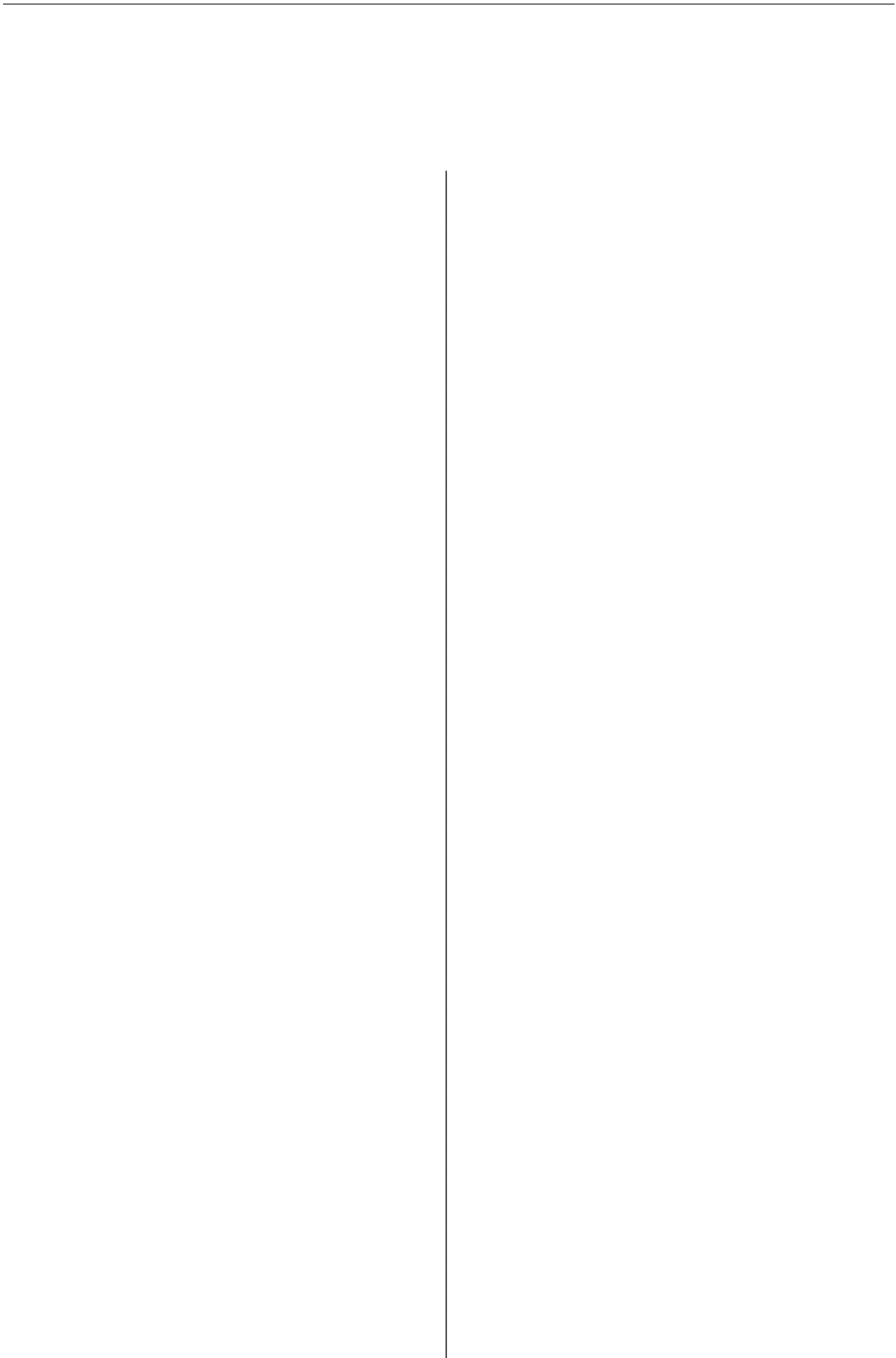


Figure A.4: Arbre linéaire de taille 500 sans labels obtenu avec Asymptote.



Figure A.5: Arbre linéaire de taille 10000 avec labels obtenu avec Asymptote.

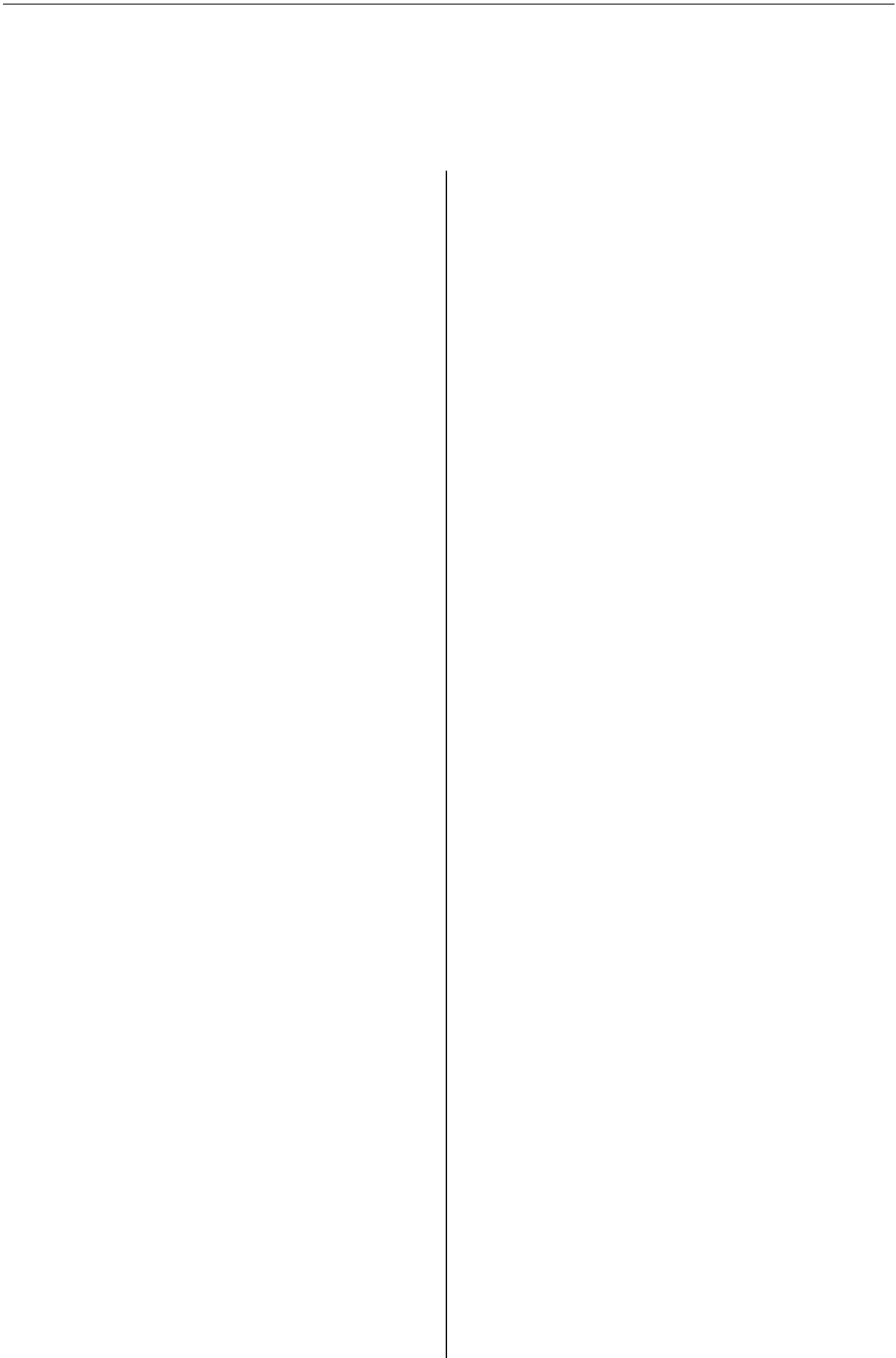


Figure A.6: Arbre linéaire de taille 10000 sans labels obtenu avec Asymptote.

Appendix B

Extraits de l'implémentation

```
1  """
2  Copyright (C) 2014 Érika ěBana et Diana Malabard
3
4  This file is part of TreeDisplay.
5
6      TreeDisplay is free software: you can redistribute it and/or modify
7      it under the terms of the GNU General Public License as published by
8      the Free Software Foundation, either version 3 of the License, or
9      (at your option) any later version.
10
11     Foobar is distributed in the hope that it will be useful,
12     but WITHOUT ANY WARRANTY; without even the implied warranty of
13     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
14     GNU General Public License for more details.
15
16     You should have received a copy of the GNU General Public License
17     along with Foobar. If not, see <http://www.gnu.org/licenses/>.
18 """
19
20 from tree import *
21 import re
22
23 nbCharAlire = 1024
24
25 def strParser (src):
26     "Parses a string file into the intern tree structure"
27
28     f=open(src, 'r')
29     s=f.read(nbCharAlire)
30     #(T, f) = parse(f, f.read(1))
31     T=parse(f, s)[0]
32     f.close()
33
34     return T
35
36 def parse (f, s, currentIndex=0):
37
38     isWhiteSpace=re.compile("[\s]")
39
40     (f, s, currentIndex) = skipSpaces(f, s, currentIndex)
41
42     if s[currentIndex]!='(':
43         raise Exception ("The tree is ill-formed : A tree should start with a '('")
44
45     (f, s, currentIndex) = incrementIndex(f, s, currentIndex)
46     size = len(s)
47     c = s[currentIndex]
48     label=""
49
50     (f, s, currentIndex) = skipSpaces(f, s, currentIndex)
51     if (s==[]):
52         raise Exception ("The tree is ill-formed : End of file reached")
53     c = s[currentIndex]
```

```

55 while (s!=[] and c!='(' and c != ')'):
56     #print("currentIndex=",currentIndex)
57     if (not isWhiteSpace.match(c)):
58         label = label + c
59         (f, s, currentIndex) = incrementIndex(f, s, currentIndex)
60         c = s[currentIndex]
61
62 (f, s, currentIndex) = skipSpaces(f, s, currentIndex)
63 c = s[currentIndex]
64
65 listChildren = []
66 while (s!=[] and c == '('):
67     (child, s, currentIndex) = parse (f, s, currentIndex)
68     listChildren.append(child)
69     #print("fils ",label,"éajout")
70     (f, s, currentIndex) = skipSpaces(f, s, currentIndex)
71     c = s[currentIndex]
72     #print("c èaprs avoir éajout le fils",label,": ",c)
73
74 (f, s, currentIndex) = skipSpaces(f, s, currentIndex)
75 c = s[currentIndex]
76 if (c == ')'):
77     (f, s, currentIndex) = incrementIndex(f, s, currentIndex)
78     return (Tree(label = label, children=listChildren), s, currentIndex)
79 else:
80     raise Exception ("The tree is ill-formed : The tree",label,"should end with a
81     ')")
82
83 def incrementIndex(f, s, currentIndex):
84     currentIndex+=1
85     l=len(s)
86     if (currentIndex>=l):
87         #print("fin du buffer -> on re-remplit")
88         s=f.read(nbCharAlire)
89         currentIndex=0
90     if (s==[]):
91         raise Exception ("The tree is ill-formed : End of file reached")
92     #print("currentIndex dans incrementIndex :",currentIndex)
93     return (f, s, currentIndex)
94
95 def skipSpaces(f, s, currentIndex) :
96     isWhiteSpace=re.compile("[\s]")
97     while (s!=[] and isWhiteSpace.match(s[currentIndex])):
98         (f, s, currentIndex) = incrementIndex(f, s, currentIndex)
99     return (f, s, currentIndex)

```

```

100 """
101 Copyright (C) 2014 Érika ëBana et Diana Malabard
102
103 This file is part of TreeDisplay.
104
105     TreeDisplay is free software: you can redistribute it and/or modify
106     it under the terms of the GNU General Public License as published by
107     the Free Software Foundation, either version 3 of the License, or
108     (at your option) any later version.
109
110     Foobar is distributed in the hope that it will be useful,
111     but WITHOUT ANY WARRANTY; without even the implied warranty of
112     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
113     GNU General Public License for more details.
114
115     You should have received a copy of the GNU General Public License
116     along with Foobar. If not, see <http://www.gnu.org/licenses/>.
117 """
118
119 from tree import *
120 from pyparsing import Word, alphas
121 import re
122
123 def dotParser (src):
124     "Parses a dot file into the intern tree structure"
125
126     f= open(src)

```

```

28 s = f.read()
30
31 (toto, i) = getWord (s, 0, '{') # [strict] (graph | digraph) [ID]
32
33 k=0
34
35 # [strict]
36 while (toto[k] == " "):
37     k += 1
38
39 if (toto[k] == 's'): #read "strict"
40     if re.compile('strict').search(toto):
41         if (re.search(u'strict', toto).start() != 0):
42             raise Exception ("Dot syntaxe error")
43         else:
44             k = re.search(u'strict', toto).end()
45     else:
46         raise Exception ("Dot syntaxe error")
47
48 # (graph|digraph)
49 while (toto[k] == " "):
50     k += 1
51
52 if (toto[k] == 'g'): #read "graph"
53     if re.compile('graph').search(toto):
54         if (re.search(u'graph', toto).start() != 0):
55             raise Exception ("Dot syntaxe error")
56         else:
57             k = re.search(u'graph', toto).end()
58     else:
59         raise Exception ("Dot syntaxe error")
60 elif (toto[k] == 'd'): #read "digraph"
61     if re.compile('digraph').search(toto):
62         if (re.search(u'digraph', toto).start() != 0):
63             raise Exception ("Dot syntaxe error")
64         else:
65             k = re.search(u'digraph', toto).end()
66     else:
67         raise Exception ("Dot syntaxe error")
68 else:
69     raise Exception ("Dot syntaxe error")
70
71 # [ID]
72 while (k<i and toto[k] == " "):
73     k += 1
74
75
76 i+= 1 #skip the {
77 print (s[i])
78
79 dico = {}
80 roots = set()
81 children = set()
82
83 try:
84     while (True):
85         (seq, i) = getWord (s, i, ';')
86         seq = re.sub(r'\s', "", seq)
87         j=0
88         if (s[i-1] == ']'):
89             # A node should be matched
90
91         (num, j) = getWord(seq, j, '[')
92         j += 1 # skip the '['
93
94         (etiquette, j) = getWord (seq, j, '=')
95         j += 1 # skip the '='
96
97         (label, j) = getWord (seq, j, ']')
98         j += 1 # skip the ']'
99
100

```

```

102         if (num == "" or etiquette != "label" or label[0] != '\ ' or
label[ len(label)-1] != '\ '):
            raise Exception ("Tree is ill-formed")

104         dico[num] = Tree (label=label[1:len(label)-1])
            roots.add(dico[num])

106     else:
108         # An arrow should be matched

110         (start , j) = getWord (seq , j , '-')
            j+=2 #skip -- or ->

112         end = seq[j:]

114         # Check if start and end are in dico
116         t = dico[start]
            c = dico[end]
118         if (t == None or c == None):
            raise Exception ("Tree is ill-formed")

120         # If start not in children add to roots , add end to children
122         #if t not in children:
            # roots.add (t)

124         if c in roots:
126             roots.remove (c)

128         children.add (c)
            #print (t , c , t.children , c.children)

130         #Add child to node
132         t.children.append(c)

134         i+=2 # skip the ; and go to the following item
    except IndexError:
136         seq = s[i:]
            seq = re.sub(r'\s', "", seq)
138         if (seq == "}"):
            # end of tree

140         #verify that there is only one root and found it for returning
142         if len(roots) != 1:
            raise Exception ("Tree is ill-formed")

144         f.close()
146         for x in roots:
            return x
148         #return roots.get()

150 #Exception indice out of array should have been raised
    f.close()
152     raise Exception ("Tree is ill-formed")

154 def getWord (s, start , end):
    "Return a substring of s that starts at indice start and ends with character end
    excluded. Return also the indice of character end."
156     res = ""
        j = start
158     while (s[j] != end):
        res = res + s[j]
160     j+=1
    return (res , j)

```

```

1 """
    Copyright (C) 2014 Érika ěBana et Diana Malabard
3
    This file is part of TreeDisplay.
5
        TreeDisplay is free software: you can redistribute it and/or modify
7        it under the terms of the GNU General Public License as published by
        the Free Software Foundation, either version 3 of the License, or
9        (at your option) any later version.

```

```
11     Foobar is distributed in the hope that it will be useful ,
13     but WITHOUT ANY WARRANTY; without even the implied warranty of
15     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
17     GNU General Public License for more details.
19
21     You should have received a copy of the GNU General Public License
23     along with Foobar. If not, see <http://www.gnu.org/licenses/>.
25 """
27
29 import xml.etree.ElementTree as etree
31 from tree import *
33
35 def xmlParser (src):
37     "Parses a xml file into the intern tree structure"
39
41     tree = etree.parse(src)
43     root = tree.getroot()[0] #getRoot returns the node with the "tree" tag ,
45                             #we want the "node" or "leaf" inside that node
47
49     return parse(root)
51
53 # Given an xml tree parsed by ElementTree, returns the corresponding Tree object
55 def parse(xmltree):
57
59     if(xmltree.tag=='leaf'):
61         # Case 1 : the tree is a leaf
63         # Try to get the id
65         l=xmltree.get('id')
67         if (l):
69             return Tree(label=l)
71         else:
73             return Tree()
75     else:
77         # Case 2 : the tree is a node with children
79         # Get the children
81         children=[]
83         for child in xmltree:
85             children.append(parse(child))
87         # Try to get the id of the node
89         l=xmltree.get('id')
91         if (l):
93             return Tree(label=l, children=children)
95         else:
97             return Tree(children=children)
```
