Visualisation d'arbres de grandes tailles [1]
Présentation de PSTL

Érika Baëna
erika.baena@etu.upmc.fr

Diana Malabard
diana.malabard@etu.upmc.fr

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

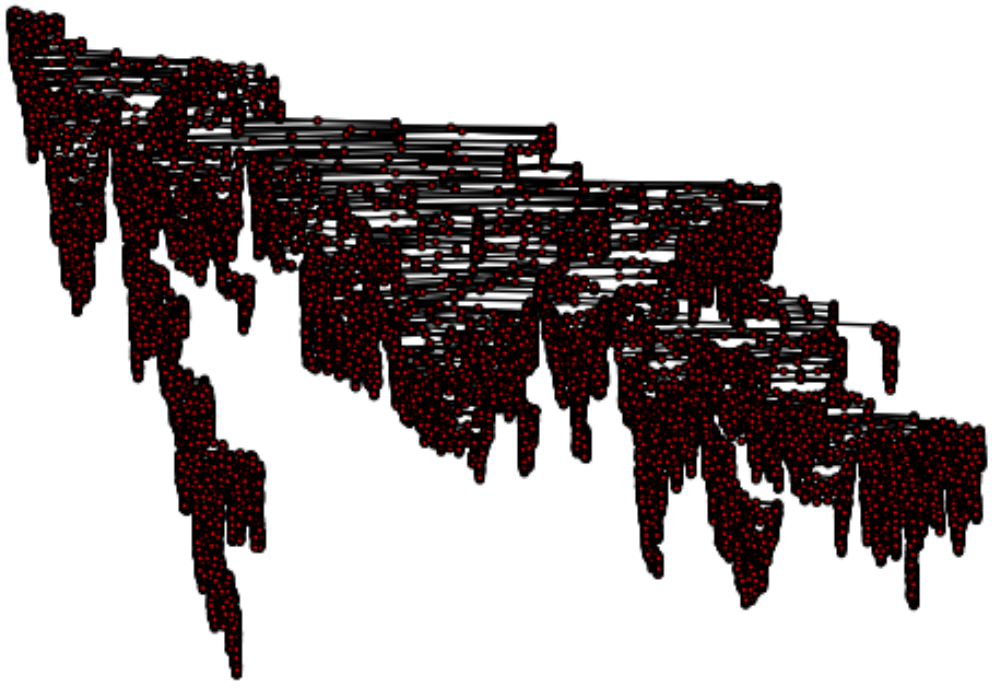
Université Pierre et Marie Curie

13 mai 2014

1. Disponible sur: `https://github.com/BErika/PSTL_TreeDisplay`

Plan de la présentation

Pourquoi ?



- Pourquoi des arbres ? Structure primordiale en informatique.
- Pourquoi afficher des arbres de grande taille ? Pour observer des tendances.
- Pourquoi une nouvelle application ? Nouveaux critères de représentation

Objectif

Objectif du projet

Affichage élégant et efficace de tout type d'arbre

Objectif

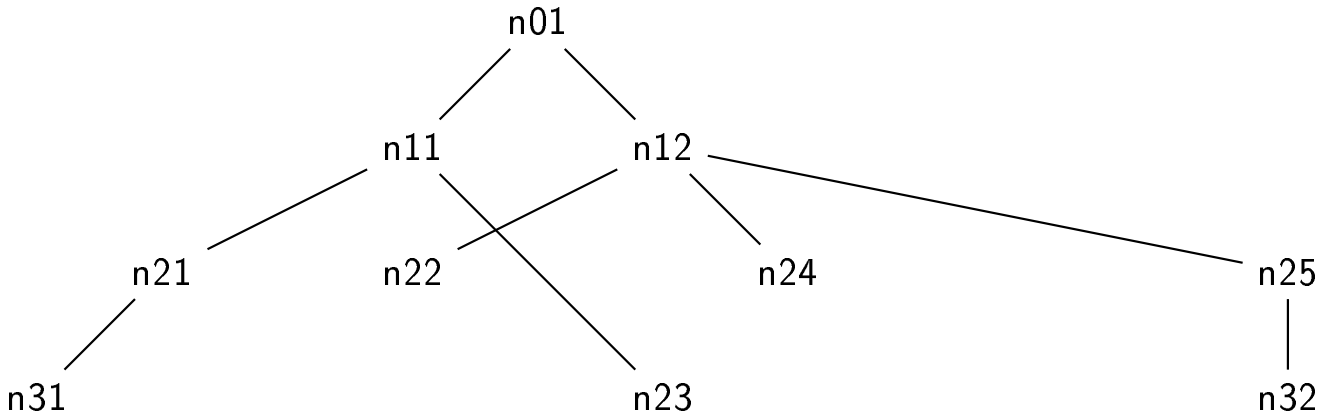
Objectif du projet

Affichage élégant et efficace de tout type d'arbre

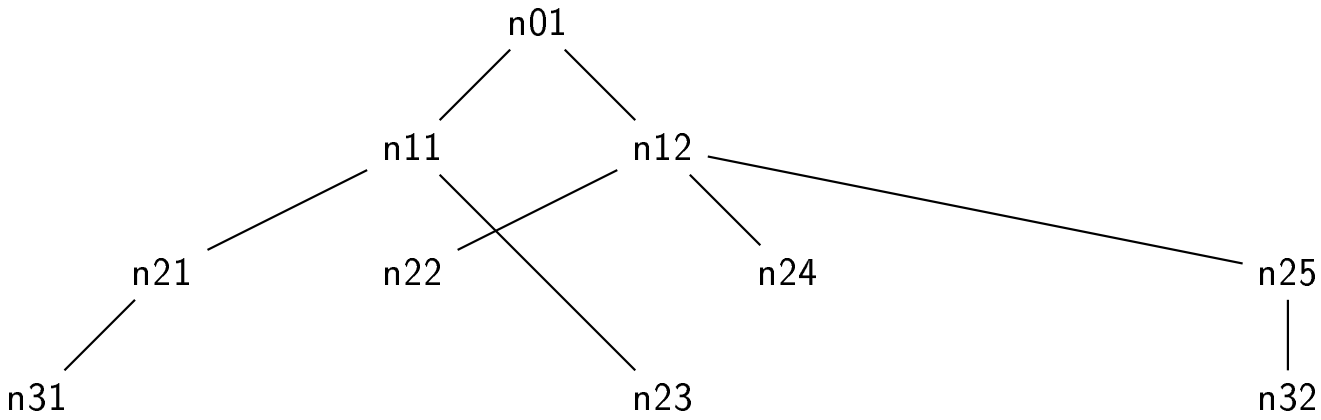
Problèmes

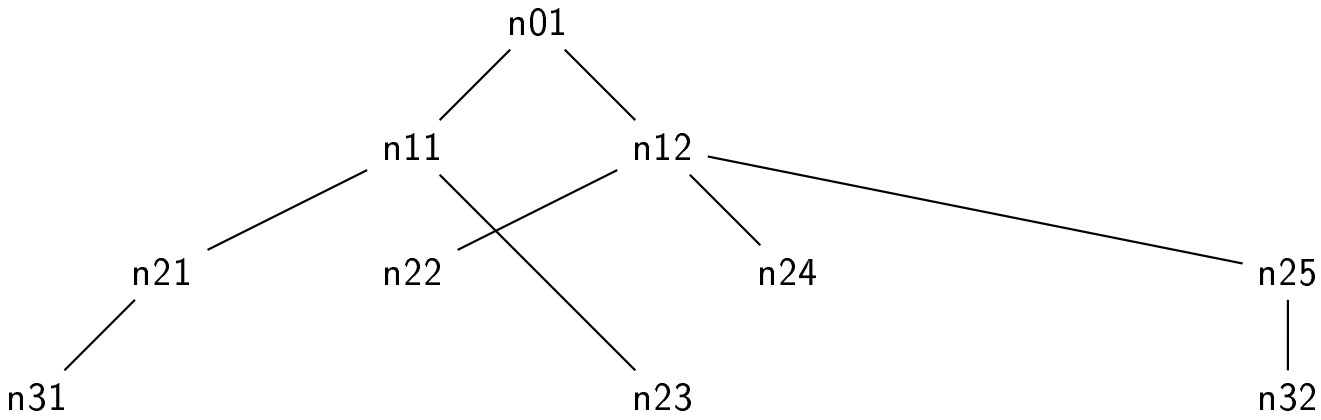
- Qu'est-ce qu'un affichage élégant ?
- Comment optimiser le calcul de la mise en page ?

Plan de la présentation



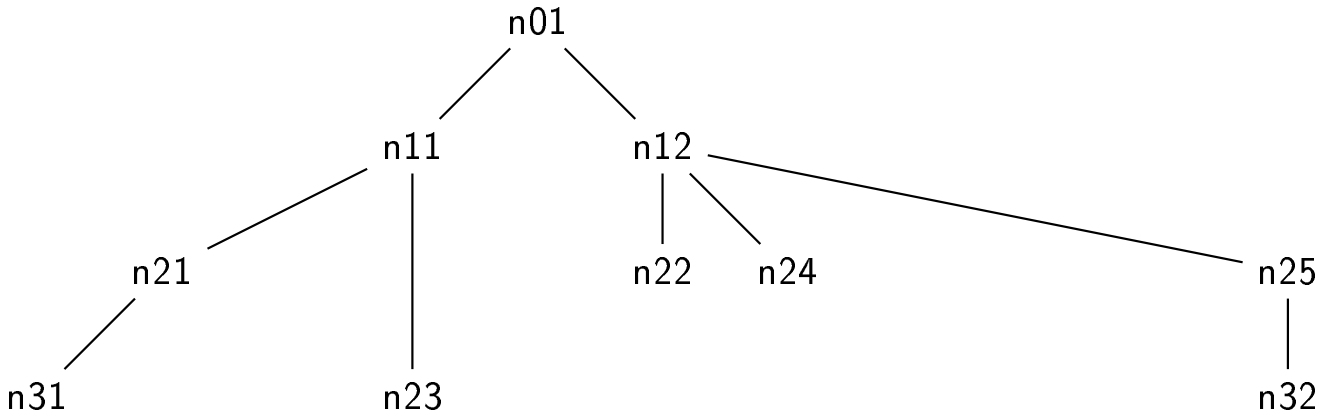
Que faire pour améliorer cet arbre ?

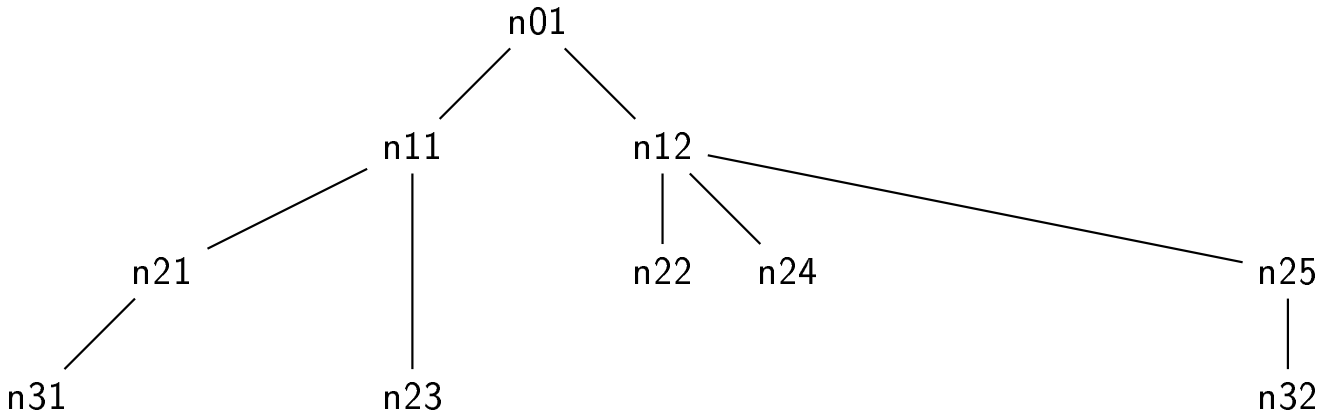

n01
n11
n12
n21
n22
n24
n25
n31
n23
n32



Principe 1

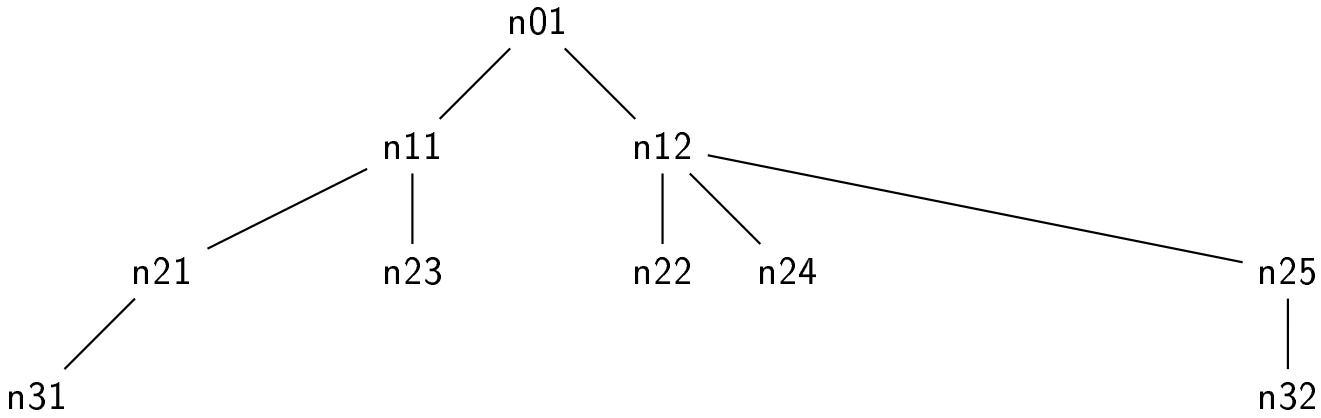
Les arêtes de l'arbre ne doivent pas s'intersecter.

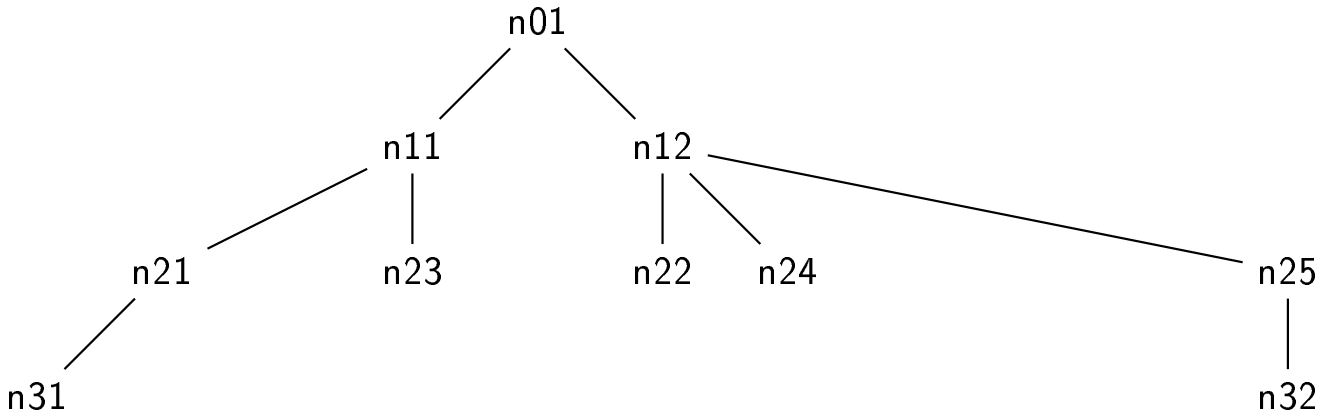

n01
n11
n12
n21
n22
n24
n25
n31
n23
n32



Principe 2

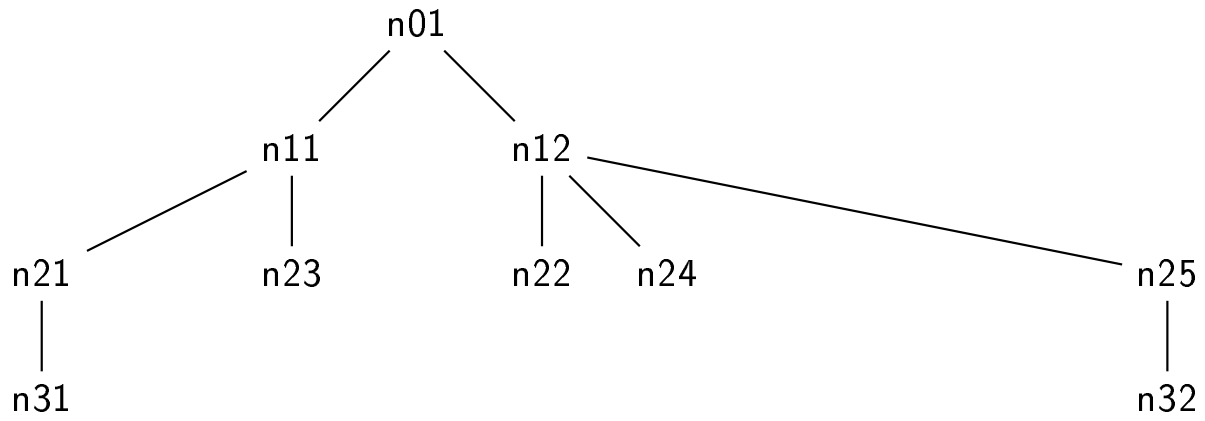
Les nœuds de même profondeur doivent être dessinés sur la même ligne horizontale.

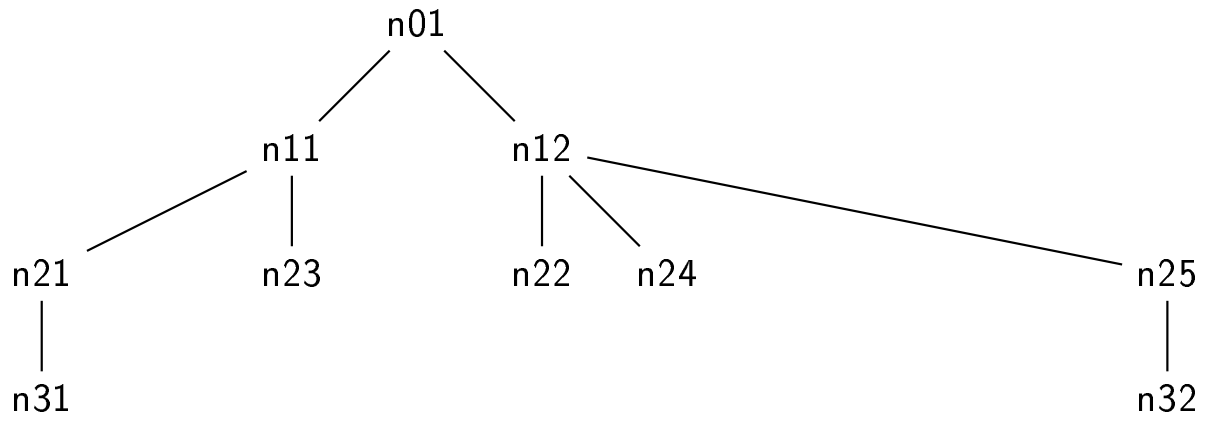

n01
n11
n12
n21
n23
n22
n24
n25
n31
n32



Principe 3

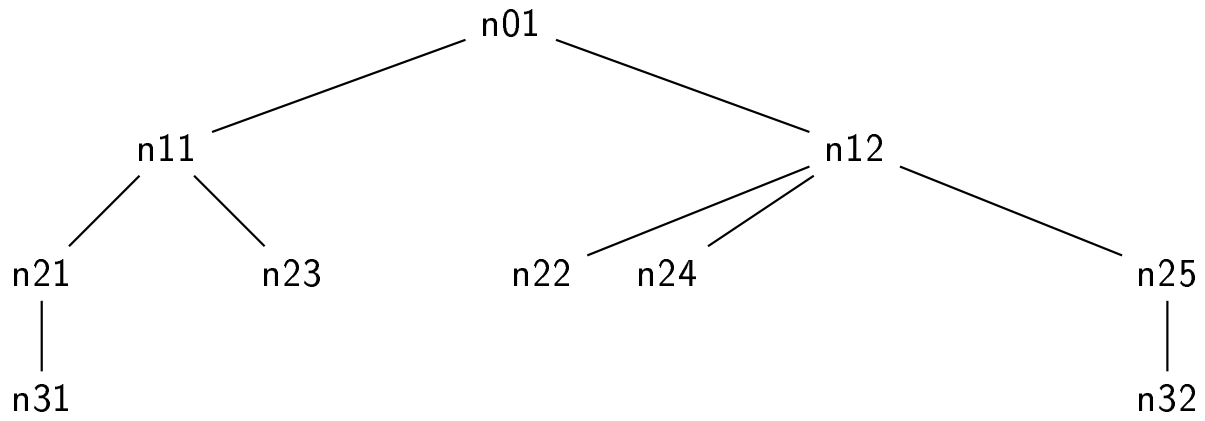
Un sous-arbre doit être dessiné de la même façon, peu importe où il est placé dans l'arbre.

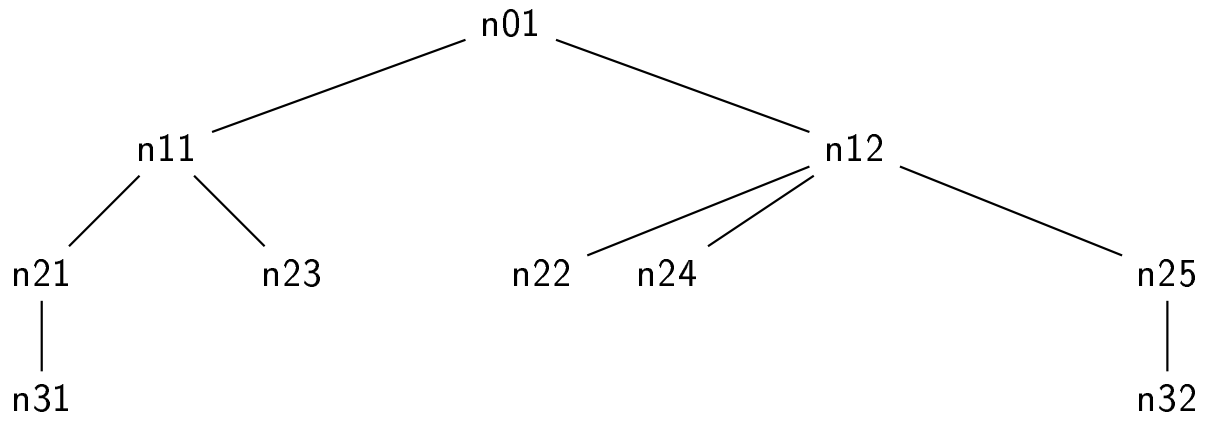

n01
n11
n12
n21
n23
n22
n24
n25
n31
n32



Principe 4

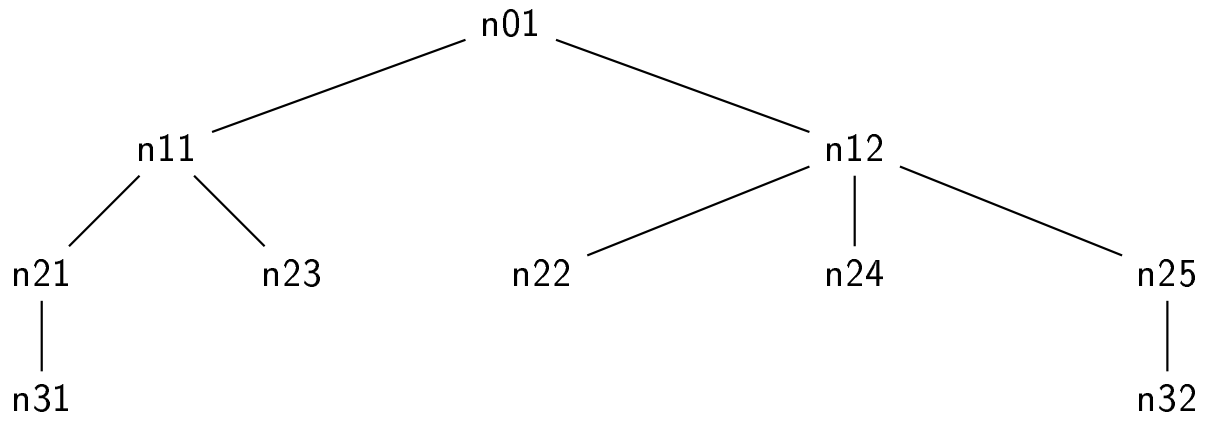
Un nœud parent doit être centré par rapport à ses fils.

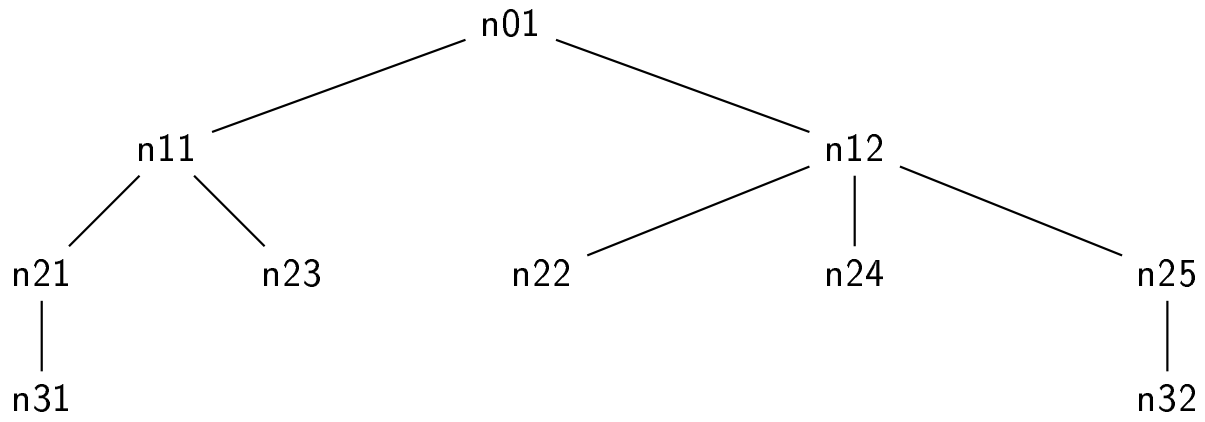

n01
n11
n12
n21
n23
n22
n24
n25
n31
n32



Principe 5

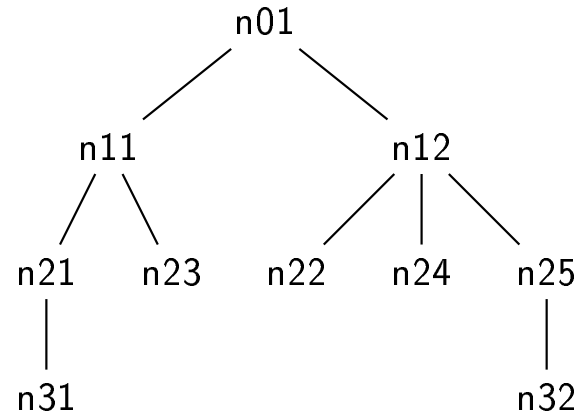
Les nœuds fils d'un nœud père doivent être espacés de manière homogène.


n01
n11
n12
n21
n23
n22
n24
n25
n31
n32



Principe 6

Les arbres doivent être dessinés de la manière la plus compacte possible.



Référence d'un arbre élégant.

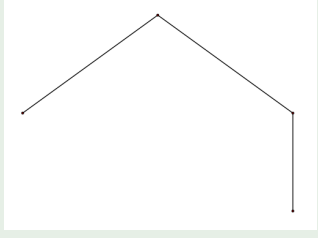
Plan de la présentation

Mots bien parenthésés

Grammaire respectée

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

Exemple

L'arbre  est représenté par ((()(()())).

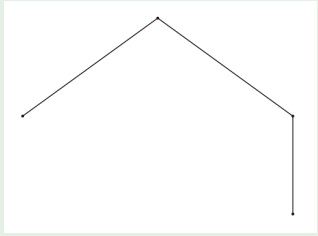
DOT I

Grammaire respectée

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

DOT II

Exemple

L'arbre  est représenté par :

```
digraph {
1 [label=""];
2 [label=""];
3 [label=""];
4 [label=""];
1 -> 2;
2 -> 3;
3 -> 4;
}
```

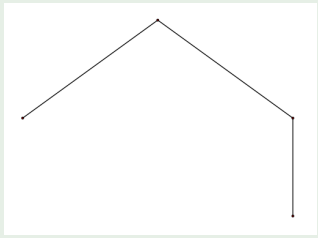
XML I

Grammaire respectée

```
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 : " [a-zA-Z1-9]* "
ID : [0-9]*
```

XML II

Exemple

L'arbre  est représenté par :

```
<?xml version="1.0"?>
<tree>
    <node type="" id=1>
        <leaf type="" id=2 />
        <node type="" id=3>
            <leaf type="" id=4 />
        </node>
     </node>
<tree>
```

Structure de données

```
1  class Tree(object):
     "Local representation of tree"
3
     def __init__(self, x = -1, depth=0, label="", children=None,
        offset = 0, isRoot = False):
5      self.x = x
       self.y = depth
7      self.label = label
       self.offset = offset
9      self.height = None
       self.width = None
11     if children is None:
         self.children = list()
13     else:
         self.children = children
```

Algorithme I
Passe 1

```
1   def setup (self, depth=0, nexts=None, offset=None):
      if nexts is None:
3       nexts = defaultdict(lambda:0)
      if offset is None:
5       offset = defaultdict(lambda:0)

7     # L'ordonnée est triviale, c'est la profondeur.
      self.y = depth
9
      # On calcule d'abord les coordonnées des enfants.
11    for c in self.children:
        c.setup(depth+1, nexts, offset)
13
      # On centre le noeud au milieu de ses enfants.
15    nbChildren = len(self.children)
      if (nbChildren == 0):
17      place = nexts[depth]
        self.x = place
19    else:
        place = (self.children[0].x +
      self.children[nbChildren-1].x) / 2
21
      # On calcule l'éventuel décalage engendré.
23    offset[depth] = max(offset[depth], nexts[depth]-place)
```

Algorithme II
Passe 1

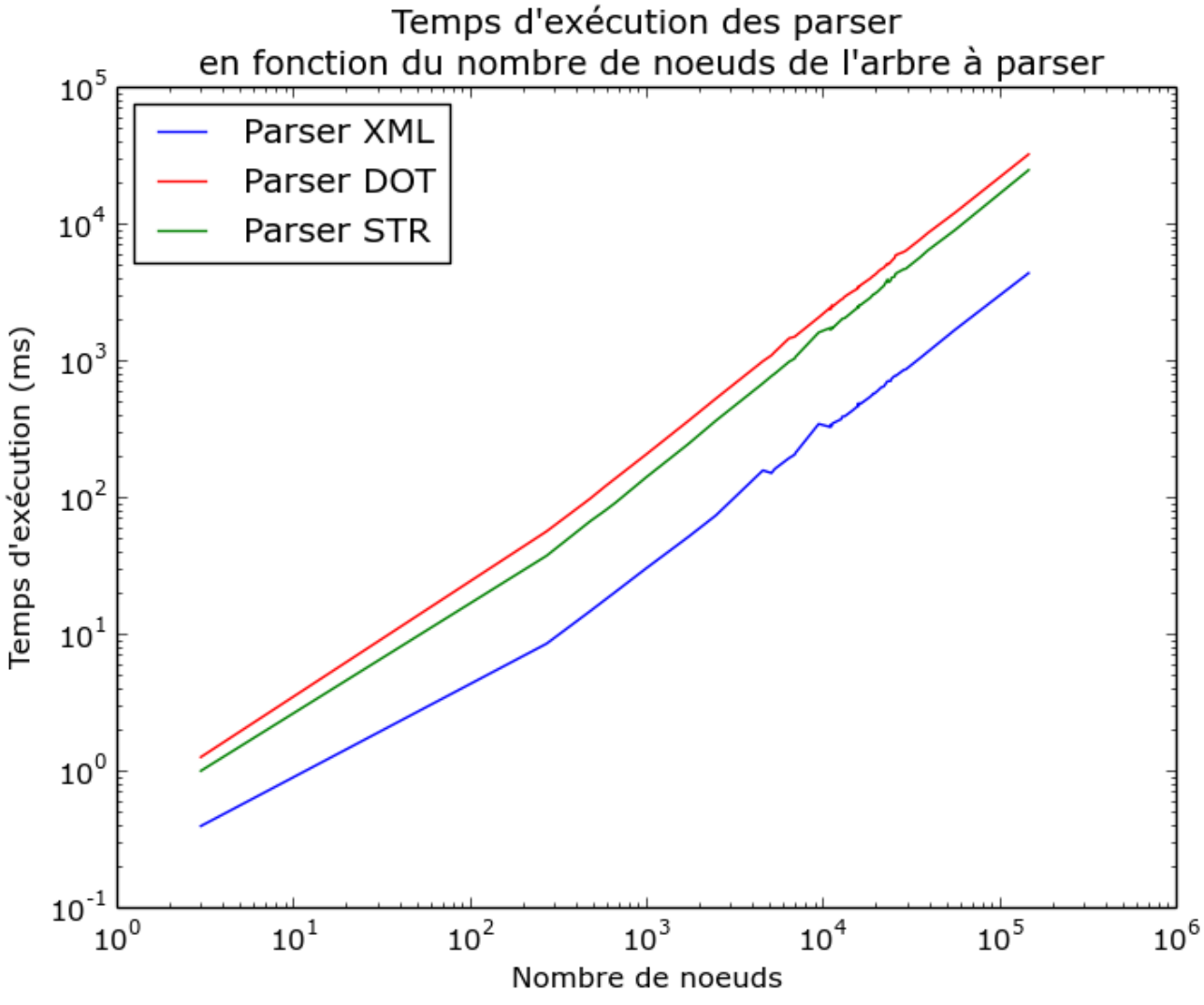
```
25      # On applique le décalage de la profondeur.
        if (nbChildren != 0):
27          self.x = place + offset[depth]

29      # On met é jour la prochaine place disponible é cette
         profondeur.
        nexts[depth] = self.x +1
```

Algorithme I
Passe 2

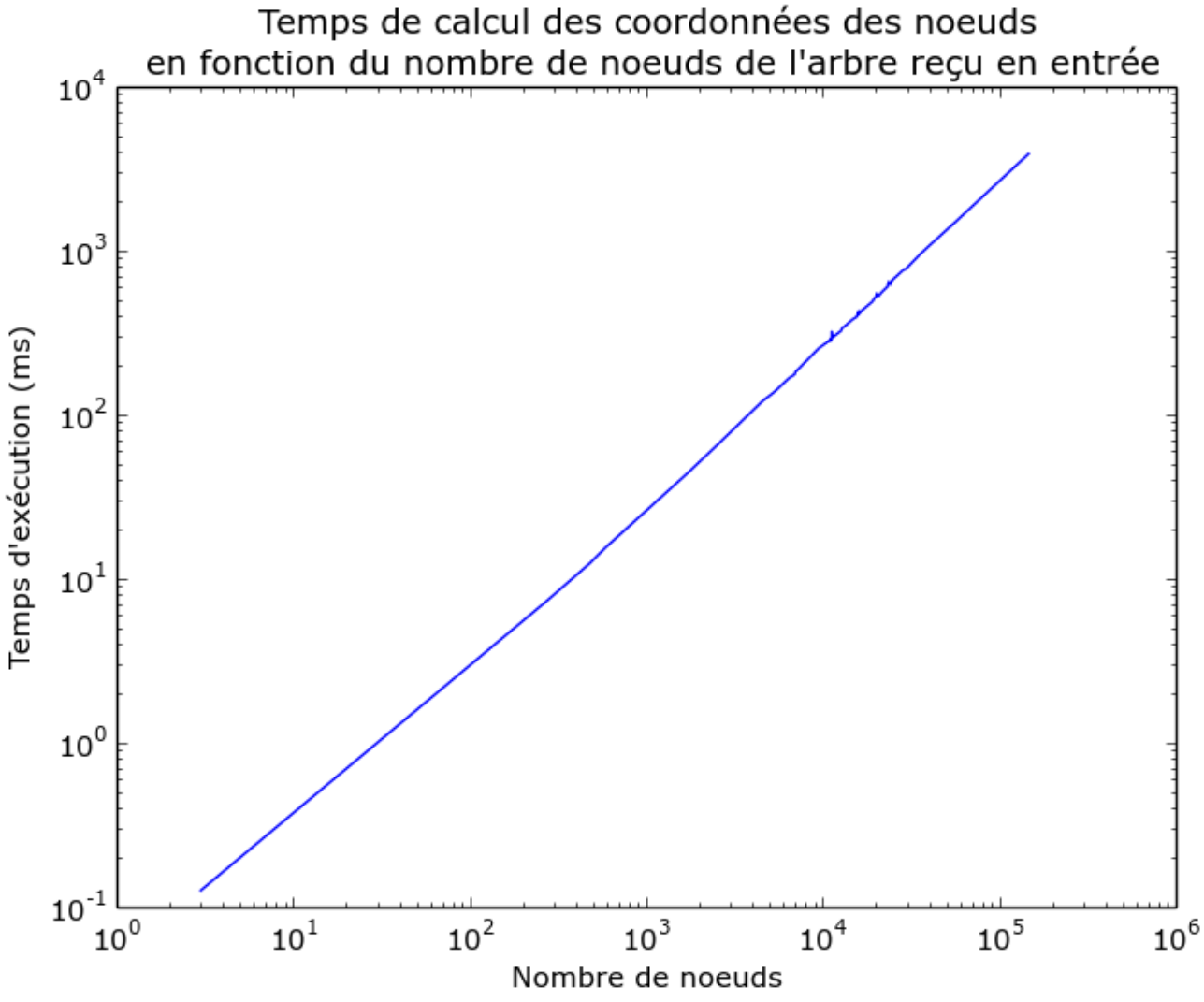
```
    def addOffsets(self, offsum=0):
2       self.x = self.x + offsum
        offsum = offsum + self.offset
4
        self.height = self.y
6       self.width = self.x

8       for c in self.children:
            c.addOffsets(offsum)
10          self.height = max(self.height, c.height)
            self.width = max(self.width, c.width)
```

Plan de la présentation

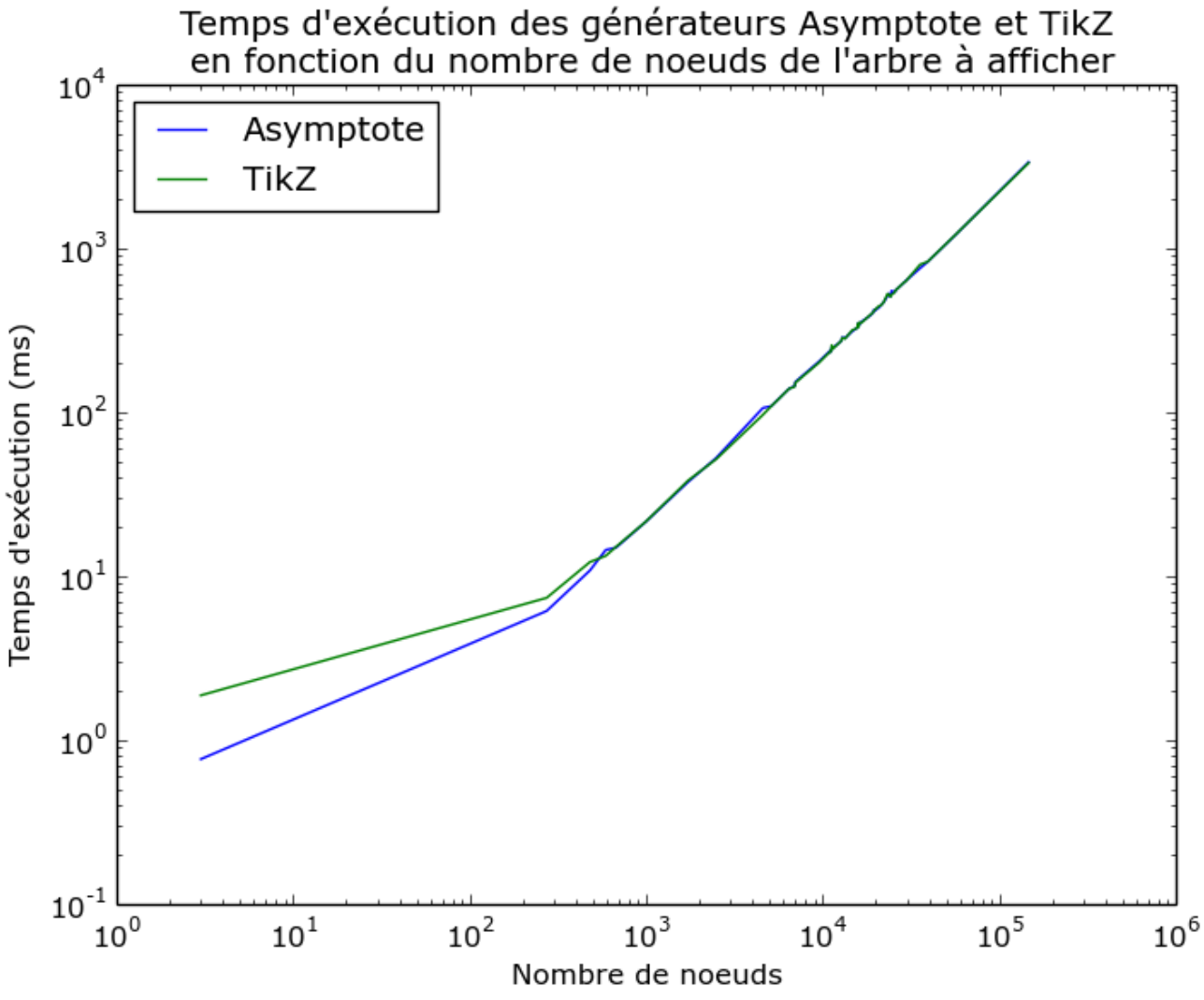
Les Parsers


Temps d'exécution des parser
en fonction du nombre de noeuds de l'arbre à parser
Parser XML
Parser DOT
Parser STR
Temps d'exécution (ms)
Nombre de noeuds
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}
10^{6}

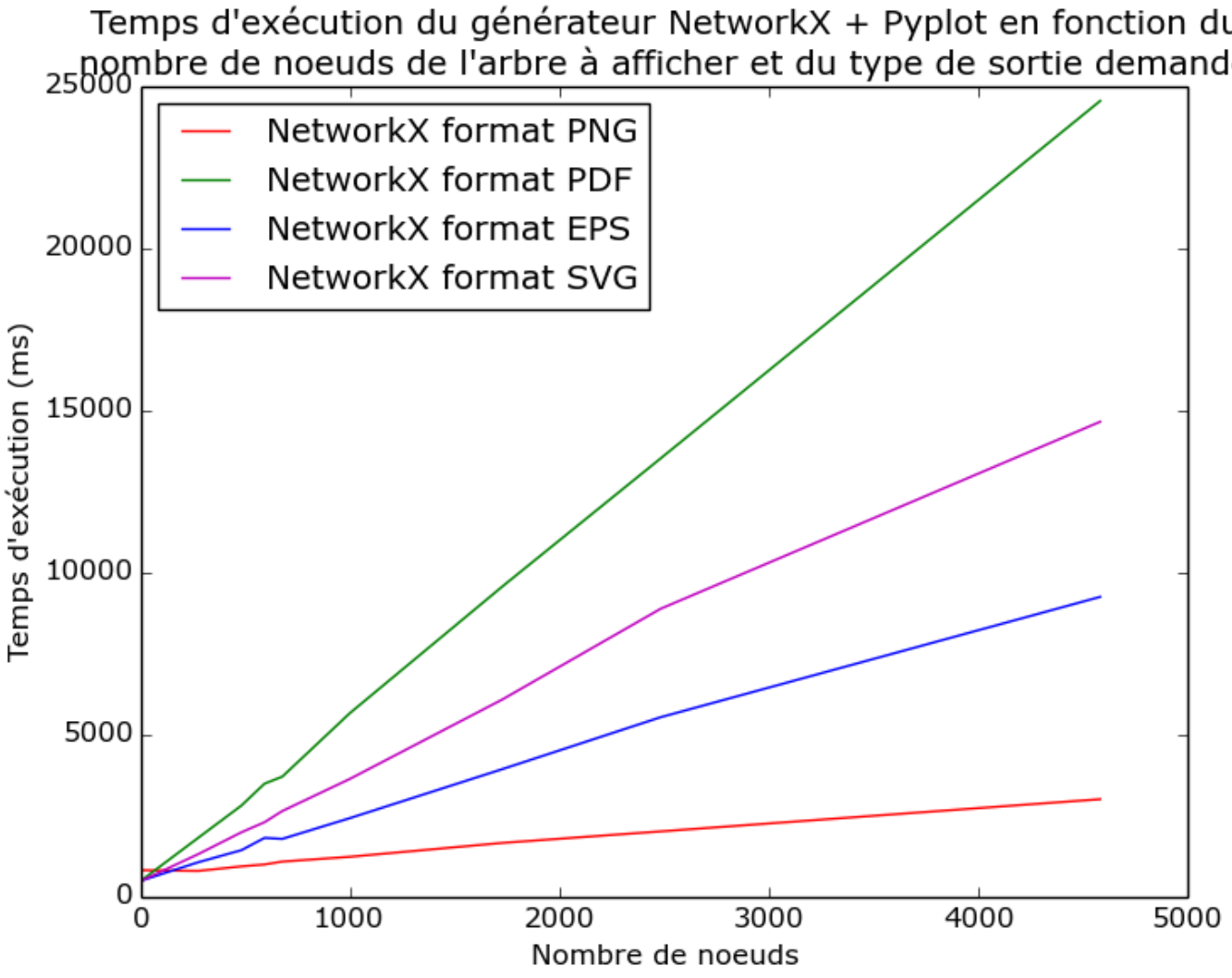
Calcul des coordonnées



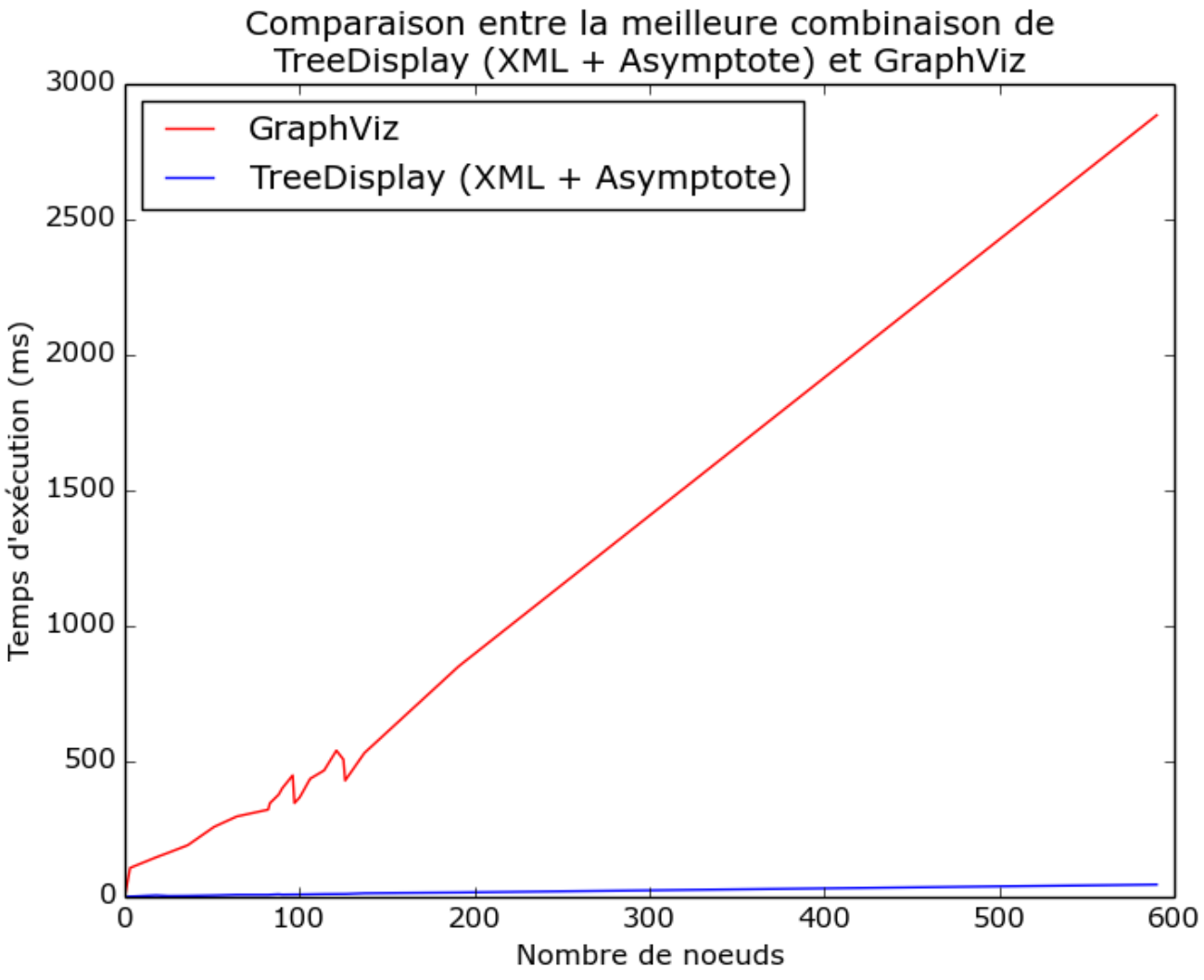
Générateurs I



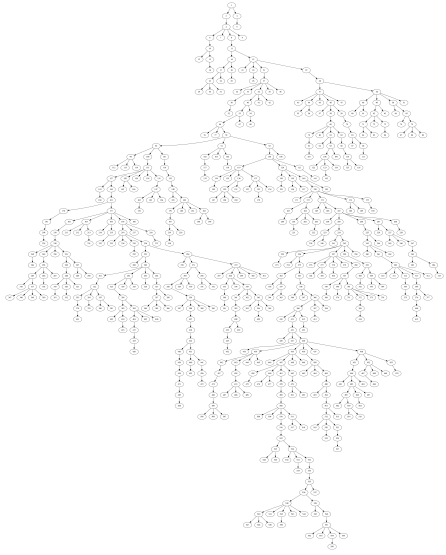
Générateurs II



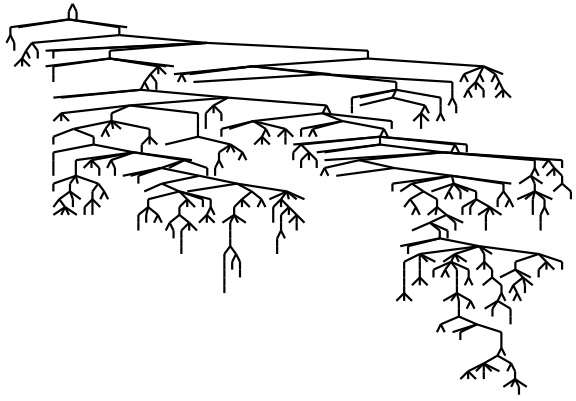
GraphViz vs TreeDisplay



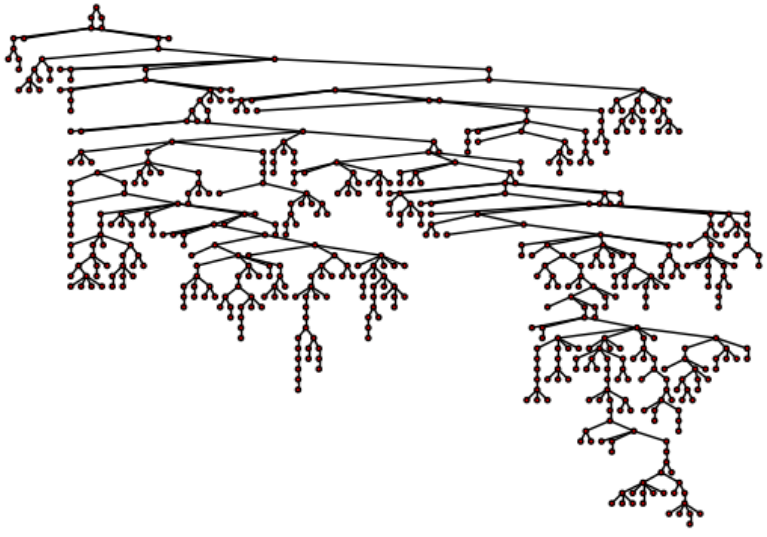
Plan de la présentation



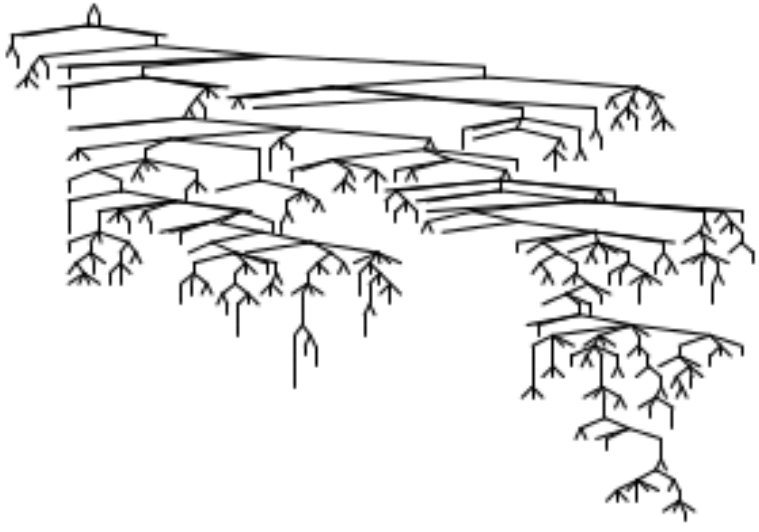
GraphViz



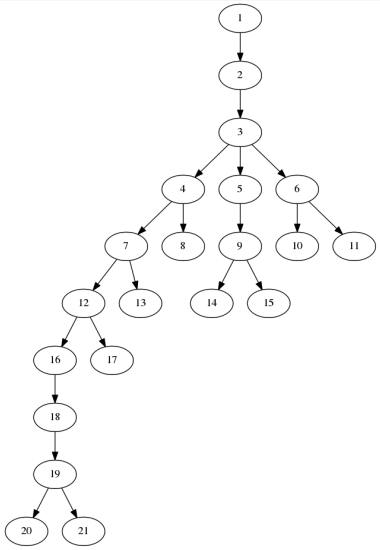
TikZ



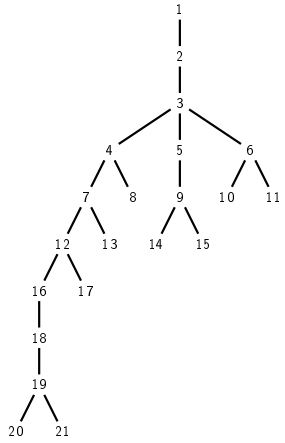
NetworkX



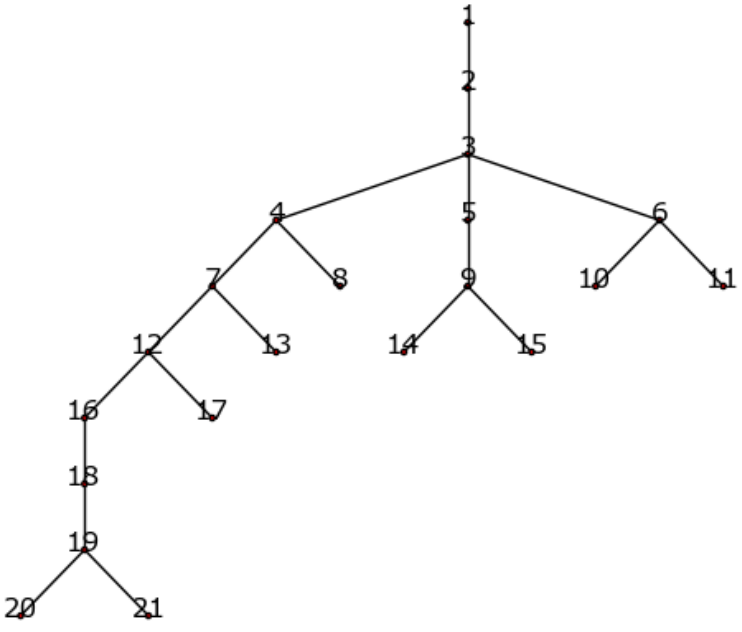
Asymptote



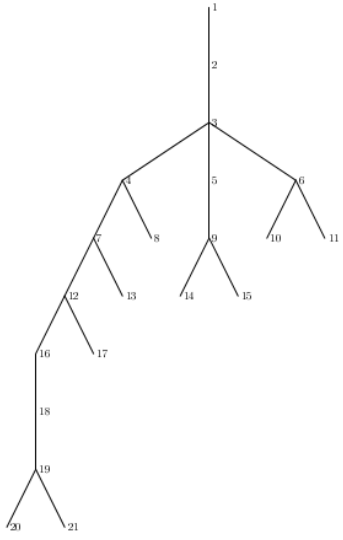
GraphViz



TikZ



NetworkX



Asymptote

Plan de la présentation

Bilan

- Étude d'articles scientifiques
- Complexité linéaire
- Modules réutilisables

Pour la suite

- Extension aux graphes
- Optimisation mémoire
- Prise en charge du format ARB
- Ajout de critères de représentation