ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

THÈSE PRÉSENTÉE À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DU
DOCTORAT EN GÉNIE
Ph.D.

PAR
LUIS EDUARDO DA COSTA

RECONSTRUCTION DE MODÈLES 3D À PARTIR D'INFORMATION 2D PARTIELLE: APPLICATION AU CAS D'UNE PLANTE

MONTRÉAL, LE 9 FÉVRIER 2007

CETTE THÈSE A ÉTÉ ÉVALUÉE

PAR UN JURY COMPOSÉ DE :

M. Jacques-André Landry, directeur de thèse
Département de génie de la production automatisée à l'École de technologie supérieure

Mme Natalia Nuño, présidente du jury
Département de génie de la production automatisée à l'École de technologie supérieure

Mme Lael Parrott, examinatrice externe
Département de Géographie à l'Université de Montréal

M. Richard Lepage, examinateur
Département de génie de la production automatisée à l'École de technologie supérieure

ELLE A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 14 DÉCEMBRE 2006

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

RECONSTRUCTION DE MODÈLES 3D À PARTIR D'INFORMATION 2D PARTIELLE: APPLICATION AU CAS D'UNE PLANTE

Luis Eduardo Da Costa

SOMMAIRE

L'analyse de champs cultivés en utilisant des méthodes de télédétection rapprochée a démontré être la meilleure méthode pour détecter des désordres physiologiques dans les plantes de ces champs. Pour faire ce type d'analyse, il est important de pouvoir manipuler virtuellement ces plantes à l'aide de modèles informatiques les représentant fidèlement ; dans cette thèse, je propose une méthode (allant de la définition d'un formalisme jusqu'au design et test d'un algorithme) pour générer ces modèles à partir de photographies 2D prises du champ.

Le formalisme choisi comme base pour la représentation de plantes s'appelle Systèmes de Lindenmayer (L-Systems) ; les L-Systems sont des systèmes grammaticaux contrôlés par une condition initiale et une ou plusieurs règles de réécriture, et l'itération répétée d'un L-System produit souvent un comportement émergent intéressant. Cependant, il est difficile de découvrir les règles qui produisent un comportement souhaité dans ce formalisme ; ce problème est appelé le « *problème inverse pour les systèmes de Lindenmayer* ». Générer un modèle informatique d'une plante est équivalent à résoudre le problème inverse pour un sous-type de ce formalisme, appelé « L-Systems à crochets » ; ce travail démontre la possibilité de résoudre le problème inverse pour des systèmes de Lindenmayer à crochets en utilisant un algorithme évolutif.

Une description détaillée de l'algorithme, ainsi que la justification du design choisi, sont présentées ; un ensemble d'expériences, choisies pour faire le test de l'adéquation de la méthode, démontre que l'algorithme explore de manière satisfaisante l'espace de solutions candidates, et que les approximations qu'il propose sont adéquates dans la majorité des cas. Ses limitations et faiblesses sont aussi rapportées et sont ensuite discutées ; une réflexion sur la recherche future complète ce document.

3D-MODEL SYNTHESIS FROM 2D PARTIAL INFORMATION : A NATURAL PLANT APPLICATION

Luis Eduardo Da Costa

ABSTRACT

The analysis of cultivated fields using near remote sensing has been demonstrated as the best method for detecting physiological disorders of the plants in the field. To perform this type of analysis it is important to be able to manipulate the plants virtually using computer models faithfully representing them ; in this thesis, a method is proposed (from the definition of a formalism to the design and test of an algorithm) for generating the models from field 2D photographes.

The formalism chosen as the base for plants representation is called Lindenmayer **S**ystems (L-Systems) ; L-Systems are grammatical systems controlled by an initial condition and one or more rewriting rules, and the repetitive iteration of an L-System often produces interesting emergent behavior. However, it is difficult to discover the rules that produce a specific desired behavior in this formalism ; this problem is called the "inverse" problem for Lindenmayer systems. Generating a computer model of a plant is equivalent to solving the inverse problem for a special subtype of this formalism, called "bracketed Lindenmayer systems" ; this paper demonstrates the possibility of solving the inverse problem for bracketed Lindenmayer systems by means of an evolutionary algorithm.

A detailed description of the algorithm, along with the justification of the chosen design, are presented ; a set of experiments, intended to test the correctness of the method, show that the algorithm explores in a satisfactory manner the space of candidate solutions, and that the approximations it proposes are adequate in most cases. Its limitations and weaknesses are also reported ; we discuss them and outline our future work.

REMERCIEMENTS

Tellement de personnes ont contribué à l'aboutissement de cette thèse, qu'il est devenu une tâche impossible de ne pas oublier quelqu'un. Sachez alors que mon oubli n'est qu'étourdissement, et que je vous remercie tous profondément.

Ceci dit, je voudrais exprimer ma gratitude au Prof. Jacques-André Landry pour être à l'origine de ce projet et bien sûr pour en avoir assumé la direction. Ses conseils ont fait de moi un meilleur chercheur, son amitié une meilleure personne ; merci pour la patience et pour le support inconditionnel. Je voudrais également remercier Natalia Nuño, pour l'honneur qu'elle me fait en présidant ce jury de thèse, et Lael Parrott et Richard Lepage pour avoir accepté d'évaluer ce travail.

Comment ne pas remercier tous les membres passés et présents du **LIVIA**, où j'ai trouvé un environnement humain idéal pour la recherche et bien plus qu'un lieu de travail. Jonh, Fred, Nedjem, Paulo, Luiz, Alexandro, Ali, Marisa, Guillaume, Clément, Albert, Eulanda, Vincent, Marcelo, sachez que vos prouesses au foot, au basket, à l'escalade, et les discussions à la table du café m'accompagneront longtemps.

Je ne serai pas ici sans l'appui inconditionnel de ma famille, répartie comme elle l'est partout dans le monde. Vous êtes mon support et le pilier principal de ma vie. Je voudrais embrasser du fond du cœur Erika et Nicholas pour toujours me récompenser avec vos sourires et votre amour. Je vous dédie ce travail, qui ne se serait jamais terminé sans vous.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS ET SIGLES

ÉTS	**É**cole de **T**echnologie **S**upérieure
LIVIA	**L**aboratoire d'**I**magerie, **V**ision, et **I**ntelligence **A**rtificielle de l'ÉTS
ACP	**A**nalyse en **C**omposantes **P**rincipales
L-System	**L**indenmayer **System**. Formalisme utilisé pour représenter des processus de développement
IFS	**I**terated **F**unction **S**ystem. L'équivalent français utilisé dans ce texte est *Système de fonctions itérées*
UV	Abréviation de *rayonnement ultraviolet*. Les rayonnements ultraviolets occupent la région du spectre électromagnétique située entre le violet du spectre visible et les rayons X. En électronique, les rayons ultraviolets sont utilisés principalement pour la photogravure et pour l'effacement de certaines mémoires à semi-conducteur. Dans le cas de la photogravure, le rayonnement UV a pour effet de polymériser les résines photosensibles. [Office de la Langue Française, 1994].
$\forall, \exists, \nexists, \nexists$	Symboles de base de la logique de premier ordre : $\forall$ signifie *pour tout* (« $\forall x \in \mathbb{R} f(x)$ » se lit « *tout* x *en* $\mathbb{R}$ *remplit* $f(x)$ ») ; $\exists$ signifie *il existe* (« $\exists x \in \mathbb{R} f(x)$ » se lit « *il existe* x *en* $\mathbb{R}$ *qui remplit* $f(x)$ »). $\nexists$ et $\nexists$ sont la négation de ces opérateurs.
$\mathbb{R}, \mathbb{C}, \mathbb{N}$	Symboles de base des divers ensembles de nombres utilisés en mathématique : $\mathbb{R}$ est l'ensemble des réels, $\mathbb{N}$ est l'ensemble des naturels, $\mathbb{C}$ est l'ensemble des complexes.

INTRODUCTION

La motivation initiale de ce projet vient du domaine de la science et de l'ingénierie appliquées à l'agriculture. Aujourd'hui, les exploitations agricoles (comme l'agriculture, l'acériculture, ou la culture de plantes ornementales) demandent une productivité accrue et hautement efficace. Baisser les coûts, sauver du temps et assurer que l'entreprise agricole, en général, soit plus efficiente, sont des points essentiels pour pouvoir être compétitifs sur les marchés locaux et internationaux.

Dans ce sens-là, l'application des technologies de l'information en agriculture joue un rôle primordial ; le positionnement satellital, les technologies sans-fil et les technologies de l'information sont en train de changer la relation du travailleur avec sa ferme. Les fermiers, ayant des mesures détaillées de leur champ, peuvent mieux gérer tous les aspects de l'opération agricole pour améliorer la productivité globale -de la distribution du sol à la plantation à la récolte- et donc de cette façon leur travail est plus spécifique et plus orienté à leurs besoins et à ceux de la ferme.

Les mesures des plantes d'un champ ont été traditionnellement obtenues manuellement, et de nouvelles techniques sont en émergence, telle la *télédétection*[1] : des caméras (hyper- ou multi-spectrales) sont portées sur des avions, de façon telle à prendre une large portion du champ avec chaque cycle de vol. Cette technique a démontré être bien adaptée à certaines tâches : par exemple, à l'estimation de la quantité de biomasse dans un champ, le contenu des plantes en eau et en chlorophylle (la chlorophylle étant corrélée au contenu en azote dans les feuilles).

Cependant, la télédétection ne peut donner un diagnostic précis de l'état physiologique d'une plante. Ceci provient du fait que la réflectance[2] des végétaux est peu spécifique :

[1] Télédétection : ensemble des connaissances et techniques utilisées pour déterminer des caractéristiques physiques et biologiques d'objets par mesures effectuées à distance, sans contact matériel avec ceux-ci (Office québécois de la langue française, http://www.granddictionnaire.com/)

[2] Réflectance (substantif féminin) : Rapport de l'intensité de l'onde incidente à celle de l'onde réfléchie. (Office québécois de la langue française, http://www.granddictionnaire.com/)

des variations similaires de réflectivité peuvent être observées pour différents stress environnementaux (Penuelas et Fillela, 1998). Par exemple, des changements de réflectance causés par la dégradation de la chlorophylle (ou par l'inhibition de sa synthèse) peuvent être attribués à des carences en azote, en fer, en soufre, en manganèse, ou même en magnésium (Penuelas et Fillela, 1998). Additionnellement,

- ces changements sont, dans la plupart des cas, une réponse tardive au stress présent (Penuelas et Fillela, 1998) et
- l'âge de la feuille observée ainsi que le stade de développement du plant peuvent fortement influencer les effets des carences minérales (Cerovic et al., 1999) : par exemple, des effets marqués d'une carence en magnésium chez de vieilles feuilles de maïs peuvent être observés, mais non chez les jeunes feuilles du même plant (Heisel et al., 1996).

En même temps, d'autres recherches ont indiqué la possibilité d'identifier les carences et désordres physiologiques en utilisant la fluorescence induite par UV et l'analyse des émissions thermiques *directement* d'une feuille *spécifique* dans l'arborescence de la plante (Cerovic et al., 1999). Il semble donc que, malgré les possibilités de suivi des cultures à grande échelle par télédétection, obtenir de l'information **précise** sur une plantation passe par une étude *rapprochée* de chacune des plantes.

Une contrainte fondamentale de ce type d'étude est l'impossibilité physique de transporter tout l'équipement spécialisé au champ pour obtenir des valeurs d'étude précises ; les mesures à prendre sont si diverses (manque/surplus de minéraux, d'azote, de fertilisant(s), d'herbicide(s), ...) et les façons de les obtenir si spécialisées (caméras hyper- et multispectrales, machines centrifuges, mélanges réactifs spéciaux, ...) qu'il devient très compliqué de faire des mesures complètes sur le champ. La plante pourrait aussi être étudiée dans un laboratoire spécialisé, mais cette façon de travailler est plutôt lente et toujours très dispendieuse.

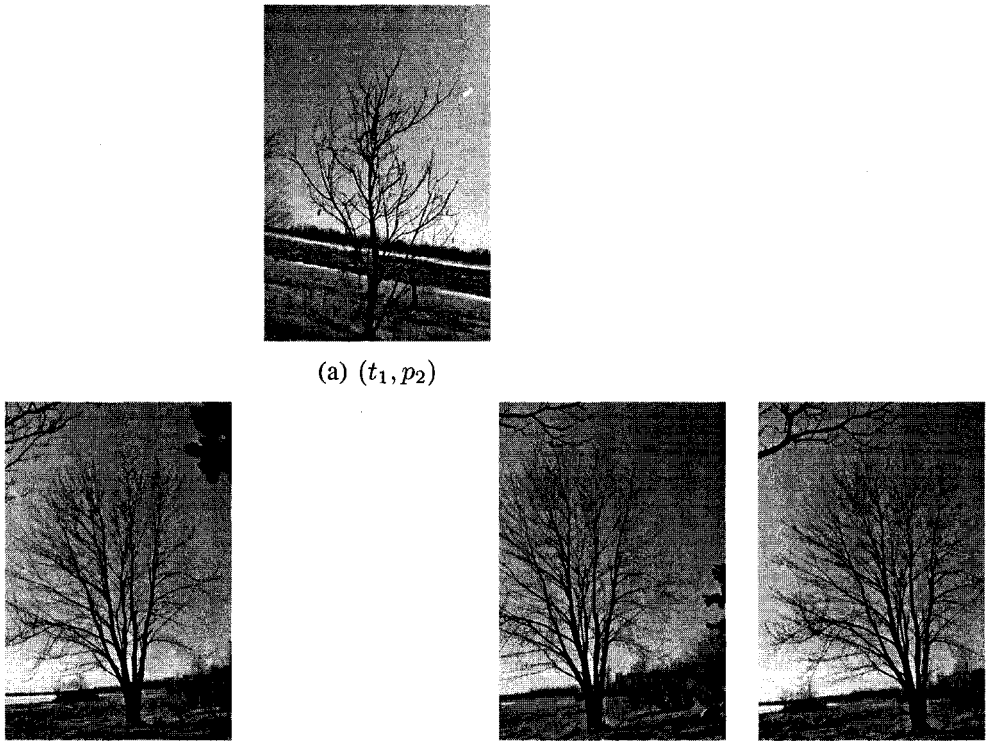
Combiner les deux approches antérieures (étudier les cultures par télédétection, mais en regardant le champ de très près) s'appelle *télédétection rapprochée*. Cette idée se base sur l'utilisation de capteurs très spécifiques embarqués dans des dispositifs plus rapprochés de la plantation (par exemple, un tracteur) ; ces dispositifs seraient alors reliés à des machines spécialisées (comme, par exemple, à des applicateurs d'engrais à taux variables (Raun et al., 2002)). Cependant, cette technique n'a pas connu beaucoup de succès a cause des restrictions *géométriques* présentes : trouver la (les) feuille(s) spécifique(s) à observer est un problème complexe en soi, nécessitant la manipulation de la plante par un opérateur humain.

Donc, en résumé, l'étude détaillée d'une plante est une source précieuse d'information sur sa santé, sur les traitements antérieurs qui ont été appliqués sur le champ et, par conséquent, comme base pour planifier les traitements à appliquer ; mais l'obtention de données intéressantes ne peut se faire automatiquement, à cause de certaines restrictions géométriques du problème. L'approche que nous proposons ici consiste à construire un modèle détaillé fidèle de la plante, de façon telle à pouvoir le manipuler et ainsi obtenir de l'information de la plante réelle ; notre idée est de bien caractériser la structure des plantes par des reconstructions numériques 3D des images obtenues en 2D pour éventuellement reconnaître et échantillonner automatiquement une ou plusieurs feuilles d'intérêt pour des fins d'analyse. Les analyses numériques 3D permettraient d'identifier la plante elle-même, sa géométrie, son stade de croissance, rendant possible des mesures détaillées (comme une évaluation réaliste de l'indice de surface foliaire ou une comparaison de la taille et de la forme du plant observé à d'autres plants).

Il est clair que c'est d'une importance première que les modèles générés se rapportent fidèlement aux plants réels *tout au long de leur développement* (et non pas à un certain et unique instant de temps) : en d'autres mots, ce modèle devrait avoir la puissance de représenter les changements temporels qui se succèdent dans la plante particulière sous

étude. Dans ce travail, une représentation d'une plante tout au long de plusieurs instants de temps (et aussi vu de plusieurs points de vue) est appelé un *processus de développement.*

La Fig. 1 présente un tel processus de développement.



(a) (t_1, p_2)

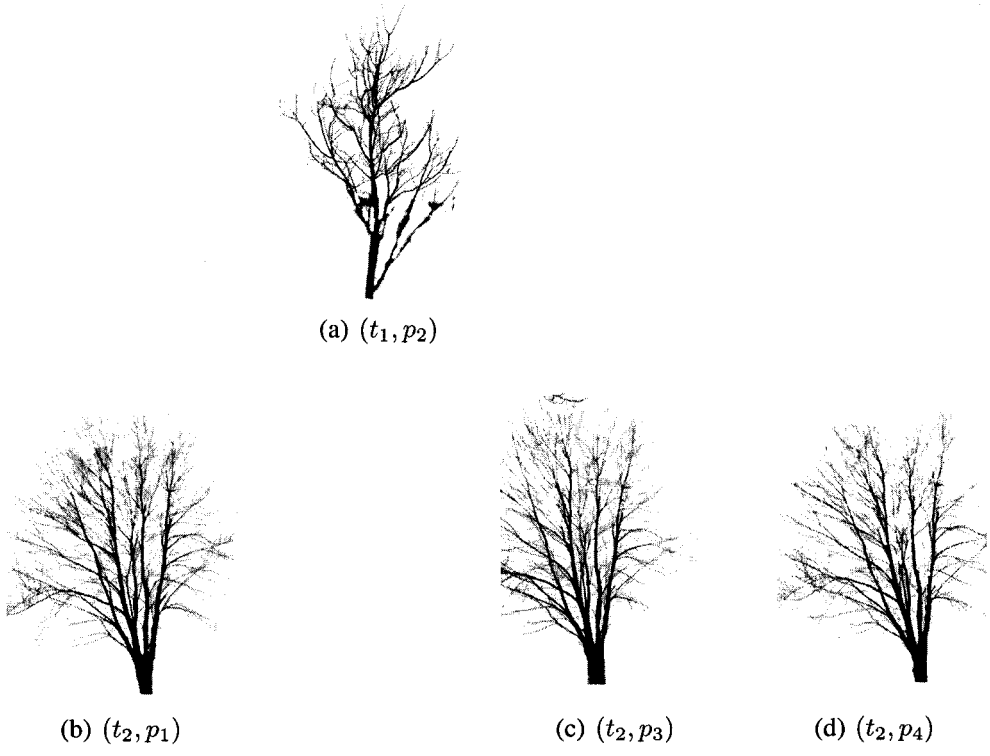
(b) (t_2, p_1) (c) (t_2, p_3) (d) (t_2, p_4)

Figure 1 Un processus de développement

Un arbre est montré à plusieurs instants de temps, vu de plusieurs points de vue différents. La première ligne de cette figure représente l'instant de temps t_1 (défini arbitrairement : t_1 peut être, par exemple, « *printemps 2005* ») et la deuxième ligne représente l'instant de temps t_2 (défini arbitrairement aussi, mais de façon consistante avec t_1 : t_2 peut être, par exemple, « *printemps 2006* »). En colonnes, des « *points de vue* » sont séparés : p_1, p_2, p_3, p_4. La définition de ces points de vue est expliquée plus tard dans ce document.

L'objectif de cette thèse est d'explorer la possibilité de synthétiser un modèle fidèle pour un arbre (ou plante) à partir d'un ensemble de photos représentant un processus de développement (comme celui montré en Fig. 1).

Le premier pas vers cet objectif serait de générer le *squelette détaillé* de l'arbre : l'ensemble complet de branches qui constituent la structure interne de support qui donne à l'arbre sa forme. Il est alors impératif de faire un pré-traitement des photos pour obtenir les squelettes présents dans chacun d'eux. Ce pré-traitement (un type spécial de segmentation de l'arbre) produit des séquences de photos comme celles montrées à la Fig. 2. Dans



(a) (t_1, p_2)

(b) (t_2, p_1) (c) (t_2, p_3) (d) (t_2, p_4)

Figure 2 Les squelettes des arbres

Ces squelettes ont été obtenus des photographies de la Fig. 1

cette thèse, la segmentation de l'arbre pour obtenir des squelettes n'est pas décrit ; une méthode à ce sujet est en développement au LIVIA.

Le reste du document est organisé de la façon suivante : au chap. 1 je présente les bases théoriques utilisées dans le développement de cette thèse, qui se résument à (1) justifier la modélisation des plantes comme instances d'objets fractals, et, (2) présenter deux for-

malismes (appelés « L-Systems » et « IFS ») pour guider cette modélisation. Une fois ces bases établies, je placerai la problématique de cette thèse par rapport à la communauté de recherche (chap. 2) ; ce travail m'a permis de découvrir que faire la construction d'un modèle pour une plante en partant d'un ensemble de « photos » (2D) revient à résoudre le problème inverse pour un formalisme appelé « L-Systems avec crochets » (présenté formellement en 1.3.2). J'ai alors fait l'étude des particularités du problème, pour ainsi aboutir au design et implantation d'une méthode pour le résoudre (chap. 3) ; un ensemble d'expériences sont alors proposées pour tester l'idée de la solution : les résultats sont dans le chap. 4. Comme conclusion, je discute ces résultats et la facon dont ils se réfèrent au problème original, ainsi que le travail par lequel cette thèse peut être augmentée.

Finalement, des annexes sont jointes à ce document : l'annexe II décrit l'environnement de travail implanté ; l'annexe III décrit la proposition d'une mesure pour comparer des arbres naturels. Et l'annexe IV fait une expansion mathématique pour compter combien de solutions possibles la méthode explore.

CHAPITRE 1

CADRE THÉORIQUE

Dans ce chapitre, les bases théoriques des outils référés dans cette thèse (et utilisés tout au long du travail) sont présentées ; ponctuellement, je parlerai de l'idée de *figures fractales* et comment elles se rapportent aux figures des arbres et plantes naturelles (section 1.1). Cette idée nous mènera à la présentation de quelques modèles pour l'architecture des plantes (section 1.2), pour finir avec deux formalismes pour la représentation des plantes : les L-Systems (section 1.3) et les systèmes itérés de fonctions (section 1.4).

1.1 Les objets fractals et les plantes

1.1.1 Qu'est-ce qu'un objet fractal ?

Le sens intuitif pour le concept de **fractal** est celui d'une figure géométrique (ou d'un objet naturel) qui combine les caractéristiques suivantes (Mandelbrot, 1982) :

a. Ses parties, à une échelle différente et (possiblement) légèrement déformées, ont la même forme ou structure que le tout

b. Sa forme est soit extrêmement irrégulière, soit extrêmement interrompue ou fragmentée

c. Il contient des « éléments distinctifs » dont les échelles sont très variées

Benoît Mandelbrot, avec son oeuvre, *The Fractal Geometry of Nature*, est souvent identifié comme le père de la géométrie fractale (Peitgen et al., 1992). Dans son livre, il étudie des objets naturels très divers, dont beaucoup sont fort familiers (tels la Terre, le Ciel et l'Océan) à l'aide d'une large famille d'objets géométriques jusqu'à ce moment jugés ésotériques et inutilisables (Mandelbrot, 1989) ; pour les étudier il a mis au point et utilisé une nouvelle géométrie de la nature. Il a développé un langage où des idées considérées

« étranges » en mathématiques ont été mises ensemble et décrites avec les mêmes mots, phrases, et grammaire (Peitgen et al., 1992).

En s'inspirant du latin *fractus* (adjectif latin signifiant « irrégulier » ou « brisé ») il a appelé « fractale » ou « objet fractal » la notion qui lui a servi de fil conducteur ; dans son livre, il poursuit 3 objectifs :

a. décrire la *forme* de divers objets,

b. expliquer et mettre l'accent sur le *hasard* sur lequel les algorithmes de construction de fractales se basent, et

c. définir ou regrouper des façons de mesurer la *dimension* des objets fractals. La *dimension fractale* d'une forme mesure son degré d'irrégularité et de brisure (Mandelbrot, 1989) ; contrairement aux dimensions Euclidiennes, elle peut être une fraction simple, comme $1/2$ et même un nombre irrationnel, tel que $\log 4/\log 3 \sim 1.2618\ldots$, ou π. De (Mandelbrot, 1989, p. 6) :

> (...)Ainsi, il est utile de dire de certaines courbes planes très irrégulières que leur dimension fractale est entre 1 et 2, de dire de certaines surfaces très feuilletées et pleines de convolutions que leur dimension fractale est intermédiaire entre 2 et 3, et enfin de définir des poussières sur la ligne dont la dimension fractale est entre 0 et 1.

Il existe une très grande bibliographie sur le thème des fractales ; particulièrement, les deux livres de Mandelbrot et celui de Peitgen *et al.*, cités dans la bibliographie de ce document, sont des sources d'information précieuses (et très agréables à lire, de surcroît). De nombreux exemples d'objets mathématiques fractals y sont étudiés ; ici, je voudrais en présenter quelques uns, pour ainsi terminer d'établir l'idée fondamentale de ce concept.

a. **L'ensemble de Cantor (Cantor, 1883).** Cantor (1845-1918) était un mathématicien Allemand de l'Université de Halle, où il développa les idées fondamentales de ce qui est maintenant connu sous le nom de *théorie des ensembles*. L'ensemble de Cantor

est un exemple d'une classe exceptionnelle d'ensembles (appelés *parfaits nulle part denses -perfect, nowhere dense* en anglais- ou *totalement discontinus*) ; il est le plus important des « premiers » fractales (Peitgen et al., 1992).

Sa construction est comme suit : commençons avec l'intervalle $[0, 1]$; maintenant éliminons l'intervalle ouvert $(1/3, 2/3)$: ceci laisse 2 intervalles ($[0, 1/3]$ et $[2/3, 1]$) de longueur $1/3$ chaque. Maintenant, nous répétons, en commençant avec ces deux intervalles : pour chacun, nous éliminons le tiers du milieu, ce qui produit 4 intervalles de longueur $1/9$. Continuer comme cela, en générant 2^n intervalles de largeur $1/(3^n)$ après le pas n ; l'ensemble de Cantor est l'ensemble de points qui restent après avoir enlevé les « tiers » des segments indéfiniment (Peitgen et al., 1992, pages 68-76). Les premiers pas de sa construction sont montrés à la Fig. 3.

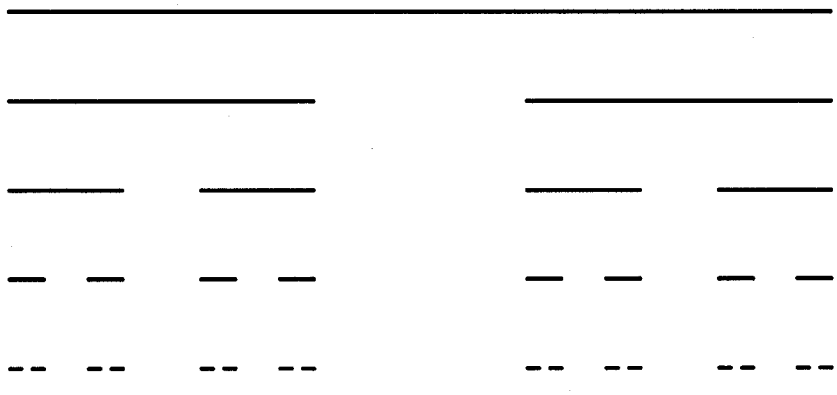


Figure 3 Construction de l'ensemble de Cantor

La dimension fractale de cet ensemble est $log2/log3 \sim 0,6309$: c'est *plus qu'un point* (dimension 0) mais *moins qu'une ligne* (dimension 1) (Peitgen et al., 1992, p. 206).

b. **Le triangle de Sierpinski (Sierpinski, 1915), (Sierpinski, 1916).** Cette figure fractale classique fût introduite en 1916 par W. Sierpinski. La construction de base de cette figure va comme suit : commencer avec un triangle dans le plan, et choisir les

mi-points de chacun de ses trois côtés. Avec les sommets du triangle original, ceci définit 4 triangles, desquels on élimine le triangle du milieu. Maintenant nous répétons la même procédure avec les trois triangles restants ; si nous continuons cette procédure jusqu'à l'infini, les points restants constituent le triangle de Sierpinski ; une approximation est montrée à la Fig. 4.

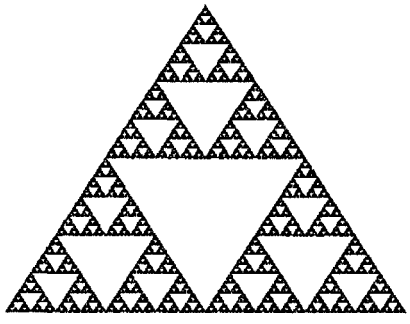


Figure 4 Une approximation au triangle de Sierpinski

Les premières itérations de sa construction sont montrées à la Fig. 5.

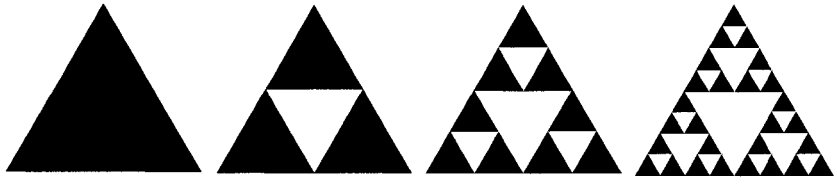


Figure 5 Construction du triangle de Sierpinski

Ce type de figure résiste à une analyse géométrique « habituelle » ; par exemple, calculons l'aire du triangle de Sierpinski[1]. Supposons que l'aire du triangle initial T_0 est 1 (une) unité. À la première itération, $1/4$ de l'aire de T_0 est enlevée pour produire T_1, qui a donc une aire de $3/4$. La deuxième itération enlève 3 triangles à T_1, chacun ayant une aire de $1/4$ de T_1 : $1/4 * 3/4 = 3/16$; donc T_2 a une aire égale

[1] Repris de http://ecademy.agnesscott.edu/~lriddle/ifs/siertri/area.htm

à $3/4 - 3/16 = 9/16 = (3/4)^2$ Il est assez simple de démontrer par induction que l'aire de T_n est $(3/4)^n$. Donc, à l'infini, l'aire du triangle de Sierpinski est 0... et, en même temps, le nombre de points sur le triangle est infini : cet apparent paradoxe ne fait qu'illustrer le fait que l'aire n'est pas une dimension utile pour cet ensemble.

Sierpinski introduit aussi une autre figure au monde des figures fractales : le « tapis » de Sierpinski (*Sierpinski carpet*, en anglais) ; son approximation est montrée à la Fig. 6.

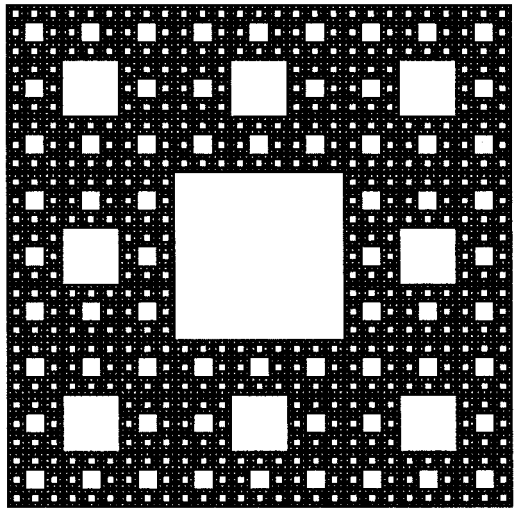


Figure 6 Une approximation au tapis de Sierpinski

La première impression que nous avons de cet objet est que c'est juste une variation du thème antérieur ; mais cette figure (dont les premiers pas de sa construction sont faciles à comprendre, et montrés à la Fig. 7) est en fait une généralisation de l'ensemble de Cantor (Peitgen et al., 1992), et« (...) constitue une espèce de *super-objet* qui contient tous les objets en 1D dans un sens topologique. » (Peitgen et al., 1992, p. 112) (pp. 112-121 pour une intéressante discussion à ce sujet).

Ceci veut dire que **n'importe quel objet en 1D se retrouve dans ce super-objet** (à une déformation topologique près). Par exemple, les trois variations de l'araignée

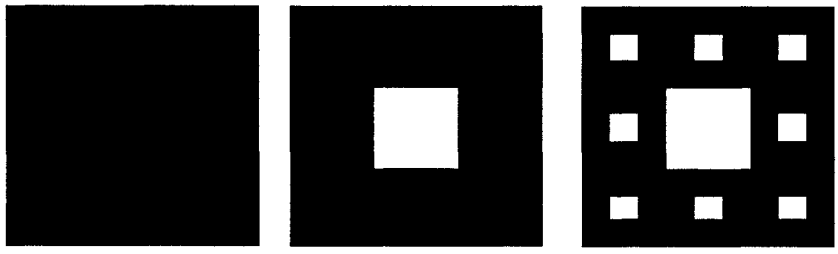


Figure 7 Construction du tapis de Sierpinski

à cinq bras de la Fig. 8 sont topologiquement équivalentes ; une des déformations existantes va se retrouver dans le tapis de Sierpinski (Peitgen et al., 1992, p. 112).

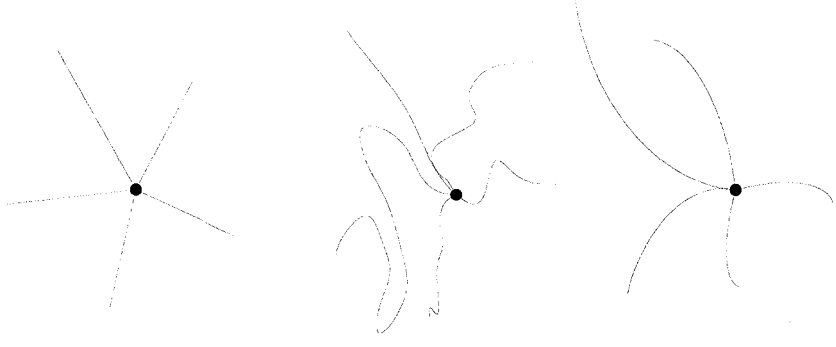


Figure 8 Quelques équivalences topologiques d'une araignée à 5 bras

La dimension fractale du triangle se calcule comme $\log 3/\log 2 \sim 1.5850$; pour le tapis, cette dimension est $\log 8/\log 2 \sim 1.8928$: ce sont *plus que des lignes* (dimension 1) mais *moins qu'un plan* (dimension 2) (Peitgen et al., 1992, p. 206).

c. **La courbe de Koch (von Koch, 1905).** Helge von Koch, mathématicien Suédois, introduit cette courbe, aussi appelée le « *flocon de neige* » de Koch (Peitgen et al., 1992) ; elle est approximativement représentée à la Fig. 9. La méthode de construction pour la figure de von Koch est la suivante (reprise de (Prusinkiewicz et Lindenmayer, 1990, pp. 1 et 2 du chapitre 1)) : prendre une figure de base (l'*initiateur*,

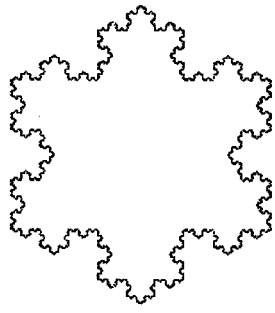


Figure 9 « Flocon de neige » de von Koch

Fig. 10(a)), et remplacer chaque segment de ligne par la figure de récursion (le *générateur*, Fig. 10(b)).

(a) Initiateur (b) Générateur

Figure 10 Règles de construction du « flocon de neige »

Continuer le remplacement avec la figure générée. Les 3 pas suivants sont montrés dans la Fig. 11.

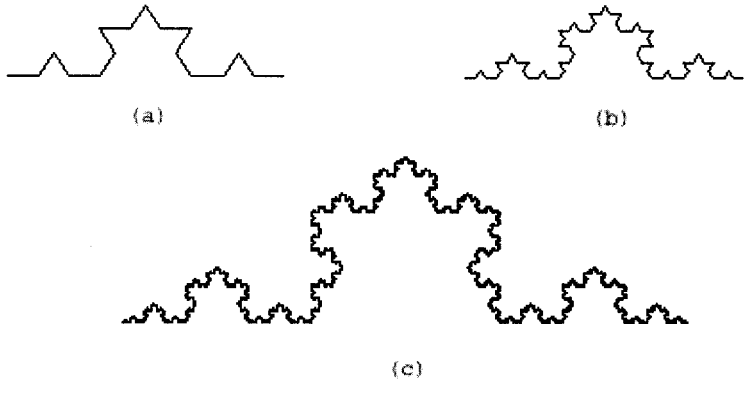


Figure 11 Dérivations du « flocon de neige »

La courbe de Koch est aussi complexe que des contours trouvés dans la nature, comme celui des côtes (Peitgen et al., 1992) : plis dans des plis dans des plis,...

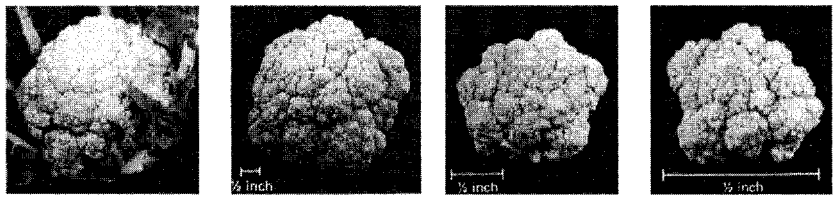
L'intention de Koch en construisant cette courbe était de donner un exemple d'une courbe *qui ne peut être différentiée*, c'est à dire qui n'admet de tangente dans aucun de ses points (Peitgen et al., 1992). La différentiation d'une courbe étant un concept de base dans la calcul (inventé de façon indépendante par Newton et Leibniz (Peitgen et al., 1992)), la découverte de courbes qui ne peuvent l'être précipita une crise mineure en mathématiques (avant Koch, Karl Weierstrass montra, en 1872, une de ces courbes). La courbe de Koch est, dans ce sens, une courbe qui a *des coins partout.*

Si l'idée d'avoir une figure avec « des coins partout » vous semble étrange, considérons pendant un instant quel est le périmètre et l'aire du « flocon de neige »[2] (Fig. 9) : la longueur de la frontière F_n à l'itération n de la construction est $3 * (4/3)^n * s$ (où s est la longueur de chaque côté de l'initiateur) : $F_n \rightarrow \infty$ quand $n \rightarrow \infty$, donc le flocon de neige a un *périmètre infini.* Mais, en plus, l'aire de la figure est clairement non-infini, et est calculé comme étant $2 * \sqrt{3} * s^2/5$. Donc *le flocon de neige de Koch est une figure à aire finie entouré d'une frontière de longueur infinie*

1.1.2 Idée d'auto-similarité

Le concept d'*auto-similarité* est une des colonnes vertébrales du concept de fractale (Mandelbrot, 1982) ; ce concept nous permet de représenter des objets complexes dans sa structure en les divisant en composantes plus petites. Mathématiquement, spécifier l'auto-similarité d'un objet $\mathcal{O}$ nécessite la spécification d'une famille de transformations qui, appliquées à $\mathcal{O}$, produisent ses parties isolées ; de façon intuitive, ce terme définit un objet dont toutes les parties sont similaires les unes aux autres, et à l'objet comme un tout. Ce concept peut être illustré à l'aide d'une figure naturelle : un chou-fleur (Peitgen et al., 1992). Si nous regardons les images obtenues de pièces chaque fois plus petites d'un chou-fleur (Fig. 12) nous pouvons dire qu'elles ont toutes *essentiellement* la même forme.

[2] Détails en http://ecademy.agnesscott.edu/~lriddle/ifs/ksnow/area.htm



(a) Chou-fleur (b) Morceau de chou-fleur (c) Morceau petit (d) Morceau très petit

Figure 12 Auto-similarité dans un chou-fleur

Les morceaux à droite se retrouvent dans les morceaux à gauche (Peitgen et al., 1992, p. 64).

Cette intuition a sans doute besoin d'être travaillée pour être utilisée comme support dans la définition d'une fractale, particulièrement sur 2 points :

a. Les objets fractals sont définis *à l'infini*[3], tandis que l'auto-similarité observée dans la Fig. 12 est seulement vraie pour un nombre fini d'itérations.

b. L'auto-similarité du chou-fleur (Fig. 12) n'est pas *stricte* : chaque pièce *ressemble beaucoup* aux autres, mais n'est pas exactement *égale*, mathématiquement parlant.

Ici je ne veux pas mener une discussion sur le concept mathématique d'auto-similarité, mais plutôt montrer comment l'idée est utile pour présenter de quelle façon les plantes peuvent être considérées comme des **représentations naturelles d'une fractale.**

Regardons la structure des tiges dans un arbre (ce qui est appelé les *axes* d'un arbre, Fig. 13). À chaque tige (axe) est associé un certain *ordre* : le tronc est un *axe de niveau 1*, les branches qui sortent du tronc sont des *axes de niveau 2*, et ainsi de suite. Ce qui est intéressant, c'est que les structures d'ordre i peuvent être approximées par les structures ayant un ordre inférieur à i, et que faire un *zoom* sur ces structures nous donnera des détails sur les branches au niveau $i + 1$. Les détails à ces 3 niveaux (j, avec $j < i$, i, et $i + 1$) sont similaires les uns aux autres : l'arbre est alors appelé un objet *auto-similaire* (Ferraro et al., 2004; Herman et al., 1975) ; ce type de branchement est appelé un *paracladium*[4].

[3] Une analogie entre les concepts d'auto-similarité et de limite d'une fonction mathématique est présentée de façon particulièrement claire dans (Peitgen et al., 1992)

[4] Un paracladium est une branche qui a le même type de structures que celles trouvées sur la partie terminale de sa branche-mère (Lindenmayer, 1977).

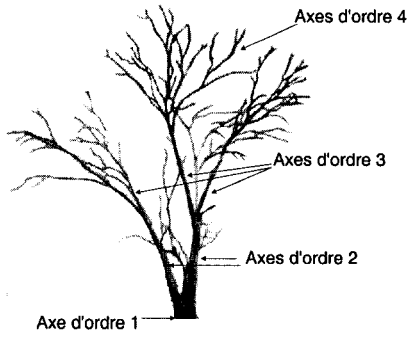


Figure 13 Axes d'un arbre

L'ensemble des tiges et branches, sans les feuilles, est le *squelette* d'un arbre.

Bien sûr, les systèmes complexes de branches dans un arbre ne sont pas *strictement* des objets fractals, n'ayant pas une *longueur infinie*. Tout ce qu'on peut dire, à la limite, c'est que les plantes naturelles ont un aspect fractal dans un rang limité d'échelles (Godin, 2000) ; dans les processus de développement d'un bon nombre d'organismes vivants il est possible d'observer des structures multicellulaires répétées (Ferraro et al., 2004). Ceci serait particulièrement vrai dans le cas des plantes : « *dans le cas le plus simple, la même structure est répétée le long d'un axe, comme (par exemple) des feuilles au long d'une branche* » (Ferraro et al., 2004). Dans des cas plus complexes, la structure entière d'un certain stage est répétée en faisant partie d'organismes à un instant ultérieur, résultant en des organismes ou organes composés (Herman et al., 1975).

Les chercheurs en botanique ont spontanément fait le lien entre l'idée d'auto-similarité et des concepts utilisés en botanique ; ce lien est évident dans plusieurs approches à la description et modélisation de plantes (Ferraro et al., 2004). Par exemple, Troll décrit une inflorescence comme *un système consistant de la florescence première et de paracladia* (Frijters et Lindenmayer, 1976) ; les *relations paracladiales dans les branchements* sont définies : une structure de branches a une relation paracladiale si « le comportement d'une branche est exactement le même que le comportement de la partie de la branche-mère qui s'est développée depuis la formation de la branche-fille » (Frijters et Lindenmayer, 1976).

Ces idées permettent de réduire la compléxité des représentations 3D de plantes (structures en arbres et inflorescences composées) (Prusinkiewicz et al., 2001).

Comme nous le voyons, l'idée implicite d'auto-similarité permet de définir de façon formelle les structures d'arbres ; mais l'auto-similarité dans une plante est aussi liée au point de développement que cette plante a atteint : il a par exemple été montré que la croissance et les processus de branchement sont similaires dans toutes les parties et à tous les stages de développement dans l'Orme Japonais (*Zelkova serrata*) (de Reffye et al., 1990). Ainsi, il est possible d'identifier un « axe théorique » qui inclut tous les états de différenciation morphologique dans la plante à travers le temps, définissant ainsi le potentiel de développement auto-similaire de la plante. Ceci est lié à l'idée que tous les méristèmes[5] d'une plante subissent la même série de stages de développement (appelés *âges physiologiques*), et deux méristèmes avec le même âge physiologique produisent essentiellement des structures similaires (Arber, 1950; Barthélémy, 1991; Gatsuk et al., 1980; Honda, 1971).

1.2 Modèles d'architecture de plantes

Les représentations d'architecture de plantes sont communément utilisées pour modéliser structure et fonction : par exemple, transfert d'eau dans la plante, croissance des racines, analyse de l'architecture, interaction avec le micro-environnement, mécanique du bois, écologie et développement ou modèles visuels (Godin, 2000). Ces représentations sont aussi appelées des *modèles* ; ils sont spécifiés en différents langages et avec différents objectifs, pour différentes applications. Ceci explique l'existence d'une grande variété de représentations, en utilisant différents formalismes et ayant différentes propriétés (Godin, 2000).

La notion d'architecture de plante est utilisée communément dans la littérature spécialisée, mais il n'y a pas de définition *universellement* acceptée (Godin, 2000) : une architecture

[5] Méristème : Tissu capable d'actives divisions cellulaires qui ajoutent de nouvelles cellules au corps de la plante.Massif ou palissade de cellules jeunes, indifférenciées, se multipliant activement [Office de la langue française, 1990].

peut être vue comme « *un ensemble de caractéristiques décrivant la forme, la taille, la géométrie et la structure interne d'une plante* » (Ross, 1981). « *Architecture de plante* » est aussi synonyme de *modèle architectural* d'une espèce de plante : c'est « *la description des formes de croissance d'un individu idéal d'une espèce* » (Hallé et al., 1978). Basés sur cette idée, les auteurs proposent un système de description (appelé **HO**, pour **H**allé et **O**ldeman) qui a premièrement été conçu pour être appliqué aux arbres tropicaux (quoique d'autres chercheurs ont postérieurement argumenté qu'il est adéquat pour décrire *toutes* les espèces d'arbres (Robinson, 1996)). En décrivant la structure des arbres, le système procède par la classification des troncs et des branches, lesquels font la restriction de la forme et de la taille d'un arbre. L'architecture résultante est décrite par 23 *modèles idéaux*, repris postérieurement pour proposer une nouvelle notation symbolique pour architectures d'arbres, ainsi que son équivalence et différences avec le modèle **HO** (Robinson, 1996). D'un autre côté, une architecture de plante peut aussi être assimilée à *une description individuelle basée sur la décomposition de la plante en composantes, en spécifiant leur type biologique et/ou forme, et/ou leur localisation/orientation dans l'espace et/ou la façon dont ces composantes sont physiquement liées les unes aux autres* (Godin, 2000) ; avec cette définition, les architectures de plantes sont divisées en 3 niveaux de complexité :

a. Représentation *globale*, où les plantes sont considérées comme un tout ; les organes de type similaires sont aussi considérés comme un tout ayant une fonction globale (par exemple, photosynthèse).

b. Représentation *modulaire*, où une plante est considérée comme une répétition de certain types de composantes (spatiales ou organiques)

c. Représentation *multi-échelle*, où le niveau de description peut être choisi selon les besoins du modélisateur.

Le choix d'un formalisme pour ce projet de thèse a été grandement influencé par le besoin de modéliser le *développement* d'une plante : ses changement à travers sa croissance. Il est alors très important que la représentation choisie supporte l'idée qu'une plante change

avec le temps, et que différents niveaux de précision sont (probablement) requis à chaque changement. En d'autres mots, *une architecture pour une plante est une description des formes de croissance d'un individu idéalisé d'une espèce*[6] (Hallé et al., 1978) et qu'elle se réfère à *un ensemble de règles qui expriment la structure et la croissance d'individus dans un groupe donné en moyenne et en conditions non-limitatives* (Hallé et al., 1978).

J'ai identifié deux formalismes qui suivent ces idées et qui, par conséquent, pourraient m'être utiles pour ce travail : les Systèmes de Lindenmayer (L-Systems : Lindenmayer System, en anglais), présentés en 1.3 et les systèmes itérés de fonctions (IFS, Iterated Function System, en anglais), présentés en 1.4.

1.3 L-Systems

Les L-Systems ont été introduits en 1968 par Aristide Lindenmayer comme cadre théorique pour étudier le développement des organismes multicellulaires (Lindenmayer, 1968). Après l'incorporation de caractéristiques géométriques, les modèles des plantes générés par les L-System ont abouti à des représentations détaillées de la réalité, et ont donc pu être utilisées par les algorithmes d'infographie pour visualiser des stages de développement naturels.

La recherche en L-Systems est un champ vraiment interdisciplinaire, et par conséquent les résultats obtenus sont applicables dans un vaste secteur de connaissances ; Prusinkiewicz écrit, « (...) *la visualisation des structures et des procédés qui les créent font que l'art et la science se rejoignent* » (traduction libre de (Prusinkiewicz et Lindenmayer, 1990, p. vi du préface)). Ce champ de recherche, fascinant à cause de la beauté mathématique des concepts utilisés, la simplicité de ses algorithmes, et des figures intrigantes qui sont obtenues, est actuellement profondément exploré par les chercheurs de bon nombre de laboratoires dans le monde ; ici, je vais présenter les idées de base, en essayant de guider le lecteur intéressé vers les ressources spécialisées pour les (nombreux) détails manquants.

[6] Pour des publications utilisant cette définition, consulter références dans (Godin, 2000, p. 414)

Les L-Systems naissent du concept primordial de la réécriture : l'idée de base est de définir des objets complexes en appliquant de façon répétée un nombre fini de règles de réécriture en partant d'un objet simple. Une figure « classique » générée par l'application récursive d'un ensemble de règles a été présentée par von Koch en 1905 ; la figure s'appelle « *flocon de neige* », et est présentée en Fig. 9 (voir page 13 et suivantes pour voir l'explication complète).

Les systèmes de réécriture les plus utilisés et les mieux compris opèrent sur des chaînes de caractères ; un grand intérêt pour ce type de système formel est apparu à la fin des années 1950 (voir les remarquables publications marquées comme (Chomsky, 1956) et (Naur et Backus, 1960) dans la bibliographie de ce document) ; dans ces travaux, des grammaires sont utilisées pour décrire des caractéristiques syntaxiques de langages naturels (Chomsky) et de programmation (Backus et Naur). Cette puissante machine théorique n'a pu être appliquée directement pour l'interprétation et la génération d'images car les langages correspondant à des classes d'images intuitivement simples, comme des lignes droites avec une inclinaison arbitraire, cercles de rayon arbitraire, et même des rectangles dans une grille carrée, étaient tous sensibles au contexte (Feder, 1968). En contraste, pratiquement tous les résultats utiles en théorie de langages formels se réfèrent à des langages libres de contexte (Feder, 1968).

Cependant, en 1968, Aristid Lindenmayer, un biologiste, a introduit l'idée des L-Systems (Lindenmayer, 1968). L'idée de Lindenmayer était d'achever une description formelle du développement d'un organisme multicellulaire simple ; la différence essentielle entre les grammaires de Chomsky et les L-Systems est que, dans ce dernier, les règles de production sont appliquées de façon parallèle et que, dans les grammaires « de Chomsky », les productions sont appliquées de façon séquentielle, une à la fois. Cette différence fait en sorte qu'il existe des langages qui peuvent être générés par des L-Systems « simples » (indépendants du contexte, appelés « 0L-Systems ») mais qui ont besoin de grammaires de Chomsky plus puissantes (Prusinkiewicz et Hanan, 1989). À la Fig. 14, un schéma de re-

lation entre les deux différents types de grammaires est présenté (Prusinkiewicz et Hanan, 1989, page 5).

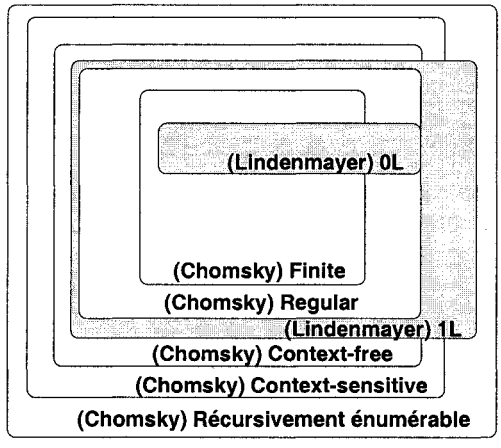


Figure 14 Grammaires de Chomsky et L-Systems

Les L-Systems ont alors commencé à être utilisés pour représenter des plantes. Ce modèle a postérieurement été appliqué aux plantes supérieures, parce que leur croissance et développement sont hautement auto-similaires et donc bien « adaptés » à être représentés par cette classe de modèles (Prusinkiewicz et Lindenmayer, 1990). Des influences environnementales, qui jouent un rôle important dans la croissance des plantes, ont aussi été représentées (Mech et Prusinkiewicz, 1996). Par ailleurs, l'interprétation des L-Systems est approchée d'une façon différente, en se concentrant sur les représentations avec une géométrie rigoureusement définie, ce qui a montré que des L-Systems simples indépendants du contexte pouvaient générer des figures particulières : des fractales (Szilard et Quinton, 1979). Plus tard, des courbes classiques (« *space-filling* », en anglais) sont définies avec l'aide de L-Systems (Siromoney et Subramanian, 1983) ; Prusinkiewicz présenta alors davantage d'exemples de fractales et des structures semblables aux plantes avec des L-Systems : elles ont été obtenues en utilisant une représentation basée sur une tortue de type LOGO et sa contrepartie en 3 dimensions (Prusinkiewicz, 1986, 1987; Smith, 1984).

Le potentiel des L-Systems pour la modélisation de plantes reçoit donc beaucoup d'attention, et notamment Prusinkiewicz et Hanan démontrent leur capacités sous différentes formes (Prusinkiewicz et Hanan, 1989; Hanan, 1992).

Une des caractéristiques les plus étonnantes de cette approche à la modélisation est la génération d'objets complexes (du point de vue infographique) avec des descriptions très concises (Prusinkiewicz et Hanan, 1989; Smith, 1984) ; cette capacité forme la base de l'application des L-Systems à la synthèse d'images (Prusinkiewicz et al., 2003).

1.3.1 Quelques définitions

Formellement, un L-System consiste en un ensemble de règles textuelles appelées *productions* (Prusinkiewicz et al., 2001), ou *règles de production*. Dans la *phase de génération*, ces productions sont appliquées dans une séquence de *pas de dérivations* à une chaîne initiale, appelée un *axiome* (Prusinkiewicz et al., 2001). Donc l'état d'un L-System est codifié dans une chaîne de caractères : un *LString* (Prusinkiewicz et al., 2001). Dans une *phase d'interprétation* subséquente, le LString est converti en sa représentation géométrique (Prusinkiewicz et al., 2001).

Définition 1.1 (L-System). *(Prusinkiewicz et al., 1996, p. 6) Soit $\mathcal{T}$ un alphabet, $\mathcal{T}^*$ l'ensemble de tous les mots sur $\mathcal{T}$, $\mathcal{T}^+$ l'ensemble de tous les mots non-vides sur $\mathcal{T}$. Un L-System est un triplet ordonné $\mathbf{G} = \langle \mathcal{T}, \omega, \mathcal{P} \rangle$, où*

- *$\mathcal{T}$ est* l'alphabet du système
- *ω es un mot non-vide appelé l'*axiome
- *$\mathcal{P} \subset \mathcal{T} * \mathcal{T}^*$ est un ensemble fini de* productions *ou* règles *($\mathcal{T} * \mathcal{T}^*$ est l'ensemble des paires où le premier élément appartient à $\mathcal{T}$ et le élément deuxième appartient à $\mathcal{T}^*$).*

Une règle r est écrite $r = (a, \chi)$ ou encore $r : a \longrightarrow \chi$; a est le *prédécesseur* et χ est le *successeur* de r. Pour chaque élément a de l'alphabet $\mathcal{T}$ il existe, formellement, une

règle $r = (a, \chi) \in \mathcal{P}$; si aucune n'est explicitement présente, $a \longrightarrow a$ (donc $\chi = a$) est implicite.

Définition 1.2 (Dérivation directe). *Un mot* υ

$$\upsilon = \chi_1, \chi_2, \ldots, \chi_m$$

dérive directement *de* μ

$$\mu = a_1, a_2, \ldots, a_m$$

($\mu \Rightarrow \upsilon$*) si (et seulement si)* $\forall a_i$*, il existe une règle* $r_i \in \mathcal{P}, r_i : a_i \rightarrow \chi_i$.

Les dérivations directes dans une grammaire sont notées avec le symbole $\longrightarrow$: $a \xrightarrow{r_i} b$ se lit « b *est obtenue de* a *par l'application de la règle* r_i ».

Définition 1.3. *Un mot* υ *est « généré par* G *en* n *pas »s'il existe* $\mu_0, \mu_1, \ldots, \mu_n$ *tels que :*

a. $\mu_0 = \omega$,

b. $\mu_n = \upsilon$ *et*

c. $\mu_0 \longrightarrow \mu_1 \longrightarrow \ldots \longrightarrow \mu_n$

Regardons le L-System L comme exemple de ces concepts :

$$L :: \begin{cases} \omega = F \\ \mathcal{P} = \{r_1\} \\ r_1 : F \longrightarrow F - F + F \end{cases}$$

Les trois premiers pas de dérivations de L sont :

$$
\begin{aligned}
s_1 (= \omega) &: \quad F \\
s_2 &: \quad F - F + F \\
s_3 &: \quad F - F + F - F - F + F + F - F + F
\end{aligned}
$$

ou

$$
\begin{aligned}
&\omega \quad :: F \\
&\ldots \xrightarrow{r_1} \overbrace{F - F + F}^{F} \ldots \\
&\ldots \xrightarrow{r_1} \overbrace{F - F + F}^{F} - \overbrace{F - F + F}^{F} + \overbrace{F - F + F}^{F}
\end{aligned}
$$

Nous dirons dans ce cas que $s_1 \Rightarrow s_3$ ou bien $\omega \Rightarrow s_3$

1.3.2 Interprétation graphique

Les chaînes (LString) résultantes de la dérivation d'un L-System peuvent être visualisées en interprétant chaque lettre comme une commande pour une tortue **LOGO** ; cette idée a été proposée et développée par Prusinkiewicz en (Prusinkiewicz et Lindenmayer, 1990), (Prusinkiewicz, 1986) et (Prusinkiewicz, 1987). Son idée principale est la suivante : l'*état* de la tortue est défini comme un triplet (x, y, α), où les coordonnées Cartésiennes (x, y) représentent la position de la tortue, et l'angle α représente la direction dans laquelle la tortue « regarde ». La tortue répond aux commandes suivantes :

a. **F(d), f(d)** : déplacement « vers l'avant » une longueur d. L'état de la tortue change à

$$(x' = x + d * \cos(\alpha), y' = y + d * \sin(\alpha), \alpha)$$

Si la valeur de d n'est pas spécifiée (c'est à dire, la commande est **F** ou **f**), une valeur d' par défaut est supposée connue, et les commandes se traduisent alors par **F(d')** et

f(d').

Si la commande est **F**, un segment de droite entre les points (x, y) et (x', y') est tracé.

b. $+(\delta)$: Tourner à *droite* un angle δ. L'état de la tortue devient $(x, y, \alpha + \delta)$. Si δ est omis (commande $+$) on suppose l'existence d'une constante par défaut δ' qui traduit la commande en $+(\delta')$

c. $-(\delta)$: Tourner à *gauche* un angle δ. L'état de la tortue devient $(x, y, \alpha - \delta)$. Si δ est omis, on suppose l'existence d'une constante par défaut δ' qui traduit la commande en $-(\delta')$

Additionnellement, pour décrire les structures de branches retrouvées dans les plantes, Lindenmayer présenta une notation pour représenter des arbres (naturels ou théoriques, comme des graphes) ; les crochets ([et]) sont à la base de cette notation (Lindenmayer, 1968). $\mathcal{T}$ est alors augmentée avec ces deux nouveaux symboles ; la tortue les interprète comme suit :

- [garde l'état de la tortue dans une pile[7].
-] prends l'état qui est le premier de la pile et l'interprète comme l'état actuel.

Définition 1.4 (Interprétation graphique d'un L-System G). *Soient v un LString représentant l'état de la dérivation d'un L-System* G*, un état initial pour la tortue (x_0, y_0, α_0), et paramètres fixes d' et δ'. L'*interprétation *de* G *générée par la tortue est l'ensemble de lignes dessiné par la tortue en lisant v.*

Cette description établit une méthode rigoureuse pour associer des LString à des figures ; elle définit aussi (implicitement) le système de coordonnées utilisé par la tortue :

[7] En informatique, une pile (*stack* en anglais) est une structure de données basée sur le principe **LIFO** (**L***ast* **I***n,* **F***irst* **O***ut*), ce qui veut dire que les derniers éléments ajoutés à la pile seront les premiers à être récupérés. Le fonctionnement est celui d'une pile d'assiettes : on ajoute des assiettes sur la pile, et on les récupère dans l'ordre inverse, en commençant par la dernière ajoutée. (http://fr.wikipedia.org/wiki/Pile_(informatique))

- la tortue regarde initialement dans la direction de l'axe y positif
- l'axe x négatif fait un angle de 90 degrés à partir de la tortue
- l'axe z est défini par la règle de la main droite[8]

Comme exemple, considérons G, un L-System avec des crochets.

$$\mathrm{G} :: \begin{cases} \omega = F \\ r_1 : F \rightarrow F[-F]F[+F][F] \end{cases}$$

Les LString obtenues des trois premiers pas de la dérivation de G sont :

(1) $F[-F]F[+F][F]$

(2) $F[-F]F[+F][F][-F[-F]F[+F][F]]F[-F]F[+F][F][+F[-F]F[+F][F]][F[-F]F[+F][F]]$

(3) $F[-F]F[+F][F][-F[-F]F[+F][F]]F[-F]F[+F][F][+F[-F]F[+F][F]][F[-F]F[+F][F]]...$

$[-F[-F]F[+F][F][-F[-F]F[+F][F]]F[-F]F[+F][F][+F[-F]F[+F][F]][F[-F]F[+F][F]]]...$

$F[-F]F[+F][F][-F[-F]F[+F][F]]F[-F]F[+F][F][+F[-F]F[+F][F]][F[-F]F[+F][F]]...$

$[+F[-F]F[+F][F][-F[-F]F[+F][F]]F[-F]F[+F][F][+F[-F]F[+F][F]][F[-F]F[+F][F]]]...$

$[F[-F]F[+F][F][-F[-F]F[+F][F]]F[-F]F[+F][F][+F[-F]F[+F][F]][F[-F]F[+F][F]]]$

Leur interprétation par une tortue **LOGO** est montrée à la Fig. 15, en regardant la figure à partir d'un point dans l'espace (x, y, z) avec $x = 0, y = 0$ et $z > 0$. La figure à l'itération 4, incluant l'axe de coordonnées, est montrée à la Fig. 16.

[8] Les règles de la main droite et de la main gauche déterminent dans quelle direction un axe, construit comme le produit croisé des deux autres axes, se trouve. La règle de la main droite dit que l'orientation de ce vecteur est déterminée en plaçant les deux premiers vecteurs (x'Ox) et (y'Oy) ensemble à la queue (au « point de départ »), et en ouvrant la main droite la mettre en direction de (x'Ox) ; courber alors les doigts en direction de (y'Oy) : le pouce pointe alors dans la direction de (z'Oz). Pour plus de référence consulter, par exemple, http://mathworld.wolfram.com/Right-HandRule.html

(a) Itération 1 (b) Itération 2 (c) Itération 3 (d) Itération 4

Figure 15 Interprétation géométrique de G

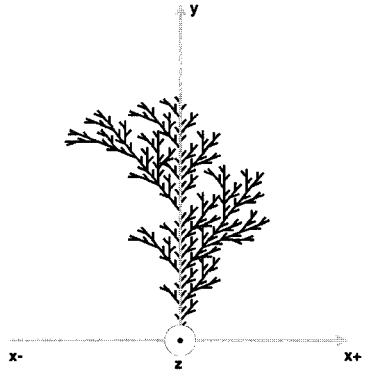


Figure 16 Itération 4 de G, avec les axes de coordonnées

1.3.3 Types de L-Systems

Différents types de L-Systems ont été définis avec différentes motivations (Prusinkiewicz, 1986) ; ils diffèrent principalement en 3 points :

a. **Les règles sont-elles appliquées suivant l'état actuel de l'interprétation ?** Pour les processus naturels qui sont modélisés à l'aide des L-Systems il est parfois avantageux de pouvoir exprimer des changements qui sont liés à un état particulier de l'environnement ; par exemple, en modélisant les différents stages de la vie d'un arbre, nous pourrions vouloir dire que les fruits apparaissent seulement quand certaines conditions structurelles existent dans l'arbre. L'objectif serait alors, dans un L-System G,

de pouvoir exprimer des règles qui s'activent selon une condition existante dans le LString représentant la dérivation de G.

Comme exemple, considérons deux grammaires, $\mathbf{G}_1$ et $\mathbf{G}_2$:

$$\mathbf{G}_1 :: \begin{cases} \omega :: & baaaaaaa \\ p_1 : & b \longrightarrow a \\ p_2 : & a \longrightarrow b \end{cases} \qquad \mathbf{G}_2 :: \begin{cases} \omega :: & baaaaaaa \\ p_3 : & b \longrightarrow a \\ p_4 : & b < a \longrightarrow b \end{cases}$$

Les règles de production de $\mathbf{G}_1$, p_1 et p_2, sont appliquées indépendamment de l'état de l'environnement (c'est à dire, du LString représentant l'environnement de la dérivation de $\mathbf{G}_1$) ; ainsi, sa dérivation sera :

$$\omega :: baaaaaaa \longrightarrow abbbbbbb \longrightarrow baaaaaaa \longrightarrow abbbbbbb \cdots$$

$\mathbf{G}_1$ est dite être « indépendante du contexte ».
Dans le L-System $\mathbf{G}_2$, la règle p_4 se lit « *remplacer a par b si (et seulement si) il succède à un b* » ; donc sa dérivation sera :

$$\omega :: baaaaaaa \longrightarrow abaaaaaa \longrightarrow aabaaaaa \longrightarrow aaabaaaa \cdots$$

Comme on peut l'apprécier, le symbole b « glisse » vers la droite du LString tout au long de la dérivation (une possible interprétation est celle d'un signal qui se propage au long d'une chaîne de symboles, (Prusinkiewicz et Lindenmayer, 1990)). $\mathbf{G}_2$ est appelée « sensible au contexte ».

Plusieurs extensions sensibles au contexte ont été proposées (Prusinkiewicz et Lindenmayer, 1990, pp. 30–31) ; les 2-L-Systems utilisent des productions de la forme

$$r : a_l < a > a_r \longrightarrow \chi$$

où le symbole a (le *prédécesseur stricte* de r) produit χ si et seulement si a est précédé par le symbole a_l et suivi par a_r. a_l est le *contexte gauche* de a, et a_r le *contexte droit* de a. La classe générale de ce type de L-Systems est appelée $(k;l)$-L-Systems, où le contexte gauche est un mot de longueur k et le contexte droit un mot de longueur l (Prusinkiewicz et Lindenmayer, 1990).

Si des règles libres de contexte et sensibles au contexte coexistent, les règles sensibles au contexte prennent précédence sur les règles libres de contexte. Par exemple, en $\mathbf{G}_3$ deux règles (p_2 et p_3) ont comme prédécesseur stricte le symbole a.

$$\mathbf{G}_3 :: \begin{cases} \omega :: & baa \\ p_1 : & b \longrightarrow a \\ p_2 : & a \longrightarrow b \\ p_3 : & b < a \longrightarrow c \end{cases}$$

La dérivation sera

$$\omega :: baa \longrightarrow acb$$

et non

$$\omega :: baa \longrightarrow abb$$

car p_3 a précédence sur p_2

D'un autre côté, si deux règles sensibles au contexte s'appliquent au même symbole, la *plus spécifique* (Déf. 1.5) doit être élue au moment de l'application.

Définition 1.5. *Une règle* r_1, $r_1 : p_1 < a > s_1 \longrightarrow \chi_1$ *est plus spécifique qu'une règle* r_2, $r_2 : p_2 < a > s_2 \longrightarrow \chi_1$ *si*

(a) p_2 *est une chaîne de caractères contenue dans* p_1, *et,*

(b) s_2 est une chaîne de caractères contenue dans s_1

Par exemple, soit $\mathbf{G}_4$

$$\mathbf{G}_4 :: \begin{cases} \omega :: & baaaa \\ p_1 : & b \longrightarrow a \\ p_2 : & b < a \longrightarrow b \\ p_3 : & aab < a \longrightarrow c \end{cases}$$

aura une dérivation

$$\omega :: baaaa \xrightarrow{p_1,p_2} abaaa \xrightarrow{p_1,p_2} aabaa \xrightarrow{p_1,p_3} aaaca \cdots$$

L'apparition du symbole c correspond à l'activation de la règle p_3, qui a donc pris précédence sur p_2 à un moment spécifique de la dérivation.

L'introduction de contexte pour les L-Systems à crochets est plus difficile qu'avec les L-Systems sans crochets, car un LString à crochets ne garde pas l'information de voisinage de segments (Prusinkiewicz et Lindenmayer, 1990). Par exemple, considérons le LString L_1 (Prusinkiewicz et Lindenmayer, 1990, p. 32) :

$$L_1 : ABC[-DE][SG[-HI[+JK]L]MNO]$$

La règle $r : BC < S > G[-H]M$ ne peut être appliquée sur L_1, car, syntactiquement, le contexte gauche de S n'est pas BC, ni son contexte droit est $G[-H]M$; cependant, examinons l'interprétation graphique de r (Fig. 17) et de L_1 (Fig. 18) (Prusinkiewicz et Lindenmayer, 1990, page 30). Géométriquement, les contextes de r effectivement correspondent à la structure de l'arbre (ceci est indiqué à l'aide des ellipses grises). C'est pourquoi, en travaillant avec l'idée de contexte avec les L-Systems à crochets, la procédure devra probablement ignorer quelques symboles pour chercher la correspondance entre les contextes.

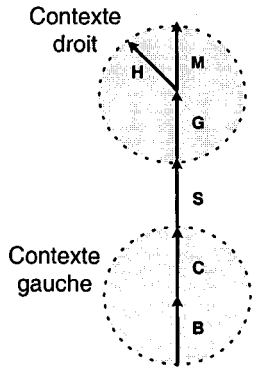


Figure 17 Interprétation graphique de r

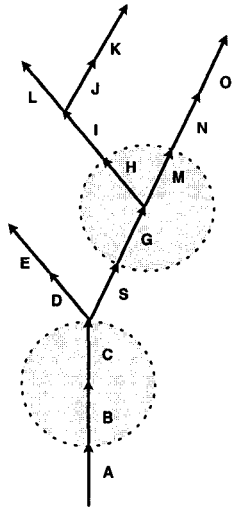


Figure 18 Interprétation graphique de L_1

b. **Existe-t-il plusieurs règles pouvant être appliquées à un même symbole ?** L'interprétation d'un L-System G est un processus déterministe, et donc la figure résultante est toujours la même. Ceci est naturel et souhaitable, mais peut poser un problème pratique, selon l'usage qui est fait de G : si G est pensé comme un modèle d'un type d'arbre peuplant une forêt (par exemple), la scène obtenue avec des répétitions de G aura une apparence trop « monotone », pas assez « réaliste ». Un peu de variation serait la bienvenue pour ce type d'application. Pour cela, l'idée est de pouvoir associer des probabilités aux règles de production.

Regardons comme exemple le L-System suivant :

$$\mathbf{G} :: \begin{cases} \omega :: & & F \\ p_1 : & \{0,33\} & F \longrightarrow F[+F]F[-F]F \\ p_2 : & \{0,33\} & F \longrightarrow F[+F]F \\ p_3 : & \{0,34\} & F \longrightarrow F[-F]F \end{cases}$$

p_1, p_2 et p_3 indiquent que, au moment de remplacer F, il faut « lancer un dé »pour décider si ce symbole sera remplacé par

(a) $F[+F]F[-F]F$ (probabilité associée de $0,33$)

(b) $F[+F]F$ (probabilité associée de $0,33$)

(c) ou par $F[-F]F$ (probabilité associée de $0,34$)

À une profondeur 5 (5 itérations de G), les figures montrées à la Fig. 19 peuvent être obtenues (Prusinkiewicz et Lindenmayer, 1990, page 29).

Figure 19 Interprétation stochastique

Ce type de grammaire est appelée *stochastique* ; si une grammaire n'est pas stochastique, elle est *déterministe*.

c. **Les règles de production sont-elles paramétrisées ?** Pour augmenter la flexibilité dans la modélisation avec les L-Systems, Lindenmayer proposa l'association de paramètres numériques aux règles de production des L-Systems (Lindenmayer, 1974). Ceci permet d'associer des conditions aux règles ; par exemple, considérons A (Prusinkiewicz et Lindenmayer, 1990, p. 42) :

$$A(t) : t > 5 \longrightarrow B(t+1)CD(t^{0,5}, t-2)$$

Cette règle spécifie le remplacement du symbole A par d'autres symboles (B, C, et D) eux aussi paramétrisés, si et seulement si le paramètre t de la règle est supérieur à 5. Donc, par exemple, $A(9)$ serait remplacé par $B(10)CD(3,7)$, et $A(4)$ ne subirait aucune transformation.

Pour les lecteurs intéressés, les détails de la motivation et définition formelle des L-Systems paramétriques sont présentées par Prusinkiewicz et Lindenmayer (Prusinkiewicz et Lindenmayer, 1990, pp. 40–50).

1.3.4 Applications

L'explication détaillée des L-Systems et de ses dérivations, faite dans les sous-sections précédentes, s'appuie fortement dans les recherches faites par Lindenmayer et Prusinkiewicz. Je voudrais maintenant commenter d'autres travaux parallèles effectués avec ce formalisme, ce qui aiderait à voir l'étendue de l'importance des L-Systems dans la modélisation de plantes réelles.

Trois phénomènes naturels sont utilisés pour illustrer l'utilisation de L-Systems : d'abord, les inflorescences, qui sont modélisées comme des modules[9] dans l'espace. Cette application exemplifie l'utilisation des L-Systems pour représenter des structures complexes en

[9] Dans le contexte des L-Systems, « module » représente des unités discrètes, répétées dans le développement de la plante : par exemple, une fleur ou une branche (c.f. (Bell, 1991, page 4), cité dans (Prusinkiewicz et al., 1995))

branches ; deux publications à ce sujet sont (Frijters et Lindenmayer, 1976) et (Frijters et Lindenmayer, 1974).

D'un autre côté, d'autres auteurs proposent des mécanismes pour expliquer les différents types de développement de branches à fleurs observés dans la nature ; celles-ci sont appelées *séquences acropétales*, quand les fleurs progressent d'en bas vers le haut de la plante, et *séquences basipétales*, pour le contraire. Un L-System indépendant du contexte suffit pour décrire des séquences acropétales, tandis que pour modéliser les séquences basipétales, deux mécanismes différents sont invoqués : soit la caractérisation de la vigueur ou potentiel de croissance des nœuds (Luck et Luck, 1994), (Luck et al., 1990), ou le contrôle par l'envoie de signaux (Frijters et Lindenmayer, 1974), (Janssen et Lindenmayer, 1987).

Un autre phénomène amplement observé dans la nature est l'adéquation de la structure des plantes supérieures aux changements de leur environnement : ainsi, les arbres peuvent compenser la perte de branches par la croissance plus vigoureuse d'autres branches. Un modèle qui capture ce phénomène a été proposé dans (Borchert et Honda, 1984). Une idée plus générale a été publiée dans (Hart et al., 2003), où les auteurs proposent un logiciel que permet de modéliser les influences environnementales sur des arbres simulés. Cet effort aboutit à un outil prédictif qui peut être utilisé dans l'industrie de l'infographie et aussi de la gestion de forêts (Hart et al., 2003).

Un exposé détaillé d'un état de l'art des développements dans le domaine peut aussi être consulté dans (Prusinkiewicz et al., 1995).

1.4 Systèmes Itérés de Fonctions

La géométrie fractale travaille avec des sous-ensembles « complexes » d'espaces géométriques « simples » (comme $\mathbb{C}$, les nombres complexes, ou $\mathbb{R}^2$, l'espace bidimensionnel réel) ; munis d'une *distance*, ceux-ci sont appelés des *espaces métriques* (Barnsley, 1993).

Dans un espace métrique (X, d) -espace X muni d'une distance d-, l'*espace des fractales* $\mathcal{H}$ est l'ensemble des sous-ensembles compacts de X (Barnsley, 1993). Dans $\mathcal{H}$, un *système itéré de fonctions* (IFS) est un ensemble de contractions affines : fonctions de la forme $ax + b$ qui font que les points en $\mathcal{H}$ se rapprochent. Les points fixes de ces fonctions sont des figures fractales.

Comme exemple, un système itéré de fonctions composé de 4 sous-fonctions est présenté en Éq. (1.1) (Barnsley, 1993, p. 86)

$$\begin{cases} f_1(x,y) = \begin{bmatrix} 0,85 & 0,04 \\ -0,04 & 0,85 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0,00 \\ 1,60 \end{bmatrix} \\ f_2(x,y) = \begin{bmatrix} -0,15 & 0,28 \\ 0,26 & 0,24 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0,00 \\ 0,44 \end{bmatrix} \\ f_3(x,y) = \begin{bmatrix} 0,20 & -0,26 \\ 0,23 & 0,22 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0,00 \\ 1,60 \end{bmatrix} \\ f_4(x,y) = \begin{bmatrix} 0,00 & 0,00 \\ 0,00 & 0,16 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \end{cases} \tag{1.1}$$

L'attracteur de ce système (les points fixes) forment une figure de *fougère*, présentée à la Fig. 20. Le sommet de la fougère est le point fixe de f_1, et les points extrema des deux branches les plus basses sont les images du sommet sous f_2 et f_3 (Wagon, 1991).

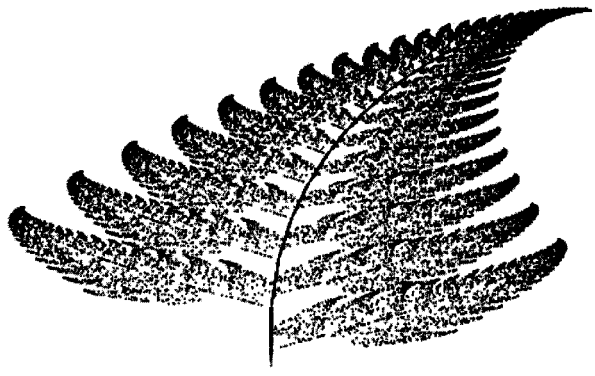


Figure 20 Fougère de Barnsley

Dans ce chapitre, les bases théoriques des outils utilisés dans cette thèse ont été présentées ; l'idée de *figures fractales* et leur relation avec les figures d'arbres et plantes naturelles a été introduit (section 1.1). Une présentation de quelques modèles pour l'architecture des plantes (section 1.2) nous a mené aux deux formalismes principalement étudiés pour la représentation des plantes : les L-Systems (section 1.3) et les systèmes itérés de fonctions (section 1.4).

Dans le prochain chapitre, le travail antérieur spécifique à ce travail de recherche va être introduit : nous allons observer comment la question du *problème inverse pour la modélisation de processus de développement* a été approchée dans la littérature scientifique.

CHAPITRE 2

LE PROBLÈME INVERSE : ÉTAT DE L'ART

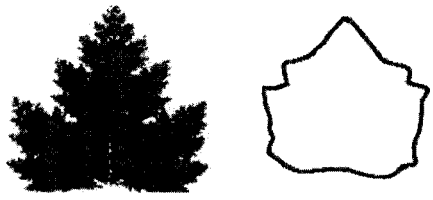
Ce chapitre présente un sommaire de la recherche publiée à propos de la résolution du problème inverse pour une (un ensemble de) figure(s) fractale(s) ; mon objectif est de présenter le travail lié aux deux formalismes identifiés dans le chapitre précédent (les IFS et les L-Systems), pour ainsi pouvoir comparer ses avantages et désavantages.

Résolution du problème inverse avec IFS.

Dans la littérature concernant la recherche effectuée avec les IFS (ce formalisme a été défini dans ce document à la section 1.4, page 34) nous retrouvons des solutions au problème de génération d'un modèle approximant une figure donnée ; en effet, les modèles IFS sont construits en utilisant un théorème clé caractérisant des objets auto-similaires, qui établit que si un objet peut être décomposé en un nombre fini de copies réduites de lui-même (en d'autres mots, s'il est auto-similaire), il est complètement décrit par un ensemble de transformations qui transforment l'objet comme un tout dans ses parties (Hutchinson, 1981). Le résultat de Hutchinson a été augmenté pour l'appliquer à des objets qui sont *presque* auto-similaires : c'est le *théorème du collage* (Barnsley et Demko, 1985).

Le *théorème du collage* énonce que, pour trouver un IFS dont l'attracteur « ressemble » à un ensemble (figure) donné(e), il faut simplement trouver un ensemble de contractions dont l'union (ou *collage*) est « suffisamment proche » de l'ensemble cible : le théorème alors garantit que l'attracteur de l'IFS est aussi « suffisamment proche » de l'ensemble cible. Un exemple visuel est présenté à la Fig. 21 (repris de (Barnsley, 1993, chapitre 4)).

Avec cette idée, le problème inverse peut maintenant être défini comme un problème d'optimisation :



(a) Approximation (b) Cible

Figure 21 Exemple d'approximation par IFS

Définition 2.1 (IFS : le problème inverse comme optimisation). *Dans l'espace des contractions, trouver un sous-ensemble tel que la distance entre son collage et l'ensemble cible est plus petite qu'une distance pré-définie* d_1.

Des méthodes d'optimisation ont effectivement été appliquées pour résoudre ce problème ; une méthode est présentée qui permet à l'usager de manipuler un objet fractal défini par un type spécial d'IFS (décrivant les associations entre les parties d'une fractale à l'aide d'un graphe) (Cart, 2003) . Les solutions sont générées en temps réel, résultant en un système interactif de modélisation pour faire le design d'objets avec apparence naturelle. Une autre méthode est proposée pour améliorer la représentation et les méthodes d'optimisation utilisées pour générer les figures (Collet et al., 2000). Des algorithmes évolutifs ont été utilisés pour résoudre ce problème inverse pour IFS (Lutton, 1999).

Le problème inverse des L-Systems.

Le formalisme des L-Systems a été utilisé pour résoudre des problèmes du *monde réel* ; par exemple, en architecture, pour obtenir des solutions de design nouvelles et inexplorées (Bentley, 1999, chap. 14). Chaque solution candidate (une grammaire L-System) codifie une solution au problème, laquelle est alors évaluée suivant une fonction objectif qui compare ses caractéristiques à une liste de restrictions que toutes les solutions doivent rencontrer ; les auteurs (Coates, Broghton, et Jackson) illustrent leur méthode en faisant le design de tunnels de vent et des formes en spirale.

Pour le problème plus spécifique concernant l'utilisation de L-Systems pour générer un modèle pour une figure, Lindenmayer et Prusinkiewicz proposent d'appliquer des méthodes prises de la théorie de grammaires sur graphes, appelées *remplacement d'arêtes* et *remplacement de nœuds*, pour générer des types de courbes (Prusinkiewicz et Lindenmayer, 1990). Comme exemple, ils s'attaquent à la construction de figures FASS (*space-Filling, self-Avoiding, Simple and self-Similar*) : des courbes qui remplissent un espace (tout le carré unité en deux dimensions, ou le cube unité en trois dimensions, ou un hypercube en n-dimensions)[1]. McKenna montra comment générer ce type de courbes (McKenna, 1994) ; des exemples sont montrés à la Fig. 22.

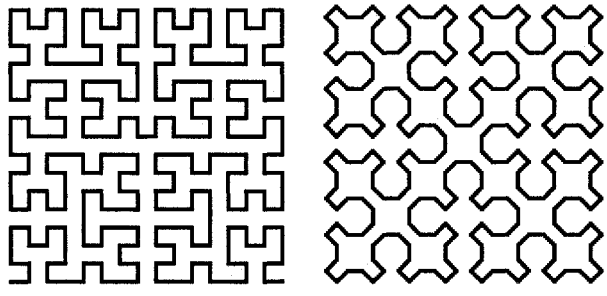


Figure 22 Deux courbes FASS

La courbe de gauche a été proposée par Hilbert et celle de droite par Sierpinski ; repris de http://www.seanet.com/~garyteachout/fill

La limitation de cette méthode par rapport au travail présenté pour cette thèse est que leur approche reste limitée à construire une fractale d'un certain type et non à construire une certaine figure prédéterminée.

Concernant l'approximation de figures, les L-Systems ont été utilisées pour décrire graphiquement la circulation du sang dans une rétine humaine (Vanyi et al., 2000). Un algorithme évolutif fait la synthèse d'une description grammaticale d'une photographie des vaisseaux sanguins (représentés à la Fig. 23). Les auteurs ont défini les différentes formes que peut

[1] Repris de Wikipedia, http://en.wikipedia.org/wiki/Peano's_curves

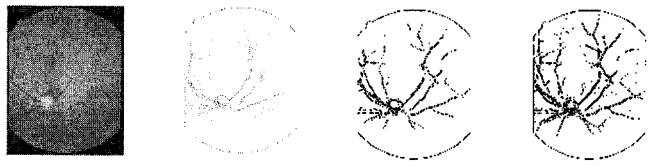


Figure 23 Vaisseaux de la rétine

avoir un vaisseau sanguin ; ils ont alors fait la traduction en règles L-System (les appelant **B**, **Y**, et **T**) et ont alors défini un cadre pour la forme que les grammaires L-System doivent avoir (grammaire type 2.1)

$$\mathbf{G} :: \begin{cases} \mathcal{T} = & \{S, F(a), +(a), -(a), [,], \mathbf{B}(a,b), \mathbf{Y}(a,b,c), \mathbf{T}(a,b)\} \\ \text{axiome } \omega = & S \\ p_1 : S & \longrightarrow \{S, \mathbf{B}(a,b), \mathbf{Y}(a,b,c), \mathbf{T}(a,b), [,]\}^* \\ p_2 : \mathbf{B}(a,b) & \longrightarrow S + (a)F(b) \\ p_3 : \mathbf{Y}(a,b,c) & \longrightarrow F(a)[+(b)S][-(c)S] \\ p_4 : \mathbf{T}(a,b) & \longrightarrow F(a)[+(b)S]S \end{cases} \tag{2.1}$$

$(a, b, c \in \mathbb{R})$.

Cette approche, extrêmement intéressante, est sérieusement diminuée par une restriction sévère : la seule différence entre les différentes grammaires suivant la grammaire type (2.1) est l'axiome ω, car les règles sont fixées. L'algorithme évolutif « optimise » seulement le choix de ω pour obtenir une grammaire ayant la meilleure interprétation possible pour la figure d'entrée. Il navigue alors à travers un choix très restreint de grammaires possibles (solutions au problème).

Dans le même contexte, le formalisme des L-Systems, combiné avec la Programmation Génétique (PG), a été utilisé pour trouver les règles de production pour une grammaire s'interprétant comme une forme spécifique (Koza, 1993), (Kokai et al., 1998). Il est spécifié qu'une solution possible G au problème doit suivre les restrictions suivantes :

- L'alphabet doit être $\mathcal{T} = \{F, +, -\}$
- G est déterministe et libre de contexte.
- L'axiome de G est fixe et pré-défini.
- L'angle de branchement pour les symboles $+$ et $-$ est fixe et pré-défini.
- G a seulement 1 règle de production.
- G est itérée exactement 2 fois avant d'être interprétée.

Donc, dans ce contexte, et avec ces conditions, « *résoudre le problème inverse* » est synonyme à « *trouver* **la** *règle de production d'un L-System donné* » ; cette règle de production est la composition de fonctions primitive et des symboles terminaux : un programme. En d'autres mots, ce travail est plus un exemple des capacités de la méthode de GP qu'une méthode générale pour résoudre le problème inverse.

Dans un autre travail, Prusinkiewicz et Lindenmayer présentent une méthodologie pour construire des modèles de plantes réelles (Prusinkiewicz et Lindenmayer, 1990, chap. 3 et suivants) ; elle est basée sur deux points :

a. L'observation du développement de plantes réelles : le(s) chercheur(s) doivent observer l'espèce de plantes à modéliser pour ainsi identifier les points principaux du processus de développement qui produit la forme observée.

b. L'utilisation de 3 niveaux de spécification de modèles : le niveau le plus abstrait (« *L-System partiel* ») utilise la notation des L-Systems indépendants du contexte pour spécifier les possibilités structurelles que la plante peut prendre. Le prochain niveau est celui du « schéma L-System », qui spécifie comment les différentes possibilités de développement du modèle s'expriment. Et finalement le modèle complet incorpore de l'information géométrique pour en faire la visualisation complète.

La visualisation de la croissance de structures à branches est l'objectif des auteurs dans (Linsen et al., 2005) ; cette visualisation est supposée être celle d'un arbre prototype d'une

certaine espèce, et celui-ci est supposé s'adhérer à des données réelles, mesurées dans des situations réelles (Linsen et al., 2005). Leur effort se base à construire un modèle d'arbre similaire aux arbres existants, puis ensuite en faire l'extension en couplant des règles de branchements avec des règles dynamiques de croissance. La faiblesse de ce travail vient du fait que cette construction initiale du modèle n'est pas un processus automatisée, mais un travail explicitement fait par les chercheurs. Dans cette thèse, l'objetif est d'effectuer ce pas de façon automatique.

Le problème inverse pour la dérivation de L-Systems est approchée avec des algorithmes génétiques en (Runqiang et al., 2002). Cette méthode va de la représentation de la structure en branches, la traduction de L-Systems à leur structure en arbres, comparaison des structures en branches, et l'application d'algorithmes génétiques. Additionnellement, des mécanismes de réparation sont aussi appliqués pour trouver les modèles en L-Systems qui mieux s'approchent des données observées (Runqiang et al., 2002). La faiblesse que je trouve dans ce travail (et que j'essaie d'éviter dans la formulation de cette thèse) provient justement de ce mécanisme de réparation : l'algorithme génétique génère des solutions, qui sont alors « réparées » par celui-ci ; cette réparation n'est pas évidente, et c'est mon opinion que cela introduit des résultats qui ne proviennent pas de la méthode comme telle, mais de l'artifice créé par la réparation. Il faut donc investir beaucoup de temps et de connaissances pour bien la définir, ce qui fait en sorte que la méthode n'est pas automatique, mais en fait très guidée par ce mécanisme.

La recherche entamée dans cette thèse s'appuie et prend son inspiration des trois derniers travaux mentionnés ci-haut : (Prusinkiewicz et Lindenmayer, 1990), (Linsen et al., 2005) et (Runqiang et al., 2002). L'objectif général du projet de recherche que nous entreprenons est de reprendre leurs meilleures idées pour aboutir à créer une méthode automatique qui fasse le moins possible appel au chercheur humain pour générer les modèles. Cette idée est développée et expliquée dans ses détails dans le prochain chapitre.

CHAPITRE 3

LE PROBLÈME ET SA SOLUTION

Jusqu'à ce point, j'ai décrit la motivation du projet, les outils nécessaires à la résolution du travail de recherche, et la recherche existante dans la littérature sur le sujet. Dans ce chapitre le problème à résoudre est formalisé (section 3.1), en montrant comment celui-ci peut être décrit avec les concepts des chapitres précédents ; le choix des L-Systems à crochets comme formalisme à utiliser pour cette thèse est ensuite présenté et justifié (section 3.2). Après un bref commentaire sur les forces et faiblesses de ce choix, un schéma particulier est proposé comme formalisme pour décrire plantes et arbres ; ce schéma est alors utilisé pour définir la méthode à utiliser pour résoudre le problème (section 3.3).

3.1 Formalisation du problème

Le problème s'intéresse à la génération d'un modèle décrivant un *processus de développement* ; par exemple, la Fig. 24 pourrait être le processus cible à approximer.

(a) (b) (c) (d)

Figure 24 Un processus de développement

Donc, pour commencer, une définition de ce qu'est un *processus de développement* est requise.

Un processus de développement pour une structure en branches S est un ensemble d'instantanées de la forme 3D de S, obtenues aux instants $\{t_1, \ldots, t_i, \ldots, t_n\}$ (en unités arbitraires, mais fixes) ; chaque instantanée est une projection (3D à 2D) définie selon un *point de vue* p_i. Pour définir la notion de "« point de vue » je me dois d'expliquer comment les interprétations tridimensionnelles des objets sont projetées dans un espace 2D (l'écran d'un ordinateur, par exemple). L'espace en 3 dimensions est associé à un système de coordonnées, ∇, fixé à l'avance (soit par la méthode utilisée pour générer l'objet à projeter, soit par une convention quelconque, selon l'application particulière) ; à partir de ∇, les projections sont équivalentes à prendre des photographies avec une caméra regardant cet objet, dans le monde en trois dimensions (Foley et al., 1990, pages 229–284) ; trois paramètres de cette caméra définissent la construction d'un prisme rectangulaire de visualisation :

a. Sa position : où se situe la caméra dans ∇.

b. Son point d'intérêt (aussi appelée *distance de mise au point* en jargon photographique) : c'est le point sur ∇ où l'image est complètement nette.

c. Sa profondeur de champ : en photographie, elle définit une zone de netteté autour de la distance de mise au point. De façon équivalente, cette zone est délimitée par une distance *en arrière* du point d'intérêt, et une distance *en avant* du point d'intérêt.

La construction du prisme de projection suit les étapes suivants :

a. le plan perpendiculaire au vecteur qui va du point d'intérêt à la caméra est **le plan de projection**.

b. les distances définissant la profondeur de champ sont appelées F, pour la distance en avant du point d'intérêt, et B, pour la distance en arrière. Les plans parallèles au plan de projection, situés à une distance $+F$ et $-B$, définissent les plans avant et arrière du prisme.

c. Autour du point d'intérêt, l'image est nette d'une distance verticale h (donc $h/2$ au dessus et en dessous du point d'intérêt) et d'une distance horizontale w (donc $w/2$ à droite et à gauche du point d'intérêt). Les plans perpendiculaires au plan de projection, aux distances mentionnées, définissent les plans complétant le prisme.

L'explication graphique de ces détails est montrée à la Fig. 25, en supposant que la caméra est sur l'axe x positif de ∇ et que le point d'intérêt est l'origine $(0, 0, 0)$. Cette méthode

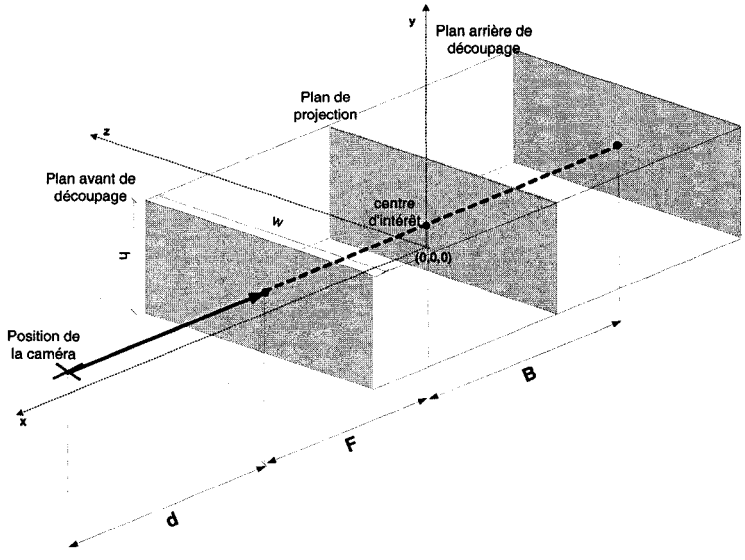


Figure 25 Prisme de projection en 3D

projette tout ce qui est à l'intérieur du prisme, et rien de ce qui se retrouve à l'extérieur (une différence avec la photographie « usuelle », qui montre les détails à l'extérieur, mais flous)

En utilisant cet espace 3D, la position de l'observateur est spécifiée par la Déf. 3.1

Définition 3.1 (Point de vue de l'observateur). *Soit* ∇ *un système de coordonnées orthogonal 3D ; dans* ∇, *un* point de vue $\mathcal{P}$ *est un ensemble de 6 valeurs* $\{p, c, F, B, w, h\}$ *où*

a. p *est la position de la caméra,* $p = (x_p, y_p, z_p) \in \nabla$

b. *c est le « centre d'intérêt », $c = (x_c, y_c, z_c) \in \nabla$*

c. *F et B sont des distances, appelées plans avant et arrière de découpage.*

d. *w et h sont la largeur et la hauteur (respectivement) du canevas de projection.*

Sachant ce qu'est un point de vue, nous pouvons définir un contexte spatio-temporel (Def. 3.2) pour arriver à un processus de développement (Def. 3.3).

Définition 3.2 (Contexte spatio-temporel Υ). *Un contexte spatio-temporel*

$$\Upsilon = \{\Upsilon_1, \ldots, \Upsilon_n\} = \{(t_1, p_1), \ldots, (t_n, p_n)\}$$

est une liste ordonnée de paires $\Upsilon_i = (t_i, p_i)$ où :

- *t_i est une représentation d'un instant de temps ; $t_i \in [0, \infty]$*
- *p_i définit un point de vue dans l'espace*

Les éléments en Υ sont ordonnées par leur élément temporel t_i : $t_1 \leq t_2 \ldots \leq t_n$

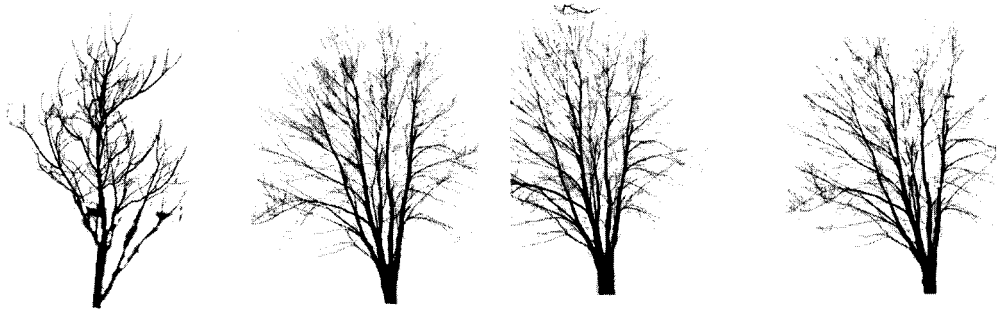
Définition 3.3 (Processus de développement). *Le processus de développement d'un objet 3D, $\mathcal{S}$, sous un contexte Υ, $\mathcal{D}_{(\mathcal{S},\Upsilon)}$, est une séquence $\mathcal{L}_f$ de figures 2D*

$$\mathcal{D}_{(\mathcal{S},\Upsilon)} = \mathcal{L}_f = \{f_{(\Upsilon_1)}, f_{(\Upsilon_2)}, \ldots, f_{(\Upsilon_k)}, \ldots, f_{(\Upsilon_n)}\}$$

$f_{(\Upsilon_i)}$ correspond à la visualisation de S sous $\Upsilon_i = (t_i, p_i)$: c'est la projection en 2D de $\mathcal{S}$, vue du point de vue p_i, à l'instant t_i.

Faisons la formalisation du processus montré en Fig. 24 (repris en Fig. 26). Cette formalisation nécessite la spécification du contexte spatio-temporel responsables de ces figures ; ayant 4 figures en 2D, le contexte Υ aura la forme suivante :

$$\Upsilon = \{\Upsilon_1, \Upsilon_2, \Upsilon_3, \Upsilon_4\} = \{(t_1, p_1), (t_2, p_2), (t_3, p_3), (t_4, p_4)\}$$



(a) En Υ_1 (b) En Υ_2 (c) En Υ_3 (d) En Υ_4

Figure 26 Un processus de développement (repris de Fig. 24)

Deux choix s'imposent pour compléter cette définition :

a. **Le temps.** Un choix (d'une certaine façon arbitraire) doit être fait par rapport au temps : ceci est dû au fait que selon l'espèce à modéliser les changements sont évidents à des périodes de temps différentes. Il y a des espèces dont le cycle de vie se situe dans une même saison, et donc leurs changements sont évidents en peu de temps ; d'autres vivent beaucoup plus longtemps, et leurs changements se montrent d'une année à la suivante. Ici, nous posons l'*année* comme unité de temps ; la première photo de la Fig. 26 a été prise au printemps 2005, donc $t_1 = 0$ sera *printemps 2005*, et $t_2 = t_3 = t_4 = t_1 + 1 = 1$, *printemps 2006*. Ce choix est complètement arbitraire : j'aurais pu choisir le *mois* comme unité temporelle, et *janvier 2005* comme *début du temps*, et alors t_1 (*avril 2005*) serait 3, et $t_2 = t_3 = t_4 = t_1 + 12 = 15$ (*avril 2006*).

b. **Les points de vue.** Ici, c'est le choix d'un système de coordonnées qui s'impose, pour ainsi pouvoir caractériser la position spatiale des arbres. Selon l'objectif poursuivi en modélisant cette scène, ce choix pourrait changer, mais la démarche générale à suivre consiste à chercher des points fixes stables qui puissent servir de repères naturels. Supposons pour cet exemple que certains points géographiques ne changent pas dans le décor où les arbres sont situés : des repères naturels, comme des mon-

tagnes, ou des repères artificiels, comme un bâtiment, un phare, etc. Avec ces repères il serait donc possible de construire un système orthonormé ∇ et l'utiliser pour établir les coordonnées des points de vue.

Pour cet exemple, l'origine des axes $(0, 0, 0)$ est choisi comme la base du tronc des arbres, et l'unité choisie est le mètre. Ces détails sont regroupés au Tableau I.

Tableau I

Point de vue de l'exemple (Fig. 26)

	p_1	p_2	p_3	p_4
Position de la caméra p	$(30, 0, 2)$	$(30, 1, 2)$	$(30, -15, 2)$	$(30, +22, 2)$
Centre d'intérêt c	$(0, 0, 0)$	$(0, 0, 0)$	$(0, 0, 0)$	$(0, 0, 0)$
(F, B)	$(10, -10)$	$(10, -10)$	$(10, -10)$	$(10, -10)$
(w, h)	$(16, -16)$	$(16, -16)$	$(16, -16)$	$(16, -16)$

3.2 Choix d'un formalisme

Une fois le problème présenté, le point suivant consiste à établir quel type de formalisme sera utilisé pour le résoudre ; nous avons présenté deux d'entre eux dans le chapitre 1 : les L-Systems (section 1.3, page 19) et les IFS (section 1.4, page 34).

3.2.1 Choisir entre L-Systems et IFS

Pour la thèse, j'ai décidé de travailler avec les L-Systems ; la raison de ce choix est importante à saisir : un L-System décrit la morphogénèse d'une forme (individu) et s'applique à des structures qui changent de façon temporelle. Les modèles pour faire la représentation des plantes doivent être à la fois simples et suffisamment puissants pour pouvoir exprimer le processus développemental qui s'ensuit, avec ses particularités et ses formes (Prusinkiewicz, 1993) : de la même façon qu'une fractale n'est pas intéressant si nous oublions le processus dynamique qui l'a généré, un modèle de plante n'est pas intéressant si sa croissance (son développement) n'est pas incluse dans le modèle. Comme exemple

de cette idée, je présente le « développement » d'une figure d'arbre artificiel représenté avec un IFS (Fig. 27). Bien que la figure « finale » résultante suite à ce développement se

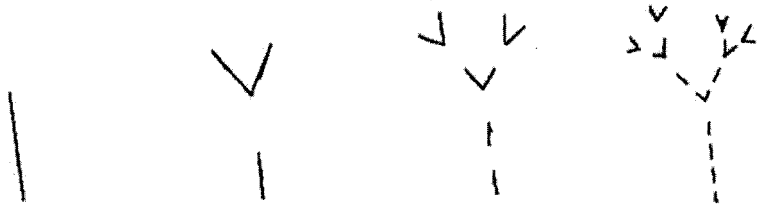


Figure 27 IFS : développement d'une figure d'arbre (1)

Le temps augmente de gauche à droite

rapproche de la figure d'une plante, celle-ci n'est pas complète et ne représente pas une figure plausible biologiquement. Nous pouvons par contre changer la définition du IFS pour obtenir une figure « unie » à la fin : le résultat est montré à la Fig. 28. Bien que la figure

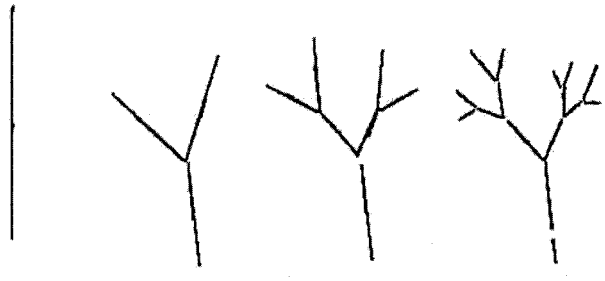


Figure 28 IFS : développement d'une figure d'arbre (2)

Le temps augmente de gauche à droite

résultante dans ce cas a une forme naturelle acceptable, le processus de développement montré ne semble pas naturel : en particulier, plus le temps avance, plus les figures sont petites.

Faire le design d'un L-System pour accomplir la même tâche donne comme résultat le processus présenté à la Fig. 29. Avec des L-Systems, les figures intermédiaires correspondent à ce que nous pourrions nous attendre à observer dans le monde naturel. Ceci est la raison principale d'avoir choisi le formalisme des L-Systems comme base pour ce travail.

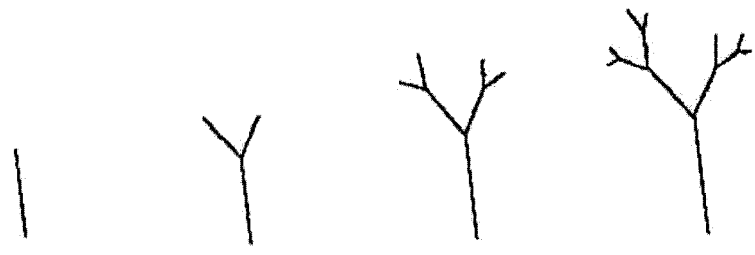


Figure 29 L-System : Développement d'une figure d'arbre

Le temps augmente de gauche à droite

Dans le cadre de ce formalisme j'ai mentionné, à la page 25, que Lindenmayer définit les « L-Systems à crochets » pour pouvoir représenter les structures en branches des arbres et plantes ; il est donc naturel de choisir ce formalisme pour ce travail.

L'expérimentation avec cette méthode de visualisation a été effectuée à l'aide d'un environnement programmé exclusivement pour ce travail. En effet, pour cette thèse il est apparu essentiel d'avoir un environnement qui soit (a) intuitif, simple à comprendre et à naviguer, et (b) suffisamment complexe pour nous permettre de faire une bonne visualisation et maniement des L-Systems. Ceci m'a motivé à faire le design et l'implantation d'un environnement pour la visualisation et l'interaction avec des L-Systems. Ses caractéristiques principales sont :

- Des grammaires L-System de différents types peuvent être définies : (non) paramétrique, déterministe ou stochastique, indépendante du contexte ou sensible au contexte.
- Les modèles sont spécifiés en 3 dimensions.
- Tous les paramètres de visualisation peuvent être changés par l'usager. L'environnement envoie continuellement des *indices* concernant la position dans l'espace de la caméra.
- L'environnement a un module pour projeter ces objets en 2D (équivalent à *prendre des « photos »* (des instantanées) d'une certaine position ; ces projections peuvent

alors être manipulées comme désiré. Un module de visualisation permet de manipuler les photos acquises.

- Le code source est disponible à l'adresse http://www.livia.etsmtl.ca/people/costa

L'environnement est divisé en trois parties principales : (1) le **module de description 3D**, qui permet la définition d'un objet avec un L-System (2) le **module de visualisation**, qui permet la visualisation des instantanées de ces objets (les projections 2D) et (3) le **module de synthèse**. La description détaillée se trouve à l'Annexe II ; ici je présente les fonctions principales de ces modules.

a. **Module de description 3D** : Montré à la Fig. 30, l'idée du module de description 3D est de présenter un environnement graphique facile à utiliser pour permettre à l'usager de définir, avec des grammaires L-System, des modèles pour des objets en trois dimensions et, en même temps, de fournir une façon de visualiser comment ces objets changent avec le *temps*. Les différentes sections de cette présentation (marquées [1] à [8] à la Fig. 30) sont décrites ici :

   - La section [1] permet l'entrée d'une description basée en L-System par l'usager.
   - La section [2] permet à l'usager de choisir entre différents types de L-Systems.
   - La section [3] permet la définition des paramètres par défaut pour une grammaire.
   - La dérivation du LString pour une itération donnée est montrée à la section [4]
   - La section [5] montre la figure projetée en 2D.
   - Différents paramètres, utilisées surtout pour des fins de visualisation pour le programmeur et pour faire la sauvegarde et la récupération de données, sont spécifiés à la section [7].
   - La section [8] permet de prendre des « photos » (instantanées) des projections.

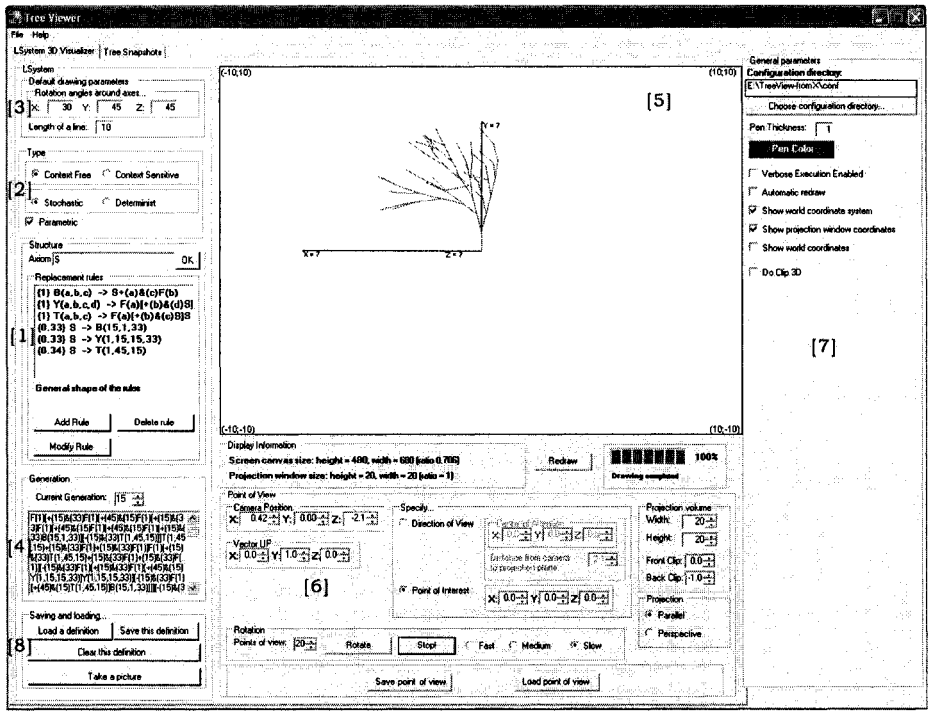


Figure 30 Module de description 3D

b. **Le module de visualisation** : En utilisant un répertoire spécifique pour une figure 3D, ce module charge et fait la visualisation des figures 2D gardées dans celui-ci. Les parties principales de cette section sont montrées à la Fig. 31. La section [1] contient la définition du L-System 3D qui a généré la photo ; la section [2] montre la position de la caméra quand la photo a été prise. Ces deux sections sont pour « lecture seulement » : elles sont montrées seulement comme référence pour l'usager.

La section [3] contient les mêmes options que pour le module 3D et le canevas (section [4]) est défini de la même façon. Section [5] contient une identification pour l'instantanée qui est montrée.

c. **Le module de synthèse** permet de faire des expériences avec un fichier de configuration.

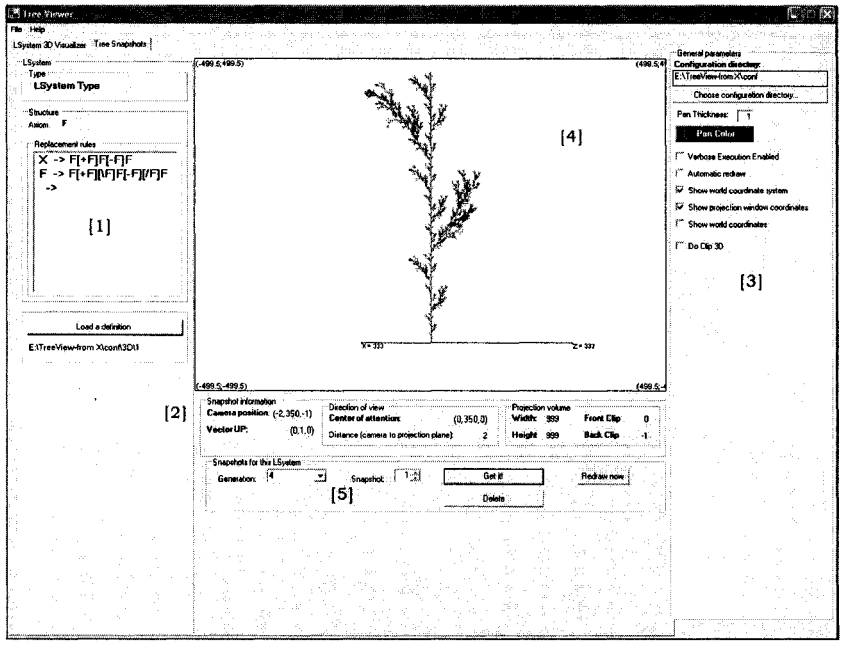


Figure 31 Environnement de visualisation

Avec cet environnement, le concept du prisme de visualisation (tel que présenté à la Fig. 25) a pu être testé ; l'espace en 3 dimensions a, dans le cas des L-Systems, des repères définis par le système de coordonnées utilisé par la tortue en faisant l'interprétation graphique du L-System étudié (cf. 1.3.2, page 24 ; un exemple est présenté aux Figures 15 et 16, page 27).

Comme exemple, la visualisation d'un squelette d'arbre 3D (généré avec une grammaire L-System) est présentée ci-après. Le point de vue est changé pour montrer comment le choix du prisme modifie la projection de l'objet : pour commencer, le point de vue est défini comme au Tableau II ; la projection est alors telle que montrée à la Fig. 32

Tableau II

Définition d'un point de vue

Position de la caméra p	$(30,0,0)$	(F,B)	$(6,-6)$
Centre d'intérêt c	$(0,0,0)$	(w,h)	$(50,50)$

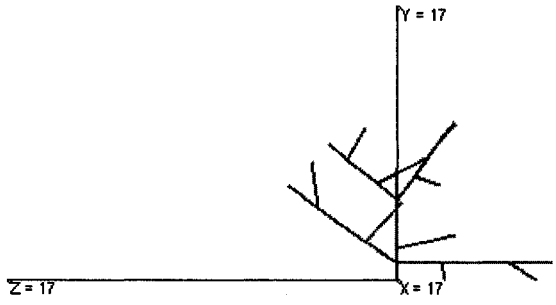


Figure 32 Exemple de visualisation (I)

Modifions le prisme pour voir seulement ce qui est *derrière le plan de projection* ; la façon de faire est de définir $F = 0$. La projection devient alors comme à la Fig. 33(a). Pour montrer seulement ce qui est *devant le plan de projection*, on définit $B = 0$. La projection devient alors comme à la Fig. 33(b).

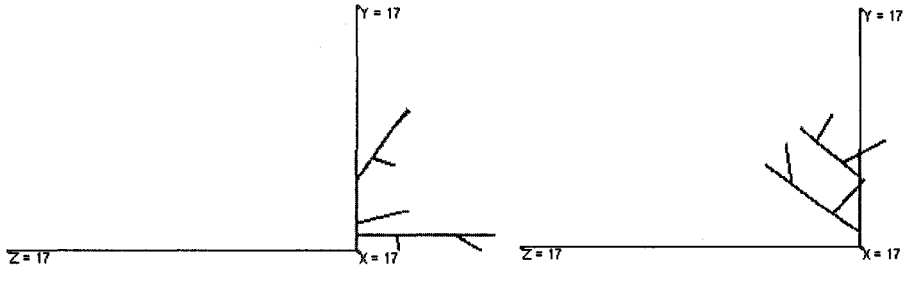


Figure 33 Parties *en avant* (a) et *en arrière* (b) de l'arbre

Si mon point de vue n'est pas directement sur la « racine » de l'arbre, mais un peu en avant de celle-ci (disons, à $c = (1, 0, 0)$), la projection de tout ce qui est « devant le plan de projection » devient des segments de droite sans lien : voir Fig. 34 pour $(F, B) = (6, 0)$

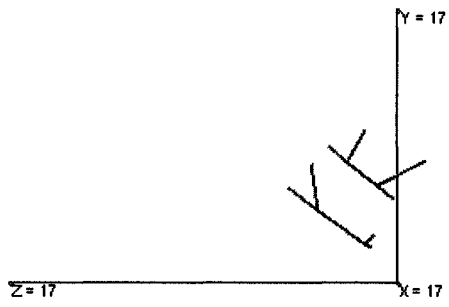


Figure 34 Exemple de visualisation (II)

3.2.2 Φ : Une formalisation syntaxique des formes d'arbre

Tel que mentionné en 1.3 (page 19), Lindenmayer a fait l'extension de la définition originale des L-Systems pour permettre la modélisation de plantes d'ordre supérieur avec ce formalisme ; cette extension a produit la famille des L-Systems à crochets. L'expérimentation avec l'environnement graphique antérieurement présenté nous a permis de voir que les L-Systems à crochets constituent un ensemble très bien adapté à notre problème, mais qu'une spécification plus fine sera nécessaire pour ce travail ; particulièrement, parce qu'*il y a des L-Systems à crochets qui ne représentent pas des arborescences*. Par exemple, prenons G, une grammaire définie de la façon suivante :

$$\mathbf{G} :: \begin{cases} \omega = F \\ r_1 : F \longrightarrow Ff[F] \end{cases}$$

Les trois premières dérivations de G produisent :

$$\begin{aligned} F &\xrightarrow{r_1} Ff[F] \\ \ldots &\xrightarrow{r_1} Ff[F]f[Ff[F]] \\ \ldots &\xrightarrow{r_1} Ff[F]f[Ff[F]]f[Ff[F]f[Ff[F]]] \end{aligned}$$

ce qui est interprété graphiquement tel que montré à la Fig. 35. Le développement observé ne correspond évidemment pas à une arborescence (entre autres, le « tronc » n'est

Figure 35 Dérivations de **G**

pas continu). En d'autres mots, l'affirmation *toute structure générée par un L-System à crochets est représentative d'une structure naturelle* n'est pas vraie ; comme notre objectif est de modéliser *le développement d'une plante*, nous avons alors choisi de prendre un sous-ensemble de l'ensemble général des L-Systems à crochets pour mieux représenter nos modèles : nous appellerons ce sous-ensemble Φ.

Dans ce but, j'ai repris les opérateurs du travail de Vanyi *et al.* (voir p. 40 de ce document) comme inspiration pour définir 3 *opérateurs géométriques* appelés **B**, **Y**, et **T**. Leur effet graphique en deux dimensions est montré à l'aide d'un arbre simple (Fig. 36) (l'extension à un espace 3D est simple et directe) ; syntaxiquement, cet arbre représente l'axiome ω d'une grammaire L-System.

Figure 36 Jeune arbre

L'opérateur de branchement B. Pour cet opérateur, une branche de longueur l pousse de la partie supérieure du jeune arbre en faisant un angle a avec la branche mère (Fig. 37(a)). Quelques exemples de la famille de formes générées par cet opérateur appliqué au jeune

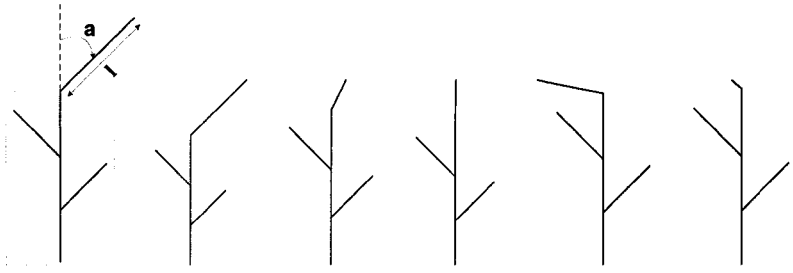


Figure 37 Exemples de figures générées par l'opérateur B

arbre de la Fig. 36 sont présentés à la Fig. 37(b-f) : en (b) $a > 0$ et l est (relativement) grand et positif ; en (c) $a > 0$ et l est (relativement) petit, en (d) $a = 0$, et l est petit, en (e) $a < 0$, et l est grand, et en (f) $a > 0$, et l est petit.

La famille de figures ainsi obtenue est écrite, en notation L-System, $\omega + (a)F(l)$; donc une règle de production L-System capturant ce type de croissance est

$$\mathbf{B}(a, l) \longrightarrow \omega + (a)F(l) \tag{3.1}$$

L'équivalent sensible au contexte de la règle (3.1) est

$$r_{i,j}^{(\mathbf{B})} : lc_i < \mathbf{B}(a, l) > rc_j \rightarrow \omega + (a)F(l) \tag{3.2}$$

(3.2) est un schéma définissant une famille de règles pour l'idée géométrique de B. Une instance de ce schéma se forme (crée une règle) en fixant un contexte gauche, lc_*, et un contexte droit rc_*

L'opérateur de croissance en *forme de Y* (Y). Pour cet opérateur, un tronc de longueur l « supporte » deux copies du jeune arbre ; ceux-ci sont inclinés d'angles a_1 et a_2, respectivement, de la verticale (Fig. 38(a)). Quelques exemples sont présentés aux Figs. 38(b-d) :

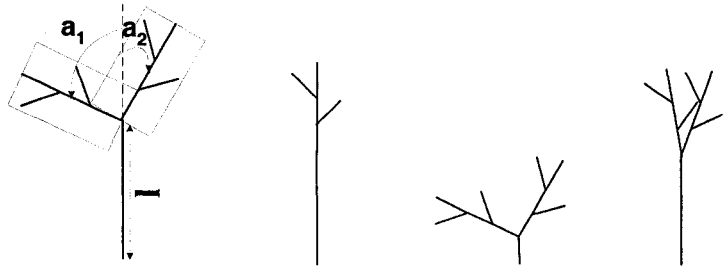


Figure 38 Exemples de figures générées par l'opérateur Y

en (b) l est grand et positif, $a_1 = a_2 = 0$; en (c) l est petit, a_1 est petit positif, et a_2 est grand négatif, et en (d) l est grand, a_1 est petit positif et a_2 est petit négatif.

Le schéma syntaxique pour Y est :

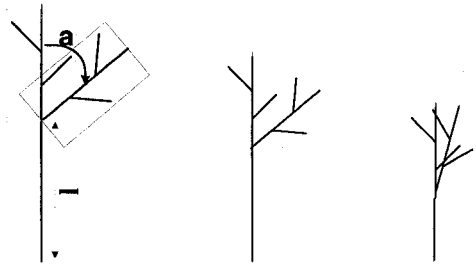
$$r_{i,j}^{(\mathbf{Y})} : lc_i < \mathbf{Y}(l, a_1, a_2) > rc_j \longrightarrow F(l)[+(a_1)\omega][-(a_2)\omega] \tag{3.3}$$

L'opérateur de croissance en *forme de T* (T). Pour cet opérateur, un tronc de longueur l « supporte » une copies du jeune arbre ; celui-ci est incliné d'un angle a par rapport à la verticale (Fig. 39(a)). Deux exemples sont présentés aux Figs. 39(b-c) : en (b) l et a sont grands et positifs ; en (c) l est petit et a est petit positif.

Le schéma syntaxique pour T est :

$$r_{i,j}^{(\mathbf{T})} : lc_i < \mathbf{T}(l, a) > rc_j \longrightarrow F(l)[+(a)\omega]\omega \tag{3.4}$$

Ceci nous permet d'apprécier le fait que, géométriquement, l'opérateur T est équivalent à l'opérateur Y ayant l'angle $c = 0$; la différence est cependant évidente du point de



(a) $\mathbf{T}$ (b) (c)

Figure 39 Exemples de figures générées par l'opérateur $\mathbf{T}$

vue syntaxique : tandis qu'avec l'opérateur $\mathbf{Y}$ la dernière partie crée une branche fermée contenant l'axiome $[-(c)\omega]$ (règle (3.3)), l'opérateur $\mathbf{T}$ se divise en une branche ouverte commençant avec l'axiome (règle (3.4)). Comme exemple, considérons les deux grammaires suivantes, $\mathbf{G_T}$ et $\mathbf{G_Y}$

$$\mathbf{G_T} :: \begin{cases} \omega = \mathbf{T}(1,45) - (30)F(1) \\ r : \mathbf{T}(l,a) \longrightarrow F(l)[+(a)\omega]\omega \end{cases} \qquad \mathbf{G_Y} :: \begin{cases} \omega = \mathbf{Y}(1,45,0) - (30)F(1) \\ r : \mathbf{Y}(l,a_1,a_2) \longrightarrow F(l)[+(a_1)\omega][-(a_2)\omega] \end{cases}$$

Tel que mentionné, les Figs. 38 et 39 nous laissent soupçonner que l'opérateur $\mathbf{T}$ est équivalent à l'opérateur $\mathbf{Y}$ avec $a_2 = 0$: $\mathbf{G_T}$ et $\mathbf{G_Y}$ devraient être, à toutes fins pratiques, équivalentes. Mais une (1) itération de chacune de ces grammaires produit des résultats différents, comme observés à la Fig. 40. À gauche, le résultat de l'itération de $\mathbf{G_Y}$ est

Figure 40 Différence entre $\mathbf{Y}$ et $\mathbf{T}$

montré, tandis que à droite se trouve l'itération de $\mathbf{G_T}$. Avec $\mathbf{G_T}$, la dernière partie de la

figure (trait qui va de (4) à (5)) est dessinée, tandis qu'avec $\mathbf{G_Y}$, ce trait se confond avec un autre déjà présent (trait qui va de (2) à (4)).

Les définitions (3.2), (3.3) et (3.4) définissent la forme syntaxique de Φ ; Φ représente, grammaticalement, l'ensemble complet de toutes les fractales avec forme d'arbre en 2D ; il est présenté en (3.5)

$$\Phi\ (2D) :: \begin{cases} \text{Alphabet, } \mathcal{T} = \{F, f, +, -, \wedge, \&, \backslash, /, (,), [,], \mathbf{B}, \mathbf{Y}, \mathbf{T}\} \cup \mathbb{R} \\ \text{axiome, } \omega \in \mathcal{T}^+ \\ r_{i,j}^{(\mathbf{B})} : \ lc_i < \mathbf{B}(a, l) > rc_j \longrightarrow \omega + (a)F(l) \\ r_{i,j}^{(\mathbf{Y})} : \ lc_i < \mathbf{Y}(l, a_1, a_2) > rc_j \longrightarrow F(l)[+(a_1)\omega][-(a_2)\omega] \\ r_{i,j}^{(\mathbf{T})} : \ lc_i < \mathbf{T}(l, a) > rc_j \longrightarrow F(l)[+(a)\omega]\omega \end{cases} \tag{3.5}$$

Géométriquement, l'ensemble de formes correspondant à une instance $\mathbf{G}$ de Φ est définie par la spécification de chaque contexte gauche lc_i et chaque contexte droit rc_j.

La même définition, mais en 3D, nécessite la spécification d'un autre paramètre contrôlant une rotation sur l'axe vertical (« Y ») (voir (3.6))

$$\Phi :: \begin{cases} \text{Alphabet, } \mathcal{T} = \{F, f, +, -, \wedge, \&, \backslash, /, (,), [,], \mathbf{B}, \mathbf{Y}, \mathbf{T}\} \cup \mathbb{R} \\ \text{axiome, } \omega \in \mathcal{T}^+ \\ r_{i,j}^{(\mathbf{B})} : \ lc_i < \mathbf{B}(a_1, l, a_2) > rc_j \longrightarrow \omega + (a_1)\&(a_2)F(l) \\ r_{i,j}^{(\mathbf{Y})} : \ lc_i < \mathbf{Y}(l, a_1, a_2, a_3) > rc_j \longrightarrow F(l)[+(a_1)\&(a_3)\omega][-(a_2)\&(a_3)\omega] \\ r_{i,j}^{(\mathbf{T})} : \ lc_i < \mathbf{T}(l, a_1, a_2) > rc_j \longrightarrow F(l)[+(a_1)\&(a_2)\omega]\omega \end{cases} \tag{3.6}$$

3.3 Apport d'une solution

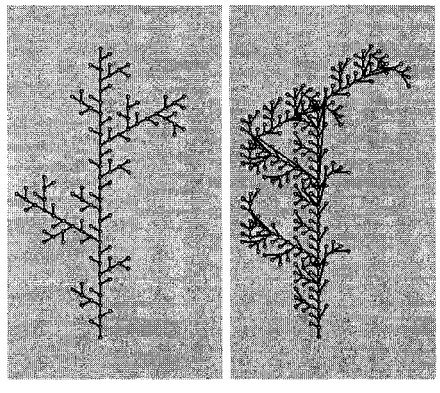
Le problème est maintenant réduit à synthétiser un L-System suivant le schéma (3.6) représentant un processus de développement particulier ; ce problème est appelé le *problème*

d'inférence ou le *problème inverse* pour la théorie des L-Systems (Prusinkiewicz et Hanan, 1989, p. 11).

Pourquoi et comment est-ce un problème difficile ? La synthèse de modèles à partir de représentations naturelles relève encore plus de l'art que d'une approche systématique plus propre à la science. En grande partie, ceci est dû à l'extrême compacité de la représentation (une grammaire L-System), car « des modifications aléatoires des productions [L-System] donnent peu de pistes sur la relation entre les grammaires L-Systems et les figures qu'elles génèrent » (Prusinkiewicz et Hanan, 1989, p. 11, traduction libre). Comme exemple, considérons les grammaires $\mathbf{G}_1$ et $\mathbf{G}_2$:

$$G_1 :: \begin{cases} \omega = F \\ r : F \longrightarrow F[+F]F[-F]F \end{cases} \qquad G_2 :: \begin{cases} \omega = F \\ r : F \longrightarrow F[+F]F[+F][-F][-F]F \end{cases}$$

$\mathbf{G}_1$ et $\mathbf{G}_2$ ont toutes les deux le même axiome (F) et une seule règle de production ; la seule différence se trouve dans celle-ci : tandis que celle de $\mathbf{G}_1$ est F[+F]F[-F]F, celle de $\mathbf{G}_2$ est F[+F]F**[+F]**[-F]**[-F]**F. Cependant, la troisième itération de ces deux grammaires produit des résultats sensiblement différents : l'interprétation de $\mathbf{G}_1$ est présentée à la Fig. 41(a), $\mathbf{G}_2$ à la Fig. 41(b).



(a) $\mathbf{G}_1$ (b) $\mathbf{G}_2$

Figure 41 Relation entre grammaires et figures générées

Cet exemple simple montre comment il est pratiquement impossible de faire une relation directe entre les changements syntaxiques aux règles et le résultat graphique obtenu.

En utilisant ces définitions, l'approximation d'un processus de développement (*résoudre le problème inverse*) est l'équivalent trouver une grammaire G dont le processus de développement sous Υ, $\mathcal{D}_{(\mathbf{G},\Upsilon)}$ (Υ un contexte spatio-temporel) approche $\mathcal{L}_f$ le mieux. Les parties principales de ce problème sont présentées à la Fig. 42.

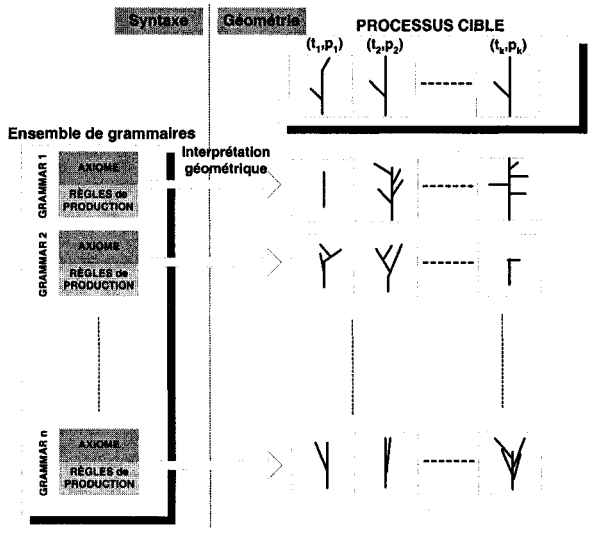


Figure 42 Spécification graphique du problème à résoudre

L'ensemble complet de grammaires Φ codifiant des arborescences est représenté à la gauche de la figure (partie *Syntaxe*). Chaque grammaire est graphiquement interprétée pour générer sa représentation géométrique ; ces représentations sont alors comparées avec l'ensemble cible. La grammaire ayant la représentation géométrique la plus proche de cet ensemble est considérée comme le meilleur modèle.

3.3.1 La méthode, intuitivement

Je commencerai par donner, dans cette partie, l'intuition d'une méthode appropriée ; pour commencer, il est très important de connaître exactement l'ensemble graphique généré par Φ (Φ, rappelons-le, est un ensemble d'objets *syntaxiques*) ; l'interprétation graphique de Φ ($\mathcal{G}_\Phi(\Upsilon)$) est l'ensemble de processus de développement des grammaires en Φ, sous un certain contexte Υ (cf. Déf. 3.2, page 46) ; $\mathcal{G}_\Phi(\Upsilon)$ correspond à la partie droite de la Fig. 42.

La première chose à remarquer par rapport à $\mathcal{G}_\Phi(\Upsilon)$ est qu'il est important que deux grammaires $\mathrm{G}_1, \mathrm{G}_2 \in \Phi, \mathrm{G}_1 \not\equiv \mathrm{G}_2$, produisent des représentations géométriques différentes ; autrement ces grammaires proposent la même solution, et seront considérées *équivalentes* (cette notion est formalisée dans les défs. 3.6 et 3.7). Un ensemble de grammaires $\in \mathcal{G}_\Phi(\Upsilon)$ générant les mêmes processus de développement forme une *classe d'équivalence* en $\mathcal{G}_\Phi(\Upsilon)$; ces classes d'équivalence définissent une partition $\mathcal{P}_\Phi(\Upsilon)$ de $\mathcal{G}_\Phi(\Upsilon)$. Donc l'ensemble « intéressant » de $\mathcal{G}_\Phi(\Upsilon)$ est $\mathcal{P}_\Phi(\Upsilon)$: celui où tous ses membres proposent des solutions différentes pour le problème à résoudre.

L'approche à suivre pour étudier la construction de $\mathcal{G}_\Phi(\Upsilon)$ se base sur la division de la structure de Φ en différents sous-ensembles, pour ainsi étudier une version réduite du grand ensemble ; postérieurement, une généralisation de ses propriété sera effectuée. La façon la plus directe de diviser Φ est par l'axiome de chacune de ses grammaires ; en effet, si Φ/ω_i est le sous-ensemble de Φ ayant axiome ω_i, nous pouvons écrire

$$\Phi = \{\Phi/\omega_1 \cup \Phi/\omega_2 \cup \ldots \cup \Phi/\omega_i \cup \ldots\}$$

Donc je commencerai par étudier ce problème comme si nous avions un (1) seul axiome ω_a, pour ensuite étendre nos conclusions à tous les axiomes : nous allons essayer d'obtenir le sous-ensemble intéressant de Φ en partant de ω_a, et ensuite établir que $\mathcal{P}_\Phi(\Upsilon)$ est l'union de tous les sous-ensembles générés. Donc, si nous appelons $\zeta(\omega_a)$ l'ensemble

des grammaires représentantes des classes d'équivalence générées à partir de l'axiome ω_a

$$\zeta(\omega_a) = \{\mathbf{G}_1^{(\omega_a)}, \mathbf{G}_2^{(\omega_a)}, \ldots \mathbf{G}_g^{(\omega_a)}\}$$

une façon « simple » de générer $\mathcal{G}_\Phi(\Upsilon)$ est la suivante :

a. définir l'ensemble de *tous* les axiomes possibles $S_\omega = \{\omega_1, \omega_2, \ldots \omega_a, \ldots\}$,

b. générer $\zeta(\omega_i)$ pour chaque axiome $\omega_i \in S_\omega$, et

c. faire l'union de tous les ensembles :

$$\mathcal{P}_\Phi(\Upsilon) = \bigcup_{i=1}^{\infty} \zeta(\omega_i)$$

La section 3.3.2 construit la méthode avec un axiome en particulier ; la section 3.3.3 fait l'extension pour tous les axiomes.

3.3.2 Sous-ensemble de Φ partageant un axiome ω_a particulier

La forme syntaxique de chaque grammaire candidate $\mathbf{G}_c$ en Φ est connue, grâce à deux points fondamentaux :

a. l'axiome de $\mathbf{G}_c$ est fixe $= \omega_a$

b. le schéma (3.6) définit les règles dans $\mathbf{G}_c$

Donc la forme générale de chaque $\mathbf{G}_c$ est

$$\Phi :: \begin{cases} \omega = \omega_a \\ r_{i,j}^{(\mathbf{B})} : \quad lc_i < \mathbf{B}(a_1, l, a_2) \quad > rc_j \longrightarrow \omega + (a_1)\&(a_2)F(l) \\ r_{i,j}^{(\mathbf{Y})} : \quad lc_i < \mathbf{Y}(l, a_1, a_2, a_3) \quad > rc_j \longrightarrow F(l)[+(a_1)\&(a_3)\omega][-(a_2)\&(a_3)\omega] \\ r_{i,j}^{(\mathbf{T})} : \quad lc_i < \mathbf{T}(l, a_1, a_2) \quad > rc_j \longrightarrow F(l)[+(a_1)\&(a_2)\omega]\omega \end{cases}$$

Du point de vue graphique, l'interprétation de chaque $\mathbf{G}_c$ est guidée par la définition de $\Upsilon = \{\Upsilon_1, \ldots, \Upsilon_k\} = \{(t_1, p_1), \ldots, (t_k, p_k)\}$

Avec cette information, tout ce qu'il y a à « choisir » ce sont les contextes (lc_i, rc_j) apparaissant dans chacune de ces règles. Ceci induit une idée pour la construction de chacune de ces candidates :

a. Construire l' ensemble R de toutes les règles $\{r_1, \ldots, r_m\}$ suivant schéma (3.6) qui peuvent s'appliquer (donc qui *modifient*) à l'axiome ω_a. Ce pas choisit effectivement les contextes gauche et droit des règles appropriées.

b. Construire tous les sous-ensembles de règles de R. Chaque sous-ensemble de taille $t = \{1, 2, \ldots, m\}$ de $R(\mathcal{S}_t, \mathcal{S}_t \subset R)$, appliqué sur ω_a, produit un LString différent $= \{ls_1, ls_2, \ldots, ls_n\}$ (i.e., une figure différente).

c. Recommencer ces trois pas avec chaque LString ls_i.

Les détails de chacun de ces pas sont présentés ci-après :

Construire les règles. Construire toutes les règles $\{r_1, \ldots, r_m\}$ suivant le schéma (3.6) qui peuvent s'appliquer à l'axiome ω_a. Cette construction se fait par trois simples pas :

1 Identifier les symboles dérivables dans l'axiome

2 Identifier les contextes gauche et droit de chacun de ces symboles

3 Construire des règles selon le schéma (3.6) avec ces contextes

[Exemple]

Soit ω_a

$\omega_a = F + (30)F\mathbf{B}(15, 1, 30) - (45)F\mathbf{Y}(2, -15, 30, 225)$

Les symboles dérivables en ω_a sont $\mathbf{B}$ et $\mathbf{Y}$, indiqués ici :

$\omega_a = F + (30)F\overline{\mathbf{B}(15, 1, 30)} - (45)F\overline{\mathbf{Y}(2, -15, 30, 225)}$

(*suite...* ⟶)

(*suite de la page antérieure*)

La règle correspondant à chaque symbole dérivable trouvé est construite avec leurs contextes :

$$r_1 : \overbrace{F + (30)F}^{\text{contexte gauche}} < \mathbf{B}(a_1, l, a_2) > \overbrace{-(45)F\mathbf{Y}(2, -15, 30, 225)}^{\text{contexte droit}} \cdots$$
$$\longrightarrow \omega + (a_1)\&(a_2)F(l)$$

et

$$r_2 : \overbrace{F + (30)F\mathbf{B}(15, 1, 30) - (45)F}^{\text{contexte gauche}} < \mathbf{Y}(l, a_1, a_2, a_3) \cdots$$
$$\longrightarrow F(l)[+(a_1)\&(a_3)\omega][-(a_2)\&(a_3)\omega]$$

Alors S, l'ensemble de règles qui peuvent déclencher une dérivation en ω_a, est $S = \{r_1, r_2\}$

Combiner les règles. Chaque sous-ensemble de taille $t = \{1, 2, \ldots, m\}$ des règles de $\{r_1, \ldots, r_m\}$ produit un LString différent $= \{ls_1, ls_2, \ldots, ls_n\}$ (i.e., une figure différente). Donc chacune d'elles forme une partie d'une grammaire différente.

[Suite de l'exemple]

Comme nous avons deux règles $\{r_1, r_2\}$ à combiner, nous obtenons 4 sous-ensembles valides :

$\mathcal{S}^{(1)} = \{r_1\}, \quad \mathcal{S}^{(2)} = \{r_2\}, \quad \mathcal{S}^{(3)} = \{r_1, r_2\}, \quad \mathcal{S}^{(4)} = \emptyset$

et donc la dérivation $\omega_a \xrightarrow{\mathcal{S}^{(i)}} s_i$ produit 4 différentes instances de s_i

$s_1 : F + (30)F\omega + (15)\&(30)F(1) - (45)F\mathbf{Y}(2, -15, 30, 225)$

$s_2 : F + (30)F\mathbf{B}(15, 1, 30) - (45)FF(2)[-(15)\&(225)\omega][-(30)\&(225)\omega]$

$s_3 : F + (30 - F\omega + (15)\&(30)F(1) - (45)F \ldots$
$\ldots F(2)[(-15)\&(225)\omega][-(30)\&(225)\omega]$

$s_4 : F + (30)F\mathbf{B}(15, 1, 30) - (45)F\mathbf{Y}(2, -15, 30, 225)$

(*suite...* $\longrightarrow$)

(*suite de la page antérieure*)

Ces quatre LString ont des interprétations graphiques différentes, et donc les grammaires les générant doivent être considérées comme des (parties de) solutions candidates différentes.

$$G_1 :: \begin{cases} \omega = \omega_a \\ r_1 \end{cases} \quad G_2 :: \begin{cases} \omega = \omega_a \\ r_2 \end{cases} \quad G_3 :: \begin{cases} \omega = \omega_a \\ r_1, r_2 \end{cases} \quad G_4 :: \begin{cases} \omega = \omega_a \\ \emptyset \end{cases}$$

Passer à la prochaine itération. Il s'agit ensuite de prendre chaque LString généré ls_i comme « point de départ » pour la construction d'un nouvel ensemble de LString-grammaires.

Cette procédure continue jusqu'à l'itération $t = t_k$; visuellement, cette idée construit un *arbre de dérivation*, où chaque nœud est un des LString générés. Un exemple construit pour l'itération $t = 2$ est montré à la Fig. 43. La construction de l'ensemble des gram-

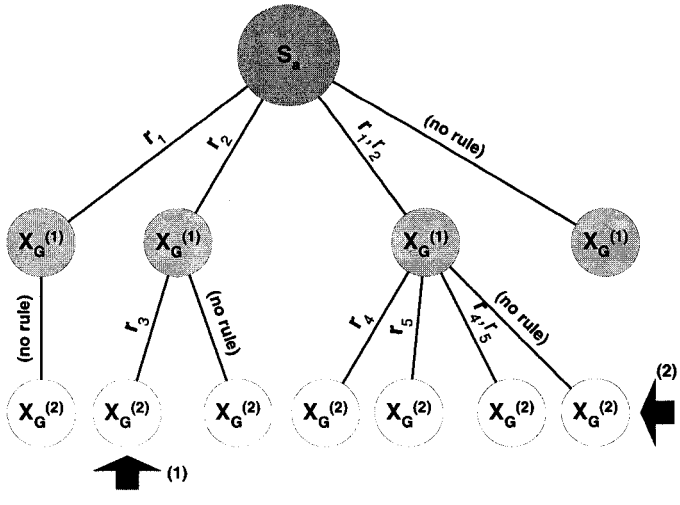


Figure 43 Un arbre de dérivations

maires est visuellement très simple avec l'image de l'arbre : pour le construire, il faut simplement prendre une feuille de l'arbre, et suivre le chemin de l'axiome ω_a jusqu'à cette feuille ; les règles visitées dans ce chemin définissent la grammaire. Par exemple, les

grammaires identifiées comme (1) et (2) dans la Fig. 43 sont :

$$(1) :: \begin{cases} \text{Axiome} : \omega = \omega_a \\ \text{Règles :} \\ r_2, r_3 \end{cases} \qquad (2) :: \begin{cases} \text{Axiome} : \omega = \omega_a \\ \text{Règles :} \\ r_1, r_2 \end{cases}$$

Cette méthode, menée jusqu'à l'itération $t = t_k$, construit toutes les grammaires en $\zeta(\omega_a)$; elle assure que **toutes les grammaires en $\zeta(\omega_a)$ sont différentes.** En effet, en utilisant l'arbre de dérivations et la façon de construire les grammaires, il est simple de voir que les règles appartenant à une grammaire, à une certaine profondeur, n'apparaissent pas dans les autres grammaires de la même profondeur ; donc toutes les grammaires sont différentes, par construction (le concept de similarité entre grammaires est défini à la Déf. 3.4).

Définition 3.4 (Similarité entre deux grammaires L-System). *Soient* $\mathbf{G}_1$ *et* $\mathbf{G}_2$ *deux grammaires :*

$$\mathbf{G}_1 :: \begin{cases} \textit{un axiome } \omega_1 \\ \textit{règles de production } R_1 \end{cases} \qquad \mathbf{G}_2 :: \begin{cases} \textit{un axiome } \omega_2 \\ \textit{règles de production } R_2 \end{cases}$$

$\mathbf{G}_1$ *est égale à* $\mathbf{G}_2$ *si (et seulement si) :*

a. $\mathbf{G}_1$ *et* $\mathbf{G}_2$ *ont le même axiome* $(\omega_1 = \omega_2)$

b. Il y a autant de règles en R_1 *qu'en* R_2

c. Toutes les règles en $\mathbf{G}_1$ *sont en* $\mathbf{G}_2$ *:*

$$\forall r_i \in R_1 \exists r_j \in R_2 /$$

$$\textit{prédécesseur strict}(r_i) = \textit{prédécesseur strict}(r_j)$$

$$\wedge \quad \textit{successeur}(r_i) = \textit{successeur}(r_j)$$

$$\wedge \quad \textit{le contexte gauche de } (r_i) = \textit{le contexte gauche de } (r_j)$$

$$\wedge \quad \textit{le contexte droit de } (r_i) = \textit{le contexte droit de } (r_j)$$

De plus, cette construction garantit que les grammaires dans l'ensemble $\zeta(\omega_a)$ sont *minimales*, dans deux sens différents :

a. **Il n'y a pas de règles non-utilisées dans les grammaires en $\zeta(\omega_a)$.** La dérivation de longueur k d'une grammaire $\mathbf{G}_i^{(\omega_a)} \in \zeta(\omega_a)$ utilise toutes les règles de $\mathbf{G}_i^{(\omega_a)}$; en d'autres mots, il n'y a pas de *règles muettes* dans $\mathbf{G}_i^{(\omega_a)}$ (une règle *muette* ou *non-exprimée* est une règle dont le prédécesseur n'est pas présent dans l'axiome ou dans aucune des dérivations de la grammaire - voir Déf. 3.5). Donc elle est minimale dans le sens qu'elle est équivalente à d'autres grammaires qui incorporent des règles muettes dans leur définition.

Définition 3.5 (Règle de production r_i non-exprimée (ou muette) en $\mathbf{G}$ en n pas). *Soient $n \in \mathbb{N}$ et $\mathbf{G}$ une grammaire :*

$$\mathbf{G} :: \begin{cases} \omega = \omega_{\mathbf{G}} \\ r_1 : \textit{def. de la règle 1} \\ \vdots \\ r_m : \textit{def. de la règle } m \end{cases}$$

Soit d la dérivation de longueur n de $\mathbf{G}$

$$d : \omega_{\mathbf{G}} \underbrace{\xrightarrow{r'_1} \ldots \xrightarrow{r'_j} \ldots \xrightarrow{r'_n}}_{n \textit{ pas}} \delta$$

r_i est non-exprimée (ou « muette ») en $\mathbf{G}$ si (et seulement si) $\forall j, 1 \leq j \leq n, r'_j \neq r_i$

[Exemple]

Soit $\mathbf{G}_1^{(\omega_a)}$ une grammaire avec axiome ω_a

$$\mathbf{G}_1^{(\omega_a)} :: \begin{cases} \omega = \omega_a = \mathbf{B}(30, 2, -45)F + f \\ r_1 : \mathbf{Y}(l, a_1, a_2) \longrightarrow F(l)[+(a_1)\omega][-(a_2)\omega] \end{cases}$$

(*suite...* $\longrightarrow$)

(*suite de la page antérieure*)

Le prédécesseur dans la seule règle de $\mathbf{G}_1^{(\omega_a)}$ (r_1, avec prédécesseur $\mathbf{Y}$) ne se retrouve pas entre les symboles présents dans l'axiome (qui sont $\mathbf{B}$, F et f) : r_1 est alors une *règle non-exprimée* ; dans la pratique, cela veut dire qu'une dérivation de longueur n de $\mathbf{G}_1^{(\omega_a)}$ génère toujours l'axiome, ω_a, car aucun remplacement n'est effectué.

Regardons maintenant $\mathbf{G}_0^{(\omega_a)}$

$$\mathbf{G}_0^{(\omega_a)} :: \begin{cases} \omega = \mathbf{B}(30, 2, -45)F + f \\ \textit{pas de règles} \end{cases}$$

Comme $r_1 \in \mathbf{G}_1^{(\omega_a)}$ n'est pas exprimée, les dérivations de $\mathbf{G}_1^{(\omega_a)}$ sont les mêmes que pour $\mathbf{G}_0^{(\omega_a)}$: toujours l'axiome. Comme les dérivations sont les mêmes, les interprétations géométriques sont les mêmes, et donc $\mathbf{G}_0^{(\omega_a)}$ exprime la même chose que $\mathbf{G}_1^{(\omega_a)}$, tout en ayant moins de règles.

b. **Les règles dans chaque grammaire de $\zeta(\omega_a)$ sont le plus spécifique possible.** Le choix des contextes (page 65) garantit que chaque règle exprime le plus spécifiquement possible l'état du LString.

[Exemple]

Pour le LString S : $S = +(30) - F\mathbf{B}(-15, 1, 30) - (15)F$

Le seul symbole dérivable est $\mathbf{B}$:

$+(30) - F\overline{\overline{\mathbf{B}(-15, 1, 30)}} - (15)F$

et donc la règle construite par le pas décrit (page 65) est :

$$r_1 : \overbrace{+(30) - F}^{\text{contexte gauche}} < \mathbf{B}(a_1, l, a_2) > \overbrace{-(15)F}^{\text{contexte droit}} \longrightarrow \omega + (a_1)\&(a_2)F(l)$$

Ce ne sont clairement pas les seuls contextes possibles. La liste des contextes possibles gauches pour B(-15,1,30) est : F, $-F$, $+(30)-F$; les contextes droits seraient $-(15)$ et $(-15)F$; le choix de chacun de ces contextes produirait les différentes règles suivantes :

(*suite...* $\longrightarrow$)

(*suite de la page antérieure*)

$$
\begin{array}{lrlll}
r_1: & F & < \mathbf{B}(a_1,l,a_2) > & -(15) & \longrightarrow \omega + (a_1)\&(a_2)F(l) \\
r_2: & F & < \mathbf{B}(a_1,l,a_2) > & -(15)F & \longrightarrow \omega + (a_1)\&(a_2)F(l) \\
r_3: & -F & < \mathbf{B}(a_1,l,a_2) > & -(15) & \longrightarrow \omega + (a_1)\&(a_2)F(l) \\
r_4: & -F & < \mathbf{B}(a_1,l,a_2) > & -(15)F & \longrightarrow \omega + (a_1)\&(a_2)F(l) \\
r_5: & +(30) - F & < \mathbf{B}(a_1,l,a_2) > & F & \longrightarrow \omega + (a_1)\&(a_2)F(l) \\
r_6: & +(30) - F & < \mathbf{B}(a_1,l,a_2) > & -(15)F & \longrightarrow \omega + (a_1)\&(a_2)F(l)
\end{array}
$$

Toutes ces règles ont le même effet sur S :

$S \longrightarrow +(30) - F\omega + (-15)\&(30)F(1) - (15)F$

Cependant, r_6 est la règle dont les contextes représentent mieux l'état de S, et est donc la plus spécifique au LString.

Donc chaque grammaire en $\zeta(\omega_a)$ représente le plus étroitement possible l'information présente.

Ces deux caractéristiques font en sorte que les grammaires en $\zeta(\omega_a)$ sont les représentantes d'un ensemble de classes d'équivalence formant une partition de Φ ; ces classes d'équivalence représentent chacune des solutions intéressantes pour le problème à résoudre (car nous avons utilisé l'information présente en Υ pour les construire). $\zeta(\omega_a)$ **est l'ensemble minimal de grammaires générant des processus de développement, différents entre eux, apportant une approximation au processus de développement cible.**

L'écriture formelle de ces concepts comporte les définitions suivantes :

Définition 3.6 (Υ-Équivalence). *Deux grammaires* $\mathbf{G}_1, \mathbf{G}_2 \in \mathcal{G}_\Phi(\Upsilon)$ *qui génèrent un même processus de développement* ($\mathcal{D}_{(\mathbf{G}_1,\Upsilon)} = \mathcal{D}_{(\mathbf{G}_2,\Upsilon)}$) *pour le contexte* Υ *sont* Υ*-équivalentes*

Définition 3.7 (∞-Équivalence). *Deux grammaires* $\mathbf{G}_1, \mathbf{G}_2 \in \mathcal{G}_\Phi(\Upsilon)$ *sont* ∞-équivalentes *si (et seulement si) ils sont* Υ*-équivalentes pour tout* Υ.

Définition 3.8 (Classe d'équivalence en $\mathcal{G}_\Phi(\Upsilon)$). *$\mathcal{C} = \{\mathbf{G}_1, \ldots, \mathbf{G}_n\}$ est une classe d'équivalence en $\mathcal{G}_\Phi(\Upsilon)$ si (et seulement si) :*

a. $\forall \mathbf{G}_i \in \mathcal{C} \Rightarrow \mathbf{G}_i \in \Phi$

b. Toute paire de grammaires $\mathbf{G}_i, \mathbf{G}_j \in \mathcal{C}$ sont Υ-équivalentes.

Définition 3.9 (Grammaire représentant une classe d'équivalence). *Soit* G *une grammaire appartenant à une classe d'équivalence $\mathcal{C}$;* G *est le* représentant *de $\mathcal{C}$ si (et seulement si)*

a. Il n'y a pas de règles muettes en G.

b. Les règles en G *sont les* **plus spécifiques** *parmi les grammaires en $\mathcal{C}$:*

$$\forall r_i \in R \not\exists \overline{G} \in \mathcal{C}, r_j \in \overline{G} /$$

$$\textit{prédécesseur stricte}(r_i) = \textit{prédécesseur stricte}(r_j)$$

$$\wedge \ \textit{successeur}(r_i) = \textit{successeur}(r_j)$$

$$\wedge \ \textit{le contexte gauche de } (r_i) \textit{ est moins spécifique que le contexte gauche de } (r_j)$$

$$\wedge \ \textit{le contexte droit de } (r_i) \textit{ est moins spécifique que le contexte droit de } (r_j)$$

(Voir 1.3.3, page 27 pour la définition de prédécesseur stricte, successeur, *et* contexte *; le concept de* spécificité *est établi à la Def. 1.5, page 29.)*

3.3.2.1 L'algorithme exhaustif

L'algorithme 1 exprime formellement la méthode de construction de $\zeta(\omega_a)$; il procède en considérant d'abord l'axiome ω_a, et choisissant toutes les règles possibles qui peuvent s'appliquer à celui-ci (une règle r « *peut être appliquée* » à un certain LString s si une itération de s par r produit un LString $l \neq s$) ; ceci initie une construction parallèle de toutes les dérivations d_*.

Algorithme 1 Générer toutes les grammaires jusqu'à un niveau k

1: **procedure** GENTOUTES(ω_a : axiome ; k : itération maximum en Υ)

SORTIES : TG, l'ensemble de toutes les grammaires

2: $S_a \leftarrow \{\omega_a\}, \mathbf{G}_a \leftarrow \{(\omega_a, \emptyset)\}$

3: **pour** $i = 1$ à k **faire**

4: $S_n \leftarrow \emptyset, \mathbf{G}_n \leftarrow \emptyset$

5: **pour tout** $\mu_i \in S_a$ **faire**

6: $S_\mu \leftarrow$ construireS($\mu_i, \mathbf{G}_i$)

7: $(TS_\mu, TG_\mu) \leftarrow$ genGEtSPrésents($\mu_i, S_\mu, \mathbf{G}_i$)

8: $S_n \leftarrow S_n \cup TS_\mu, \mathbf{G}_n \leftarrow \mathbf{G}_n \cup TG_\mu$

9: **fin pour**

10: $S_a \leftarrow S_n, \mathbf{G}_a \leftarrow \mathbf{G}_n$

11: **fin pour**

12: Retourner TG_μ

13: **fin procedure**

Dans la ligne 1.6, S_μ contient toutes les règles possibles qui peuvent être appliquées à μ_i ; le calcul de S_μ est effectué par l'algorithme 2. À la ligne 1.7, TS_μ est l'ensemble de toutes les chaînes possibles obtenues à partir de μ_i ; TG_μ sont les grammaires qui ont généré TS_μ. Ces deux ensembles sont produits par l'algorithme 3. Le calcul de toutes les règles possibles pouvant être appliquées à un certain LString μ, fait par l'algorithme 2, procède en cherchant (a) tous les symboles dérivables (**B**, **T**, et **Y**) en μ (ligne 2.2) et (b) pour chacun de ces symboles, construire la règle appropriée (suivant la structure de Φ, (3.6), page 60) : premièrement le prédécesseur et le successeur (lignes 2.5 à 2.11), et ensuite le contexte (ligne 2.12).

L'algorithme 3 calcule les chaînes possibles obtenues en commençant par un LString μ.

Algorithme 2 Construire $\mathcal{S}$

procedure CONSTRUIRES(μ : un LString ; $\mathbf{G}_\mu$: une grammaire L-System)
SORTIES : S_μ, l'ensemble de toutes les règles pouvant être appliquées à μ
  $\mathcal{D}$ = ensemble de tous les symboles dérivables en μ, avec leur contextes droits et gauches l_i et r_i. $\mathcal{D} = \{(d_1, l_1, r_1), (d_2, l_2, r_2), \ldots, (d_n, l_n, r_n)\}$
  $S_\mu \leftarrow \emptyset$
  pour tout $(d_1, l_1, r_1) \in \mathcal{A}$ **faire**
    si $d_i = \mathbf{B}$ **alors** ▷ définit le prédécesseur et le successeur de la règle r
      prédécesseur(r) = $\mathbf{B}(a_1, l, a_2)$, successeur(r) = $\omega + (a_1)\&(a_2)F(l)$
    sinon si $d_i = \mathbf{Y}$ **alors**
      prédécesseur(r) = $\mathbf{Y}(l, a_1, a_2, a_3)$, successeur($r$) = $F(l)[+(a_1)\&(a_3)\omega][-(a_2)\&(a_3)\omega]$
    sinon
      prédécesseur(r) = $\mathbf{T}(l, a_1, a_2)$, successeur(r) = $F(l)[+(a_1)\&(a_2)\omega]\omega$
    fin si
    contexte gauche(r) = l_i, contexte droit(r) = r_i
    $S_\mu \leftarrow S_\mu \cup \{r\}$
  fin pour
  Retourner S_μ
fin procedure

Algorithme 3 Générer grammaires et chaînes

procedure GENGETSPRÉSENTS(μ : un LString ; S_μ : ensemble de n_μ règles pouvant être appliquées à μ ; $\mathbf{G}_\mu$: une grammaire L-System)
SORTIES : TS_μ, l'ensemble de toutes les chaînes générées par la combinaison des règles en S_μ ; TG_μ, l'ensemble des grammaires « responsables » pour TS_μ
  Soit τ_μ l'ensemble des parties de S_μ, $\tau_\mu = \{S_\mu^{(1)}, S_\mu^{(2)}, \ldots, S_\mu^{(n_\mu)}\}$
  $TS_\mu \leftarrow \emptyset$, $TG_\mu \leftarrow \emptyset$
  pour tout $S_\mu^{(i)} \in \tau_\mu$ **faire**
    s_i = résultat de la dérivation $\mu \xrightarrow{S_\mu^{(i)}} s_i$
    $TS_\mu \leftarrow TS_\mu \cup \{s_i\}$
    $TG_\mu \leftarrow TG_\mu \cup (\mathbf{G}_\mu \cup S_\mu^{(i)})$
  fin pour
  Retourner TS_μ, TG_μ
fin procedure

3.3.2.2 L'algorithme optimisé

Nous avons présenté la première approximation de l'algorithme faisant la synthèse d'approximations de processus de développement, en partant d'un axiome ω_a : l'algorithme 1

(« exhaustif ») génère un arbre de dérivations de profondeur k (comme celui de la Fig. 44), où k est l'étiquette temporelle la plus grande trouvée en Υ. Les modèles (grammaires) proposé(e)s comme candidates sont construites en suivant les différents chemins des feuilles de cet arbre jusqu'à la racine (qui est ω_a).

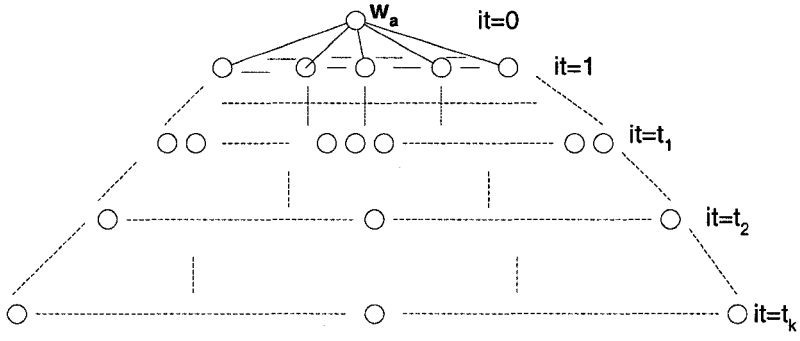


Figure 44 Un arbre complet de dérivation

En proposant cette solution, cependant, nous n'avons pas utilisé *toute* l'information contenue dans le problème ; en particulier, nous ignorons le fait que les solutions candidates doivent être évaluées aux profondeurs précises $t = t_1, \ldots, t_k$. Ceci peut servir pour simplifier l'arbre de dérivations, en utilisant l'intuition disant que la solution qui approche le mieux un processus de développement donné est aussi celle qui approche le mieux *chacune de ses parties séparées.*

Si cette affirmation s'avère vraie, chaque évaluation faite à un point temporel indiqué par Υ permet de prendre la solution la mieux adaptée jusqu'à ce point, et de continuer dans l'évaluation uniquement avec celle-ci ; de cette façon tout le reste des solutions offrant des performances (partielles) moins bonnes ne sont plus évaluées.

Évaluation des nœuds

À ce point il est important de faire une pause dans la description de la méthode pour considérer la façon dont l'évaluation des solutions sera faite ; l'évaluation d'un nœud avec

une partie du processus cible revient à comparer des branchements (qui représentent naturellement l'architecture de la structure) et à appliquer une fonction qui permette cette comparaison comme le ferait un être humain ; ceci est un problème délicat.

Pour commencer, nous pouvons définir quelles sont les *propriétés* qu'une fonction d'évaluation $\mathcal{D}$ adéquate doit avoir :

a. $\mathcal{D}$ doit être positive sur tout son domaine.

b. $\mathcal{D}$ doit diminuer quand les solutions, jugées par un expert humain, s'améliorent. Ceci implique que $\mathcal{D}$ doit valoir 0 quand la « *solution parfaite* » est évaluée.

Ici, je présente trois pistes pour arriver à cette fin qui ont été considérées de près dans cette thèse :

a. Une fonction de comparaison a été générée à l'aide d'un réseau de neurones.

b. Une fonction de distance entre les points des structures a été définie

c. Un utilisateur humain (moi, en l'occurrence) a donné directement les valeurs des comparaisons

Ces trois propositions pour $\mathcal{D}$ sont décrites ci-après ; chacune d'entre elles définit une distance entre deux figures, appelées $\mathcal{F}_T$ (la « *figure cible* ») et $\mathcal{F}_C$ (la « *figure candidate* »).

a. **Approche par réseau de neurones.** Nous avons examiné la pertinence d'utiliser un réseau de neurones pour apprendre des perceptions humaines sur la différence entre deux architectures de plantes. Un ensemble de caractéristiques est définie pour une figure de plante et les données pour les expériences sont obtenues à l'aide d'un ensemble d'usagers à travers un système en ligne. Les résultats initiaux et les analyses correspondantes sont présentées ; les détails sont en Annexe III.

   Étant intéressé à générer une métrique mimant le système visuel humain, l'idée est de lier des résultats obtenus des votes d'usagers par rapport à la ressemblance de

paires de figures avec les caractéristiques associées avec ces paires. Pour atteindre ce but, j'ai construit un environnement de travail formé de 3 composantes :

(a) **Une façon de décrire une image**

(b) **Un système de vote *en ligne*** pour permettre aux usagers de donner leur opinion sur *la distance entre deux figures*

(c) **Un environnement de synthèse d'une fonction basé sur les réseaux de neurones**

Une liste de mesures décrivant une figure générique ont été choisies pour décrire un squelette d'un arbre :

(a) **Disposition des branches** : histogramme des longueurs des segments formant les branches et histogramme des angles entre elles.

(b) **Dimension fractale** : la dimension fractale d'une forme est la portion de l'espace Euclidéen qui est *remplie* par celle-ci[1].

(c) **L'aire** : le nombre de pixels dans la figure.

(d) **Le centroïde** : les coordonnées (x, y) du centre de masse de la région.

(e) **La boîte enveloppante** : le plus petit rectangle qui contient la figure.

(f) **Longueur de l'axe majeur** : la longueur (en pixels) de l'axe majeur de l'ellipse qui a le même moment d'ordre 2 que la région.

(g) **Longueur de l'axe mineur** : la longueur (en pixels) de l'axe mineur de l'ellipse qui a le même moment d'ordre 2 que la région.

[1] Plus de détails à la section 1.1, page 7.

(h) **Excentricité** : l'excentricité de l'ellipse qui a le même moment d'ordre 2 que la région. L'excentricité est le ratio de la distance entre le foyer de l'ellipse et son axe majeur.

(i) **Orientation** : l'angle (en degrés) entre l'axe des x et l'axe majeur de l'ellipse qui a le même moment d'ordre 2 que la région.

(j) **Aire remplie** : dans la région définie par l'image, tous les « trous » sont remplis ; le nombre de pixels dans cette nouvelle image (« *remplie* ») sont comptés.

(k) **L'aire convexe** : nombre de pixels dans le plus petit polygone convexe (l'*enveloppe convexe*, *convex hull* en anglais) qui contient la région.

(l) **Diamètre équivalent** : le diamètre d'un cercle avec la même aire que la région. Calculé avec $\sqrt{(4 * Area)/(\pi)}$.

(m) **Solidité** : la proportion de pixels dans la coque convexe qui sont aussi dans la région.

(n) **Amplitude** : la proportion de pixels dans la boîte enveloppante qui sont aussi dans la région.

Un site web a été désigné pour montrer des paires d'images (prises aléatoirement d'un ensemble S, présenté à la section AIII.5 de l'Annexe III), permettant ainsi aux usagers de les comparer. La page principale du site est http://alpamayo.livia.etsmtl.ca:8080/welcomeMetricVote.php[2]. Il y est demandé d'évaluer la *distance* entre chaque paire d'images présentée ; chaque *vote* est un chiffre entier dans l'intervalle $[0, 10]$, où 0 veut dire *ces 2 images sont très différentes* et 10 veut dire *ces 2 images sont pratiquement les mêmes*.

[2] Cette adresse peut ne plus être valide, car l'auteur de cette thèse ne travaille plus au LIVIA. Plus d'information devrait être disponible à http://www.livia.etsmtl.ca/people/costa

Une base de données d'images fut construite pour expérimenter avec cette idée. Sa description et quelques considérations pratiques sont décrites en section AIII.2.3 de l'Annexe III (page 152) ; un réseau de neurones d'architecture de perceptron à une (1) couche cachée a été entraîné avec l'algorithme de rétropropagation (McClelland et Rumelhart, 1986) ; l'entrée du réseau est constituée par les 6 meilleurs explicateurs suggérés par une analyse en composantes principales effectuée sur les caractéristiques d'entrée, et la sortie est un vote ($\in [0, 10]$). La couche cachée a 12 neurones (tous les détails se trouvent à l'Annexe III). L'erreur de généralisation d'un tel réseau a été mesurée comme étant $0,662 \pm 0,036$ (voir page 158) ; ceci indique que ce réseau réussit à approcher les valeurs de vote humain à une distance nettement inférieure à 1 : je trouve ces résultats extrêmement prometteurs.

b. **Approche point-par-point** Dans cette approche, l'adéquation est mesurée avec un indice de *dissimilarité point-à-point* entre deux figures.

Définissons $\mathcal{F}_T$ et $\mathcal{F}_C$ par sa liste de points dans l'espace :

$$\begin{aligned} \mathcal{F}_T &= \{p_1^{(T)}, p_2^{(T)}, \ldots, p_k^{(T)}\} \text{ ,} k \text{ points} \\ \mathcal{F}_C &= \{p_1^{(C)}, p_2^{(C)}, \ldots, p_r^{(C)}\} \text{ ,} r \text{ points} \end{aligned}$$

En général, $k \neq r$.

La distance entre $\mathcal{F}_T$ et $\mathcal{F}_C$ est calculée en deux étapes :

(a) Pour chaque point $p_i^{(T)} \in \mathcal{F}_T$, trouver le point en $\mathcal{F}_C$, $p_j^{(C)}$, où la distance (Euclidienne) entre $p_i^{(T)}$ and $p_j^{(C)}$, $\mathbb{E}(p_i^{(T)}, p_j^{(C)})$, est minimale. La somme de toutes ces distances pour tous les points en $\mathcal{F}_T$ est appelée $\mathcal{D}_f(\mathcal{F}_T, \mathcal{F}_C)^{(1)}$.

$$\mathcal{D}_f(\mathcal{F}_T, \mathcal{F}_C)^{(1)} = \sum_{i=1}^{k} \mathbb{E}(p_i^{(T)}, p_{j_i}^{(C)}) \tag{3.7}$$

(b) Cette étape travaille avec les points en $\mathcal{F}_C$ qui n'ont pas « été choisis » dans l'étape antérieure ; pour chacun de ces points $p_{r_i}^{(C)}$, trouver $p_{p_i}^{(T)} \in \mathcal{F}_T$ qui minimise la distance Euclidéenne $\mathbb{E}(p_{r_i}^{(C)}, p_{p_i}^{(T)})$. La somme pour tous les points définit $\mathcal{D}_f(\mathcal{F}_T, \mathcal{F}_C)^{(2)}$ (Eq. (3.8))

$$\mathcal{D}_f(\mathcal{F}_T, \mathcal{F}_C)^{(2)} = \sum_{i=1}^{r_n} \mathbb{E}(p_i^{(C)}, p_{j_i}^{(T)}) \tag{3.8}$$

Finalement, la distance entre $\mathcal{F}_T$ et $\mathcal{F}_C$ est la somme des deux expressions :

$$\mathcal{D}_f(\mathcal{F}_T, \mathcal{F}_C) = \mathcal{D}_f(\mathcal{F}_T, \mathcal{F}_C)^{(1)} + \mathcal{D}_f(\mathcal{F}_T, \mathcal{F}_C)^{(2)} \tag{3.9}$$

Cette distance est alors normalisée par 2 facteurs :

(a) La longueur $\ell(\mathcal{F}_T)$ de la diagonale dans $\mathcal{F}_T$: ceci a pour effet de produire une valeur indépendante de la taille de la figure cible.

(b) La quantité de points en $\mathcal{F}_T$, $\eta(\mathcal{F}_T)$ (parce qu'une figure avec plus de points est plus difficile à approximer).

Maintenant un indice de dissimilarité sans unités peut être défini (Eq. (3.10)).

$$\mathcal{D}(\mathcal{F}_T, \mathcal{F}_C) = \mathcal{D}_f(\mathcal{F}_T, \mathcal{F}_C)/(\ell(\mathcal{F}_T) * \eta(\mathcal{F}_T)) \tag{3.10}$$

c. **Vote par un utilisateur humain** L'idée ici est de présenter à un utilisateur *expert* chaque comparaison d'image à être faite par le système. Les images sont montrées et les votes sont recueillis par une routine en **MATLAB**.

Ceci définit alors $\mathcal{D}$, une distance entre deux figures. La comparaison qui doit se faire entre deux processus de développement (*cible* et *candidat*) est une comparaison de deux listes de figures ; $\mathcal{D}$ est donc généralisé pour une liste de n figures.

Soient $\mathcal{L}_f$ et $\overline{\mathcal{L}}_f$ deux listes de figures.

$$\begin{aligned}\mathcal{L}_f &= \{f_{(1)}, f_{(2)}, \ldots, f_{(k)}, \ldots, f_{(n)}\} \\ \overline{\mathcal{L}}_f &= \{\overline{f}_{(1)}, \overline{f}_{(2)}, \ldots, \overline{f}_{(k)}, \ldots, \overline{f}_{(n)}\}\end{aligned}$$

Le *vecteur de dissimilarité* V entre $\mathcal{L}_f$ et $\overline{\mathcal{L}}_f$ est tel que sa composante i est $\mathcal{D}(f_{(i)}, \overline{f}_{(i)})$:

$$V = \begin{pmatrix} \mathcal{D}(f_{(1)}, \overline{f}_{(1)}) \\ \vdots \\ \mathcal{D}(f_{(i)}, \overline{f}_{(i)}) \\ \vdots \\ \mathcal{D}(f_{(n)}, \overline{f}_{(n)}) \end{pmatrix}$$

La dissimilarité entre $\mathcal{L}_f$ et $\overline{\mathcal{L}}_f$ est la norme de ce vecteur (Eq. (3.11))

$$\mathcal{D}(\mathcal{L}_f, \overline{\mathcal{L}}_f) = \sqrt{(\mathcal{D}(f_{(1)}, \overline{f}_{(1)}))^2 + \cdots + (\mathcal{D}(f_{(n)}, \overline{f}_{(n)}))^2} \tag{3.11}$$

L'algorithme paresseux

Maintenant que nous avons apporté des précisions au sujet de l'évaluation de chaque nœud de l'arbre avec les cibles, nous pouvons continuer avec l'approche améliorée de l'algorithme exhaustif ; en particulier, examinons la conjecture suivante :

Conjecture 1. *Soient* Υ, k, m *tels que*

- $\Upsilon = \{(t_1, p_1), \ldots, (t_n, p_n)\}$ *un contexte spatio-temporel*
- $\Upsilon^{(k)}$ *un sous-ensemble de* Υ, $\Upsilon^{(k)} = \{(t_1, p_1), \ldots, (t_k, p_k)\}$ $(k \in \mathbb{N}^+, k < n)$, *et* $\mathcal{D}_{(\mathbf{G}, \Upsilon^{(k)})}$ *un processus de développement défini par* $(\mathbf{G}, \Upsilon^{(k)})$;
- $\Upsilon^{(k+m)}$ *un sous-ensemble de* Υ, $\Upsilon^{(k+m)} = \{(t_1, p_1), \ldots, (t_k, p_k), \ldots, (t_{k+m}, p_{k+m})\}$ $(m \in \mathbb{N}^+, k + m \leq n)$, *et* $\mathcal{D}_{(\mathbf{G}, \Upsilon^{(k+m)})}$ *un processus de développement défini par* $(\mathbf{G}, \Upsilon^{(k+m)})$;

Soit $\mathbf{G}_{opt}^{(k)}$ *la meilleure solution pour* $\mathcal{D}_{(\mathbf{G},\Upsilon^{(k)})}$; $\mathbf{G}_{opt}^{(k+m)}$, *la meilleure solution pour* $\mathcal{D}_{(\mathbf{G},\Upsilon^{(k+m)})}$, *est l'expansion de* $\mathbf{G}_{opt}^{(k)}$ *avec un ensemble approprié de règles* $\mathcal{R}$.

$$\mathbf{G}_{opt}^{(k+m)} = \mathbf{G}_{opt}^{(k)} \bigcup \mathcal{R}$$

La Conjecture 1 permet de faire une approche incrémentale pour la construction de l'arbre des dérivations :

a. En commençant avec l'axiome ω, construire l'arbre de dérivations jusqu'à la profondeur t_1 ; prendre la meilleure grammaire $\mathbf{G}^{(1)}$

b. Avec $\mathbf{G}^{(1)}$, construire l'arbre de dérivations de t_1 jusqu'à t_2

c. Continuer, en utilisant à chaque itération t_i le vainqueur pour construire le sous-arbre de l'itération suivante (t_{i+1})

Ceci implique qu'*il y a des sections de l'arbre des dérivations qui n'ont pas besoin d'être évaluées* ; ou, de façon réciproque, la Conjecture 1 permet de faire un *élagage*[3] de l'arbre de dérivations. Par exemple, l'arbre de la Fig. 44 est élagué en utilisant la Conjecture 1, pour produire la simplification de la Fig. 45 ; dans celle-ci les nœuds noirs représentent les gagnants pour chaque t_i et les grands triangles gris sont les parties de l'arbre qui seront réellement évalués. Il est évident qu'une partie importante de l'arbre n'est pas évaluée ; bien que le nombre *exact* d'évaluations ne puisse être connu à l'avance (car il dépend du problème spécifique), des valeurs statistiques peuvent être reportées. Par exemple, si le processus de développement cible a été généré suivant le contexte $\Upsilon = \{(t_1 = 1, p_1), (t_2 = 2, p_2)\}$ (donc les évaluations doivent se faire à $t_1 = 1$ et $t_2 = 2$) l'arbre complet de dérivations a 35 nœuds nécessitant une évaluation. En faisant l'élagage, la méthode évalue, en médiane, 12 nœuds (37.14% du total) ; au pire cas, la

[3] L'élagage d'un arbre naturel est *une technique sylvicole qui consiste à supprimer, partiellement ou complètement, des branches dans un arbre, afin de le renforcer, de le façonner ou d'alléger sa ramure, ou encore dans le but de produire du bois de qualité sans nœuds* [Office de la langue française, 2001]. Cette métaphore est ici utilisée concernant l'arbre de dérivations.

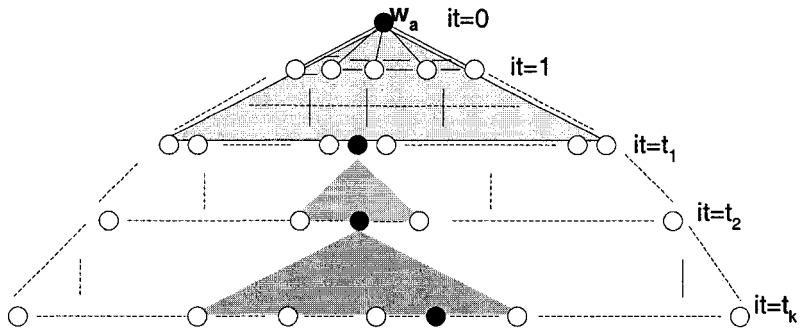


Figure 45 L'arbre de dérivations élagué

méthode devra évaluer 16 nœuds (45.71% du total). Ces valeurs détaillées, avec d'autres calculs concernant différentes valeurs de Υ, sont présentées au Tableau III.

Tableau III

Exemples élagage

		Avec élagage		
Niveaux évalués	Total	meilleur-pire	meilleur-pire (%)	médiane
$[t_1 = 1, t_2 = 2]$	35	$9 - 16$	$25.71 - 45.71(\%)$	$12(34.29\%)$
$[t_1 = 1, t_2 = 3]$	$49,387$	$35 - 32,776$	$0.071 - 66.37(\%)$	$32,776(66.37\%)$
$[t_1 = 2, t_2 = 3]$	$49,406$	$28 - 32,795$	$0.06 - 66.38(\%)$	$32,795(66.38\%)$
$[t_1 = 2, t_2 = 4]$	$2.41 * 10^7$	$54 - 1.43 * 10^7$	$0.00022 - 59.59(\%)$	$1.43 * 10^7(59.59\%)$

Cet élagage de l'arbre de dérivations simplifie les calculs et l'effort de l'algorithme, qui est maintenant donc devenu un algorithme *paresseux*, dans le sens qu'il ne calcule que les nœuds de l'arbre qui peuvent lui être utiles ; cette formalisation est présentée comme l'algorithme 4.

3.3.3 Génération complète de Φ

La dernière étape pour choisir le meilleur modèle pour notre problème concerne la répétition de l'algorithme paresseux (algorithme 4, page 84) pour *tous les axiomes possibles*. Une recherche exhaustive de ce type est exprimé par l'algorithme 6.

Algorithme 4 Algorithme paresseux (pour un seul axiome)

1: **procedure** LAZYALGSINGLEAXIOM(Υ : un contexte spatio-temporel ; $\mathcal{L}_f$: processus de développement ; ω_a : un axiome)

ENTRÉES : $\Upsilon = \{\Upsilon_1, \ldots, \Upsilon_k\} = \{(t_1, p_1), \ldots, (t_k, p_k)\}$ (cf. Def. 3.2) ; $\mathcal{L}_f = \{f_{(1)}, \ldots, f_{(k)}\}$ (cf. Def. 3.3)

SORTIES : $\mathbf{G}$, la grammaire la mieux adaptée pour approximer $\mathcal{L}_f$; f : la distance de l'interprétation de $\mathbf{G}$ à $\mathcal{L}_f$

2: $\mathbf{G} \leftarrow$ grammaire avec axiome ω_a, sans règles

3: $l_f \leftarrow \emptyset$

4: $\mathbf{G} = \text{lazyAlgIter}(\Upsilon, \mathcal{L}_f, \{\omega_a\}, \mathbf{G}, 0, l_f)$ ▷ $l_f = \{f_1, \ldots, f_k\}$, voir algorithme 5

5: Return $[\mathbf{G}, \sqrt{f_1^2 + \ldots + f_k^2}]$

6: **fin procedure**

Algorithme 5 Algorithme paresseux itératif

1: **procedure** LAZYALGITER(Υ : un contexte spatio-temporel ; $\mathcal{L}_f$: processus de développement ; $\overline{S}$: une liste de LString ; $\overline{G}$: une liste de L-System ; d : (int) ; l_f : liste des distances)

ENTRÉES : $\overline{S}$ et $\overline{G}$ sont les nœuds et grammaires (respectivement) trouvés dans l'arbre de dérivations à la profondeur d ; Υ et $\mathcal{L}_f$ comme en algorithme 4

SORTIES : $\mathbf{G}$, la grammaire la mieux adaptée pour approximer $\mathcal{L}_f$; $l_f = \{f_1, \ldots, f_k\}$, où f_i est la distance de l'interprétation de $\mathbf{G}$ sous Υ_i avec $f_{(i)}$

2: $\overline{S}' \leftarrow \emptyset$ ▷ Nouvel ensemble de LString générés

3: $\overline{G}' \leftarrow \emptyset$ ▷ Grammaires *responsables* pour $\overline{S}$'

4: **pour tout** $s_i \in \overline{S}$ **faire**

5: $g_i \leftarrow$ grammaire $\in \overline{G}$ *responsable* pour s_i

6: $R = \{r_1, \ldots, r_k\} \leftarrow$ règles pouvant être appliquées à s_i

7: $p_R \leftarrow$ ensemble de tous les sous-ensembles de R

8: **pour tout** $p_i \in p_R$ **faire**

9: $g'_i \leftarrow g_i \cup p_i$

10: $s_i \xrightarrow{\{g'_i\}} s'_i$ ▷ s'_i est application de g'_i à s_i

11: $\overline{S}' \leftarrow \overline{S}' \cup \{s'_i\}, \overline{G}' \leftarrow \overline{G}' \cup \{g'_i\}$

12: **fin pour**

13: **fin pour**

14: **si** $t_1 \neq d$ **alors** ▷ Ce n'est pas le moment d'évaluer

15: Return lazyAlgIter($\Upsilon, \mathcal{L}_f, \overline{S}', \overline{G}', d+1, l_f$)

16: **sinon** ▷ Il faut évaluer

17: $s_i \leftarrow$ LString dont l'interprétation sous $p_1(\mathcal{I})$ se rapproche le plus de $f_{(1)}$

18: $f_i \leftarrow$ distance de $\mathcal{I}$ à $f_{(1)}$

19: $g_i \leftarrow$ grammaire $\in \overline{G}$ *responsable* pour s_i

20: Return lazyAlgIter($\{\Upsilon_2, \ldots, \Upsilon_k\}, \{f_{(2)}, \ldots, f_{(k)}\}, \{s_i\}, \{g_i\}, d+1, l_f \cup f_i$)

21: **fin si**

22: **fin procedure**

Algorithme 6 Algorithme paresseux exhaustif

1: **procedure** EXHLAZYALG(Υ : un contexte spatio-temporel ; $\mathcal{L}_f$: processus de développement)
SORTIES : G= le meilleur modèle approximant $\mathcal{L}_f$ sous Υ
2: $\mathbf{G} \leftarrow \emptyset$ ▷ Meilleure grammaire
3: $b \leftarrow \infty$ ▷ Distance la plus petite à $\mathcal{L}_f$
4: Générer l'ensemble d'axiomes possibles $\mathcal{A} = \{\omega_1, \ldots, \omega_a, \ldots, \omega_n\}$
5: **pour tout** $\omega_a \in \mathcal{A}$ **faire**
6: $[\mathbf{G}', b'] \leftarrow$lazyAlgSingleAxiom($\Upsilon, \mathcal{L}_f, \omega_a$)
7: **si** $b' < b$ **alors**
8: $\mathbf{G} \leftarrow \mathbf{G}', b \leftarrow b'$
9: **fin si**
10: **fin pour**
11: Retourner G
12: **fin procedure**

Cet algorithme simple et direct repose fortement sur le calcul de la ligne 6.4 : « *générer tous les axiomes existants* » ; si bien que théoriquement il serait possible (avec certaines restrictions) d'effectivement naviguer cet espace, dans la pratique, pour une largeur maximale de l'axiome de 50 symboles, le nombre d'axiomes à visiter s'évalue à 10^{261} (voir Annexe IV pour détails). Il n'est donc pas question d'utiliser un algorithme de ce type : il faut plutôt réduire cet espace de recherche.

Réduire cet espace revient à trouver des raccourcis pour le naviguer ; une des caractéristiques que nous espérons pouvoir exploiter consiste à supposer qu'il existe une structure dans l'espace des solutions : la visite et évaluation d'une certaine grammaire en Φ produit de l'information sur la qualité des solutions environnantes. En d'autres mots, les *bonnes solutions* sont entourées de bonnes solutions. Une méthode (identifiée comme algorithme 7), désigné pour prendre avantage de ces régularités, a été inspiré par l'algorithme génétique (AG) standard (Holland, 1992), (Mitchell, 1996a), (Goldberg, 1989). Une des caractéristiques les plus appréciées de ce type d'algorithmes est son parallélisme implicite : la capacité d'explorer plusieurs régions de l'espace de solutions en même temps, ce qui permet d'explorer des espaces énormes en ne visitant qu'une fraction de ses éléments.

Avant de continuer, il est important de savoir que, pour qu'un algorithme évolutif (en général) ait une bonne performance sur un problème en particulier, sa *structure* (principalement la définition de ses opérateurs) doit refléter des traits du problème à résoudre (ceci est un corollaire direct du *No Free Lunch Theorem*, qui essentiellement dit qu'il n'existe pas de **meilleur** algorithme $\mathcal{V}$, évolutif ou pas, car ce que $\mathcal{V}$ gagne en performance sur une certaine classe de problèmes, il le perd sur une autre (Fogel, 1999), (Wolpert et Macready, 1997)) ; dans notre cas, l'algorithme est assez similaire dans sa structure à l'AG, et ses opérateurs sont définis de façon assez similaire aux opérateurs standards. Ce fait s'explique par l'effort investi pour incorporer l'information a priori dont nous disposons pour le problème, et qui a déjà été fourni dans la définition d'un modèle pour la croissance d'arbres (opérateurs L-System $\mathbf{B}$, $\mathbf{Y}$, et $\mathbf{T}$, et spécification de Φ).

Algorithme 7 Algorithme évolutif

1: **procedure** GÉNÉRERGRAMMAIRE($\mathcal{L}_f$: un processus de développement ; Υ : un contexte spatio-temporel)
ENTRÉES : Fonction d'adéquation $\mathcal{F}$; **MAX-GEN**, nombre maximum de générations
SORTIES : la grammaire
2: Initialiser aléatoirement la population d'axiomes, $\mathcal{P}$
3: gen $\leftarrow 0$ ▷ *gen* garde le numéro actuel de la génération
4: **répéter**
5: **pour tout** axiome $\omega_a \in \mathcal{P}$ **faire**
6: $[\mathbf{G}', b'] \leftarrow$lazyAlgSingleAxiom($\Upsilon, \mathcal{L}_f, \omega_a$)
7: $\mathcal{F}(\omega_a, \mathcal{L}_f, \Upsilon) \leftarrow b'$ ▷ ω_a a la valeur de justesse de sa meilleure classe d'équivalence
8: **fin pour**
9: Prendre $\overline{G}/\ \nexists(\mathbf{G} \in \zeta(\omega_a) \implies \mathcal{F}(\mathbf{G}, \mathcal{L}_f, \Upsilon) < \mathcal{F}(\overline{G}, \mathcal{L}_f, \Upsilon))$ ▷ $\overline{G}$: grammaire avec meilleure adéquation
10: gen $\leftarrow$ gen + 1
11: Générer un nouveau $\mathcal{P}$ basé sur $\mathcal{F}(\omega_*, \mathcal{L}_f, \Upsilon)$
12: **jusqu'à** ($\overline{G}$ est *suffisamment bon*) **ou** (gen > **MAX-GEN**)
13: Retourner $\overline{G}$
14: **fin procedure**

L'algorithme procède en créant une population d'axiomes (ligne 7.2) ; chacun de ces axiomes ω_a propose la meilleure grammaire possible $\mathbf{G}$ pour le problème à résoudre (ligne 7.6)

en utilisant l'algorithme 4. L'axiome ω_a a alors la *performance* de G (ligne 7.7) ; cette performance est la distance entre la proposition généré par ω_a et le processus cible, décrite dans les pages 75-81. Selon leur performance, les axiomes sont sélectionnés pour être combinés à d'autres membres de la population (ligne 7.11).

3.3.3.1 La représentation des axiomes

Utiliser l'algorithme 7 consiste, d'abord, à créer une représentation (le *génotype*) des individus dans la population. Les individus étant les axiomes de chacune des grammaires, leur représentation est assez directe ; elle est basée sur trois structures :

a. **Un gène** est la structure de base dans un LString. Un gène peut prendre les valeurs suivantes : $F(s), -(a) \mid \backslash(a) \mid /(a), \&(a), \mathbf{B}(s,a,a), \mathbf{Y}(s,a,a,a), \mathbf{T}(s,a,a)$, avec $s, a \in \mathbb{N}$

b. **Une branche linéaire** est une séquence de gènes ; une branche peut donc être $F(1)$, ou $\mathbf{B}(1,10,25)F(3) + (30)$, par exemple.

c. **Un arbre ou axiome** est une séquence de branches linéaires *et/ou* arbres (ceci permet des branches complexes) ; chacune d'entre elles est aussi associée à un bit de *crochet* : si celui-ci est 1, cela signifie que la branche linéaire forme vraiment une branche sur l'arbre ; s'il est 0, c'est tout juste une continuation de la branche courante.

Cette structure est décrite par la grammaire du Tableau IV.

Comme exemple, un arbre généré par l'interprétation du LString

$$F[+F[+F]F[-F]F]F(1/2)[-F]F$$

est montré à la Fig. 46

Tableau IV

Grammaire pour la représentation d'un axiome

arbre	: :	axiome	
axiome	: :	$\{\text{axiome} \mid \text{branche linéaire}\}^+$	*Axiome : séquence non-vide d'axiomes ou de branches*
branche linéaire	: :	$\{\text{gene}\}^+$	*Une branche linéaire est une séquence non-vide de gènes*
gène	: :	$F(s)$	*Un gène peut être n'importe lequel des symboles de l'alphabet*
	|	$+(a) \mid -(a) \mid \backslash(a) \mid /(a) \mid \&(a)$	
	|	$\mathbf{B}(s,a,a) \mid \mathbf{Y}(s,a,a,a) \mid \mathbf{T}(s,a,a)$	
s	: :	$\in [1,10]$	*pas ou distance*
a	: :	$\in \pm\{15,30,45,60,\ldots$ $\ldots,75,90,105,120,\ldots$ $\ldots,135,150,165,180\}$	*angle*

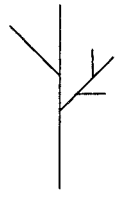
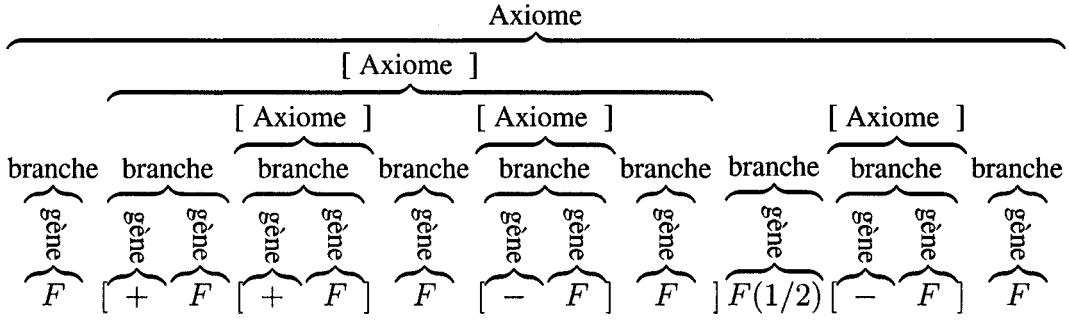


Figure 46 La représentation d'un axiome

Il est codifié comme :



3.3.3.2 Les opérateurs génétiques

La ligne 7.11 de l'algorithme 7 réfère à la génération d'une nouvelle population d'axiomes, en ayant mesuré la performance de la population actuelle. Ceci est implanté en 3 pas :

a. des paires d'individus (i_1, i_2) de la population sont **sélectionnés** en utilisant un algorithme de roulette standard (Mitchell, 1996b) sur leurs adéquations.

b. (i_1, i_2) sont recombinés avec probabilité parfaite (1) en utilisant un algorithme de **croisement**

c. Chacun des individus résultants sont **mutés** avec probabilité p_m

Les opérateurs de *croisement* et *mutation* sont décrits ici-bas.

a. **Croisement**[4] : soit p_1 et p_2 deux axiomes codifiés comme décrit en 3.3.3.1 ; p_1 a n_1 branches (Fig. 47), p_2 en a n_2 (Fig. 48) ; dans le cas général, $n_1 \neq n_2$

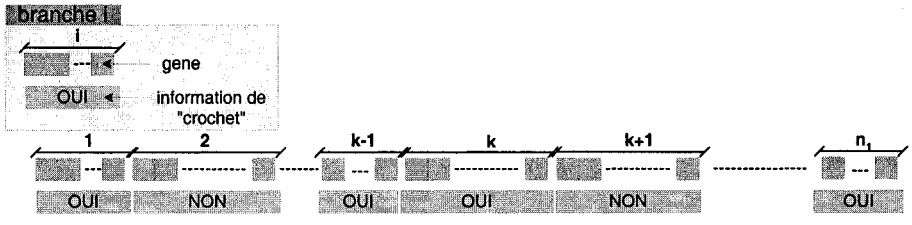


Figure 47 Représentation en génome pour i_1

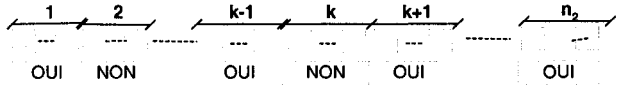


Figure 48 Représentation en génome pour i_2

Génome : séquence de branches, elles-mêmes une séquence de gènes (Fig. 47, en haut à gauche)

p_1 et p_2 sont les axiomes « *parents* » ; leur croisement s'effectue comme suit :

[4] Cet opérateur est l'adaptation, à notre problème, de l'opérateur de croisement à 1 point en algorithmes génétiques (Mitchell, 1996b).

(a) Choisir une branche aléatoire k ($k \leq n_1$ et $k \leq n_2$) pour le point de croisement. La branche k-èsième de p_1 (b_1) et de p_2 (b_2) sont des séquences de gènes.

(b) Choisir un point de croisement c *à l'intérieur des branches* b_1 et b_2 ; échanger des parties de chaque branche (Fig. 49). Ceci produit deux « branches filles »

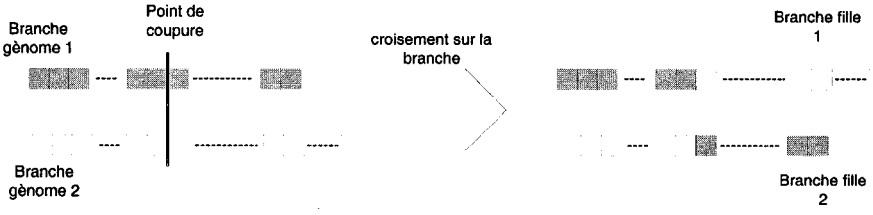


Figure 49 Branches échangeant des gènes

bf_1 et bf_2, chacune avec une portion de l'information des branches parents b_1 et b_2.

(c) Construire deux « génomes fils » f_1 et f_2 :

i. f_1 : faire la concaténation des branches 1 à k de p_1, suivi par bf_1, suivi par les branches $k+1$ à n_2 de p_2

ii. f_2 : faire la concaténation des branches 1 à k de p_2, suivi par bf_2, suivi par les branches $k+1$ à n_1 de p_1

Le résultat de l'échange au complet est montré à la Fig. 50

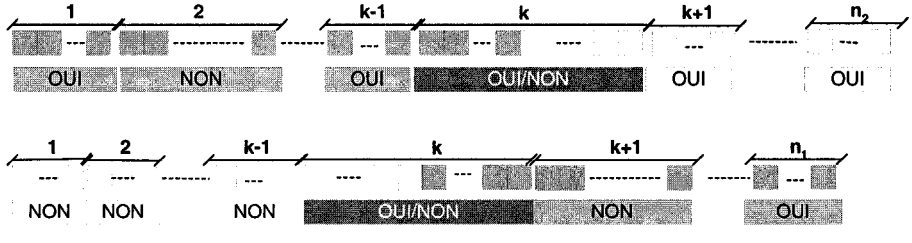


Figure 50 Résultat du croisement

b. **Mutation**[5] : il y a une probabilité p_m de changer n'importe quel gène dans un individu (ou chromosome). Ceci est implanté comme suit, dans chaque structure :

   - **Dans un axiome** : envoyer à chaque branche un *message* pour la muter ; après, avec probabilité p_m, changer son statut de crochet : si celui-ci est "OUI", mettre "NON", et viceversa. Ensuite, faire, avec probabilité p_m
     (a) créer une branche
     (b) éliminer une branche
   - **Dans une branche linéaire** : pour chaque gène, avec probabilité p_m, envoyer un signal pour le muter ; et, avec probabilité p_m
     (a) créer un gène
     (b) éliminer un gène
   - **Pour un gène** : avec probabilité p_m, changer ce gène par un autre, avec un nouvel ensemble de paramètres.

Prochaine étape

Dans ce chapitre tous les détails de la méthode proposée pour résoudre le *problème inverse* ont été présentés : un formalisme pour représenter des squelettes d'arbres naturels et un algorithme pour naviguer l'espace induit par ce formalisme en sont les composantes principales. Dans le prochain chapitre un ensemble de problèmes sera présenté à la méthode, pour essayer de cerner ses capacités et ses faiblesses ; postérieurement, des conclusions seront extraites de ces résultats.

[5] Cet opérateur est l'adaptation, à notre problème, de l'opérateur de mutation en algorithmes génétiques (Mitchell, 1996b).

CHAPITRE 4

EXPÉRIENCES ET RÉSULTATS

Je décris dans ce chapitre un ensemble d'expériences conduites pour tester la performance de l'algorithme proposé ; ces tests sont conduits sur des formes 3D générées par un ensemble de grammaires (décrites ci-bas). Les deux premières sections de ce chapitre détaillent l'ajustement des paramètres propres au fonctionnement de l'algorithme : en 4.1, par le choix d'une fonction d'adéquation appropriée pour les besoins de la thèse, et en 4.2, en réglant des détails de fonctionnement. Tout ceci se fait par la répétition de deux expériences « simples », tout en mesurant leur performance par rapport aux paramètres choisis.

La section 4.3 rapporte des résultats graphiques d'expériences menées à bien avec l'algorithme.

4.1 Pertinence des fonctions d'adéquation

Trois fonctions d'adéquation (fonction de comparaison de squelettes d'arbres) ont été décrites au chapitre 3 (détails en page 75) :

a. Une fonction de comparaison générée à l'aide d'un réseau de neurones.

b. Une fonction de distance entre les points des structures.

c. Un utilisateur humain donnant directement les valeurs des comparaisons.

Dans cette section, l'objectif est d'évaluer les trois fonctions sur un problème de reconstruction pour ainsi observer leurs performances et décider quelle fonction utiliser pour le reste des expériences. Pour cela, j'ai choisi comme problème de faire l'approximation de

la figure présentée en Fig. 51, résultat d'une (1) itération de la grammaire $\mathbf{G}_1$.

$$\mathbf{G}_1 :: \begin{cases} \omega = F \\ r_1 : F \longrightarrow F[+F]F[-F] \end{cases}$$

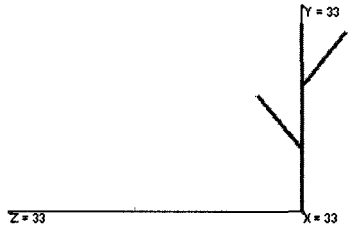


Figure 51 Une figure simple

Une (1) itération de la grammaire $\mathbf{G}_1$ produit un objet 3D ; projeté suivant la convention de la Fig. 25, avec $d = 1$, $w = 100$, $h = 100$, $F = 1$, $B = 1$

Comme il peut être apprécié, cette expérience est simple du point de vue des calculs : l'algorithme doit seulement itérer une (1) fois les solutions candidates, ce qui rend l'arbre de dérivations petit.

À chaque exécution de l'algorithme, un ensemble de paramètres pour celui-ci doit être choisi ; cet algorithme étant une modification de l'algorithme génétique standard, deux groupes de paramètres doivent être choisis :

a. **Paramètres correspondants à l'algorithme génétique standard**

   (a) Nombre d'individus formant la population n

   (b) Pourcentage d'élitisme e : quel pourcentage des meilleurs individus de la population à la génération g passe directement à la population pour la génération $g + 1$

(c) Probabilité de mutation m : probabilité de changer aléatoirement un gène d'un individu de la population

(d) Probabilité de croisement c : probabilité que deux génomes choisis de la population se croisent

(e) Condition de l'arrêt de l'algorithme : nombre maximum de générations et/ou critère de justesse des solutions

b. **Paramètres correspondant à cet algorithme en particulier**

(a) Nombre de gènes formant un individu

(b) Nombre de symboles dérivables (**B**, **Y**, et **T**) maximum « acceptés » dans chaque génome : tel que décrit à la section 3.2.2, page 55, les symboles dérivables sont la base de l'algorithme, et sont directement responsables des différentes expansions que les LString exhibent. Bien qu'il soit préférable, en pratique, d'accepter un nombre illimité de symboles dérivables dans les individus formant la population, en pratique il faut limiter ce nombre car les expansions explosent exponentiellement (comme décrit à l'annexe IV)

Ces paramètres sont résumés au Tableau V.

Tableau V

Paramètres à définir pour les expériences

Simulation		**Individu**
Taux de mutation m	Taille population n	Max. symb. dérivables
Taux de croisement c	Max. générations	Gènes par génome
Taux d'élitisme e		

Ces paramètres seront spécifiés à chaque instance de l'algorithme.

Les résultats de l'exécution de l'algorithme sont en 4.1.1, pour la votation directe d'une personne, en 4.1.2, pour la fonction synthétisée par un réseau de neurones, et en 4.1.3 pour la fonction de comparaison point à point.

4.1.1 Évaluation directe par l'usager

Les votes donnés par un utilisateur sont ici considérés comme la sortie d'une fonction d'évaluation ; ces votes prennent le rang $[0, 10]$ pour indiquer une ressemblance parfaite avec la figure cible (vote = 10) ou le manque complet de ressemblance (vote = 0). La fonction d'adéquation requise par l'algorithme est définie comme :

$$\text{(adéquation) } a = 10 - \text{(vote) } v;$$

Il est donc important de noter qu'une valeur de 0 pour la fonction d'adéquation est équivalent à une ressemblance parfaite.

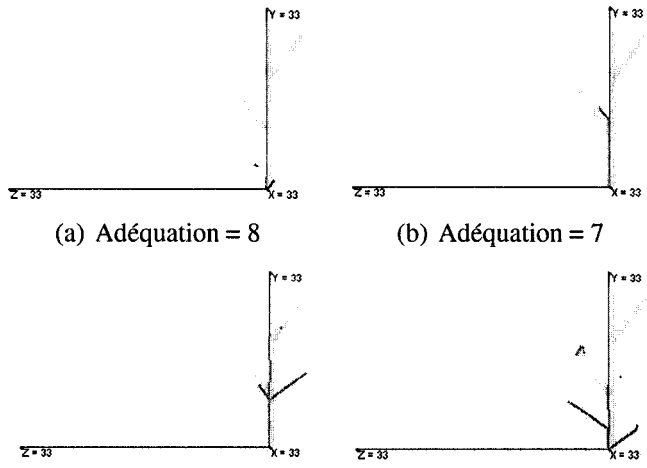
L'ensemble de paramètres de l'algorithme choisi pour cette expérience est montré au Tableau VI.

Tableau VI

Paramètres pour adéquation *définie par l'usager*

Simulation				**Individu**	
Taux de mutation m	0,02	Taille population n	30	Max. symb. dérivables	3
Taux de croisement c	1	Max. générations	100	Gènes par génome	100
Taux d'élitisme e	20%				

Les résultats de cette expérience sont présentés à la Figure 52 ; (a) a été obtenue à la génération 1 de l'algorithme, (b) à la génération 2, (c) à la génération 42 et (d) à la génération 62.



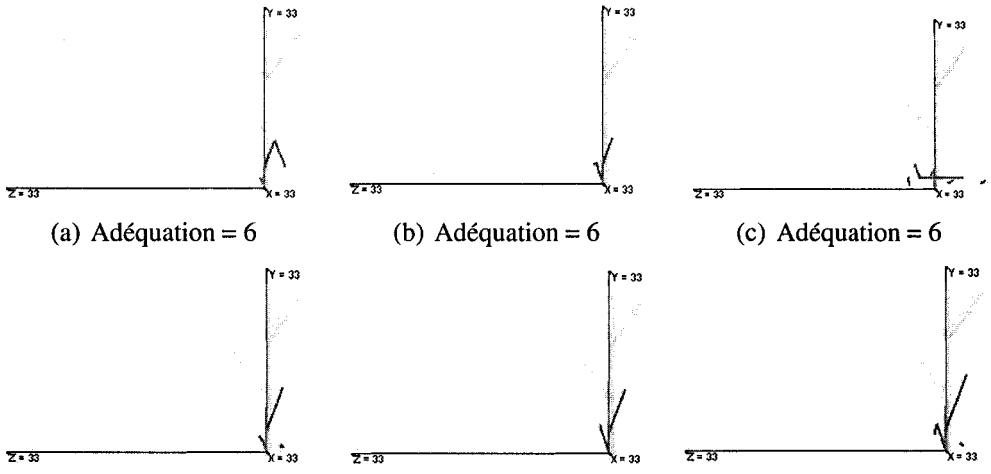
(a) Adéquation = 8 (b) Adéquation = 7

(c) Adéquation = 6 (d) Adéquation = 5

Figure 52 Algorithme avec adéquation *usager* (I)

Il est graphiquement facile d'observer comment les améliorations d'adéquation reportées sont synonymes d'améliorations qualitatives dans les solutions : la solution de la Fig. 52(a) (adéquation = 8), est juste une petite figure avec une partie qui ressemble à une branche. Déjà à la génération 2 (Fig. 52(b), adéquation = 7), un « *tronc* » a été généré avec une petite branche. Le saut à une adéquation de 6 (Fig. 52(c)), correspond à l'apparition de 2 branches ; et l'apparition d'une solution d'adéquation égale à 5 (représentée à la Fig. 52(d)), survient quand ces 2 branches sont asymétriques sur le tronc (ne naissent pas au même point), tout comme elles le sont dans la figure cible (Fig. 51).

Donc l'algorithme génère des solutions *qualitativement* meilleures au cours de son fonctionnement ; ceci est, bien sûr, une propriété désirable et que nous recherchons. Cependant, une population de 30 individus est extrêmement petite pour un algorithme explorant un espace de solutions ; une autre instance de l'algorithme, en changeant le nombre d'individus dans la population à 60, est exécutée et les solutions montrées à la Fig. 53. Les générations 1 (a), 22 (b), 25 (c), 72 (d), 96 (e) et 166 (f) sont montrées, avec leur valeur d'adéquation respective. La figure cible est montrée en arrière plan de chaque solution.



(a) Adéquation = 6 (b) Adéquation = 6 (c) Adéquation = 6

(d) Adéquation = 5 (e) Adéquation = 5 (f) Adéquation = 5

Figure 53 Algorithme avec adéquation *usager* (II)

Encore une fois, l'amélioration de l'adéquation correspond bien à une amélioration des solutions. Cependant, le plus grand problème de cette méthode est le temps requis pour compléter les expériences : avec 30 individus, 100 générations sont faites en 12 heures, où l'expérimentateur ne peut faire autre chose que voter. Ceci apporte d'autres problèmes, le plus important étant que, comme il y a forcément peu d'individus dans la population, la diversité génétique de celle-ci est petite, et donc l'exploration de nouveaux candidats est compromise ; je crois que c'est ce que nous observons à la Fig. 53, où les meilleures solutions sont petites et les branches ne correspondent pas à la disposition cherchée, mais la population n'a pas *l'inertie* pour entreprendre un changement de direction dans la recherche.

4.1.2 Avec un réseau de neurones

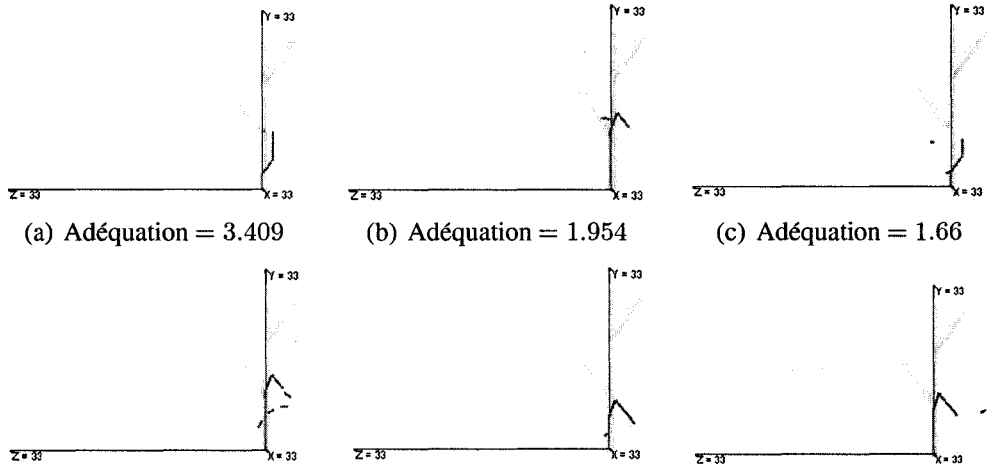
Une fonction d'évaluation est générée par l'entraînement d'un réseau de neurones ; cette méthode est décrite de façon résumée à la page 75, et en détail à l'annexe III. L'algorithme prend les paramètres du Tableau VII.

Tableau VII

Paramètres pour adéquation par réseaux de neurones

Simulation				**Individu**	
Taux de mutation m	0,02	Taille population n	60	Max. symb. dérivables	3
Taux de croisement c	1	Max. générations	100	Gènes par génome	100
Taux d'élitisme e	20%				

Les résultats de l'exécution de l'algorithme (en figures générées) sont présentés à la Fig. 54. Les générations 1 (a), 3 (b), 4 (c), 19 (d), 32 (e) et 97 (f) sont montrées, avec leur valeur



(a) Adéquation = 3.409 (b) Adéquation = 1.954 (c) Adéquation = 1.66

(d) Adéquation = 1.586 (e) Adéquation = 1.4096 (f) Adéquation = 1.392

Figure 54 Algorithme avec adéquation par réseau de neurones

d'adéquation respective. La figure cible est montrée en arrière plan de chaque solution.

Dès les premières générations, les valeurs d'adéquation sont bonnes : à la génération 1, cette valeur est déjà 3.409 (il faut se souvenir que la première génération est composée d'individus générés au hasard, donc cette valeur est celle correspondante à une solution complètement aléatoire) ; cette valeur s'améliore aussi très rapidement : 1.66 dès la génération 4. Comme une conséquence parallèle à cette remarque, l'algorithme est (dans la pratique) incapable de trouver de meilleures individus par la suite, et les valeurs d'adéquation ne diminuent pas rapidement.

Cependant, la similarité visuelle des figures générées (Fig. 54(a)-(f)) avec la figure cible (Fig. 51) n'est pas satisfaisante : en comparaison aux valeurs d'adéquation obtenues, de meilleures figures seraient envisageables. La fonction d'adéquation paraît avoir une certaine tendance à surévaluer la qualité des solutions, ce qui indique qu'elle n'est pas au point ; je crois que la direction et l'idée de codifier les votes humains est correcte, mais il est clair qu'un protocole d'expérimentation plus élaboré est nécessaire pour rendre cette fonction d'adéquation par réseau de neurones plus discriminante. Ce protocole inclurait, au minimum, une base de données de votes plus grande (englobant une variabilité plus grande de formes de structures à branches) et un nombre significatif d'expérimentateurs humains.[1]

4.1.3 Distance point par point.

La fonction point-par-point est essentiellement une fonction de distance entre deux ensembles de points ; cette méthode est décrite de façon détaillée à la page 75. La distance entre la liste de figures candidate et la liste de figures cible, mesurée avec cette fonction point-par-point, constituera l'adéquation de la candidate ; ceci implique une différence avec les deux autres fonctions d'adéquation (fonction manuelle et par réseau de neurones) présentées : tandis que celles-ci prennent des valeurs allant de 0 à 10, la fonction point-par-point atteint une valeur de 0 quand la similitude est parfaite, mais n'a pas de valeur maximale de dissimilarité.

L'algorithme a été exécuté avec les paramètres spécifiques du Tableau VIII.

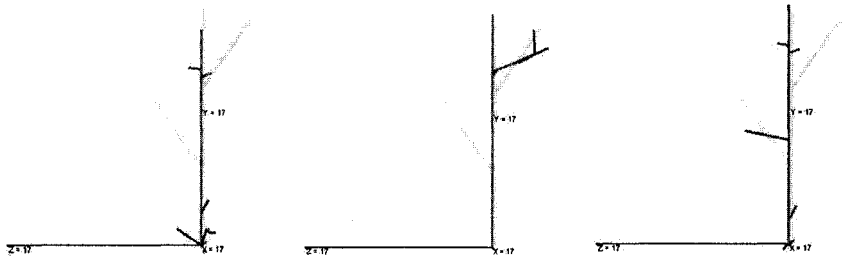
La Fig. 55 présente les meilleures solutions générées pour l'algorithme évolutif aux générations 1 (a), 10 (b), 11 (c), 33 (d), 284 (e) et 479 (f). Chaque solution a, en arrière plan, la figure cible, donnant ainsi une indication visuelle de sa qualité.

[1] Il serait aussi envisageable de changer l'architecture du réseau, pour en faire un apprentissage non-supervisé. Cette option reste ouverte comme direction de recherche.

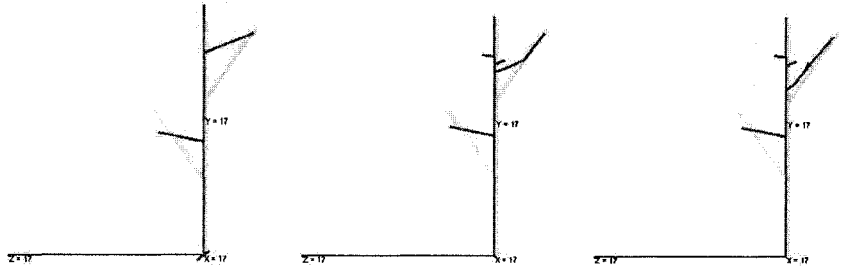
Tableau VIII

Paramètres pour adéquation par distance point-à-point

Simulation				**Individu**	
Taux de mutation m	0,02	Taille population n	200	Max. symb. dérivables	3
Taux de croisement c	1	Max. générations	500	Gènes par génome	100
Taux d'élitisme e	20%				



(a) **Génération 1** (b) **Génération 10** (c) **Génération 11**



(d) **Génération 33** (e) **Génération 284** (f) **Génération 479**

Figure 55 Algorithme avec adéquation point-par-point

Il peut être observé comment l'algorithme passe par un ensemble de solutions qui ont des détails qui se rapprochent des détails architecturaux de la cible : à la génération 1 (Fig. 55(a)), la meilleure solution a seulement le tronc en commun avec la cible, et les petites branches ne suivent aucune des caractéristiques observée dans la cible. Il est simple de justifier le fait que la meilleure solution soit assez « *aléatoire* », car *toute la population* est aléatoire au début, et donc il n'y a encore eu aucune pression évolutive d'exercée sur elle à la génération initiale.

Aux générations 10 et 11 (Fig. 55(b) et (c)), les meilleures candidates sont des solutions avec 1 branche, « *presque* » alignées avec une branche de la figure cible ; l'algorithme a obtenu une solution avec deux (2) branches à la solution 33 (Fig. 55(d)) et à partir de là il s'est concentré à approcher ces deux branches des deux branches cibles (Fig. 55(e)-(f)). La Fig. 55(f) est la meilleure approximation obtenue en 500 générations.

Les mêmes caractéristiques qualitatives ont été obtenues dans toutes les instances de cette première expérience : premièrement, une phase d'exploration très rapide de l'espace de recherche (exprimée ici par les résultats de la génération 10 à la génération 33), suivie par une longue période d'*exploitation* de la meilleure zone explorée (de la « découverte » des deux branches à la solution 33 jusqu'à la meilleure solution de la génération 479).

Cette fonction d'adéquation point-par-point est celle qui s'est avérée la meilleure pour cette expérience initiale, même si ses fonctionalités sont loin de mimer de façon précise le processus de comparaison d'un être humain. Donc, pour continuer à tester les attributs de l'algorithme, je choisis de continuer avec cette fonction de comparaison, et c'est celle-ci qui sera utilisée pour la poursuite de ce travail.

4.2 Paramétrisation et ajustement de l'algorithme

L'objectif de cet ensemble d'expériences est de déterminer les paramètres spécifiques pour l'Alg. 1 ; ces paramètres ont été présentés au Tableau V.

Des expériences préliminaires m'ont permis de cerner les valeurs de certains paramètres[2] ; ceux-ci sont :

a. Le *rang de grandeur* (nombre de gènes) pour un génome a été fixé à $[1, 100]$: chaque génome aura au moins 1 gène et au plus 100.

[2] Comme travail futur il serait probablement sage de faire un balayage expérimental de ces paramètres. Pour cette thèse j'ai seulement fait quelques expériences pour tester le terrain et évaluer une valeur des paramètres convenables pour mes objectifs immédiats.

b. Le *nombre maximal de symboles dérivables* (**B**, **Y** et **T**) a été fixé à 3 (trois).

c. Le taux de changement des gènes d'une population à l'autre (ce qui peut être appelé la « *stabilité* » d'une population) est essentiellement guidé par le taux de mutation m et le pourcentage d'élitisme e. J'ai décidé de fixer le taux de mutation à $0,02$, et varier le taux d'élitisme.

d. La Probabilité de croisement c est fixée à 1 : si deux génomes dans la population sont choisis, ils se croiseront.

Cette information est exprimée au Tableau IX. Les deux paramètres restants (taille de la

Tableau IX

Paramètres par défaut pour les expériences (I)

Simulation			**Individu**	
Taux de mutation m	0,02	Taille population n	Max. symb. dérivables	3
Taux de croisement c	1	Max. générations	Gènes par génome	$\in [1, 100]$
Taux d'élitisme e				

population et taux d'élitisme) seront choisis pour maximiser la capacité de générer des processus de développement aussi proches que possible de la cible ; cette capacité sera appelée l'*exactitude* de l'algorithme. En même temps, je voudrais explorer les caractéristiques des populations utilisées par l'algorithme ; ceci pourrait me donner des pistes concernant, par exemple, la capacité d'explorer adéquatement l'espace des solutions. L'espace de solutions possibles étant tellement immense (comme le démontrent les calculs à l'annexe IV), l'algorithme devrait s'assurer de bien répartir ses individus dans celui-ci : autrement, la probabilité de rester bloqué dans un minimum local serait importante. Pour avoir une idée intuitive et quantitative de ces deux points, les mesures suivantes sont calculées sur une population donnée :

a. L'adéquation du meilleur individu.

b. Le pourcentage d'individus structurellement uniques.

c. L'adéquation médiane et moyenne.

Cette exploration pour faire le choix des paramètres permettra de dégager des caractéristiques intéressantes de l'algorithme.

Trente (30) expériences ont été menées à bien pour chaque combinaison (n, e) des paramètres, avec $n \in \{200, 400\}$ et $e \in \{5\%, 20\%, 33\%\}$[3], pour un total de 180 réplications.

Deux expériences ont été choisies à ce sujet, les deux cherchant à faire une approximation d'un processus de développement très simple, consistant en une (1) seule figure ; elles sont présentées aux sous-sections 4.2.1 et 4.2.2. Trente (30) répétitions pour chaque groupe de paramètres ont été conduites dans chaque expérience. Les résultats finaux de cette partie sont résumés à la sous-section 4.2.3

4.2.1 Première expérience

La première expérience consiste à faire l'approximation d'un processus de développement ayant une seule figure, présentée à la Fig. 51, résultat d'une (1) itération de la grammaire G_1 (page 93). Je reprends ici la définition (Fig. 56) pour faciliter la compréhension du texte.

$$\mathbf{G_1} :: \begin{cases} \omega = F \\ r_1 : F \longrightarrow F[+F]F[-F] \end{cases}$$

Tel que mentionné, cette expérience est simple du point de vue des calculs : l'algorithme doit seulement itérer une (1) fois les solutions candidates, ce qui rend l'arbre de dérivations petit. Mais même dans ce cas, 1 génération prend environ 2 minutes quand la taille de la population est de 200 individus (donc environ 15 heures pour 500 générations). Ce temps

[3] Des expériences ont été conduites avec d'autres valeurs (n, e) pour tenter de voir avec quelles valeurs conduire les expériences importantes. Quoiqu'elles ne sont pas montrées ici, elles ont permis de retenir les paramètres $n \in \{200, 400\}$ et $e \in \{5\%, 20\%, 33\%\}$

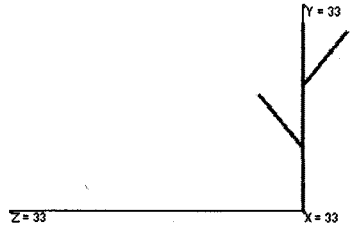


Figure 56 Une figure simple (reprise de la Fig. 51)

Une (1) itération de la grammaire $\mathbf{G}_1$ produit un objet 3D ; projeté selon la convention présentée à la Fig. 25, avec $d = 1$, $w = 100$, $h = 100$, $F = 1$, $B = 1$

de calcul est appréciable (surtout en considérant que c'est l'expérience la plus simple à conduire) ; j'ai alors décidé de fixer le nombre maximum de générations à exécuter à 500 pour toutes les expériences futures.

Les paramètres par défaut, déjà spécifiés au Tableau IX, deviennent tels que montrés au Tableau X.

Tableau X

Paramètres par défaut pour les expériences (II)

Simulation				**Individu**	
Taux de mutation m	0,02	Taille population n		Max. symb. dérivables	3
Taux de croisement c	1	Max. générations	500	Gènes par génome	$\in [1, 100]$
Taux d'élitisme e					

La Fig. 57 montre les diagrammes en boîte[4] des meilleures valeurs d'adéquation[5] des expériences. Le Tableau XI montre les valeurs moyennes avec les écarts types associés.

[4] Dans ces « diagrammes en boîte » (*boxplot* en anglais), chaque boîte a des lignes au quartile inférieur, médian, et supérieur ; la portion des boîtes qui est dans le 95% des données est montré par un rétrécissement de celle-ci. Deux lignes (les *whiskers*, en anglais) continuent les boîtes pour montrer l'extension du reste des données. Les observations aberrantes (*outliers*) sont des points de données qui se situent plus loin de la fin de ces lignes.

[5] Adéquation : mesure de l'Eq. 3.10 multipliée par 1,000, pour des raisons de facilité de manipulation. Cette valeur d'adéquation n'a pas de borne supérieure, comme commenté en page 99.

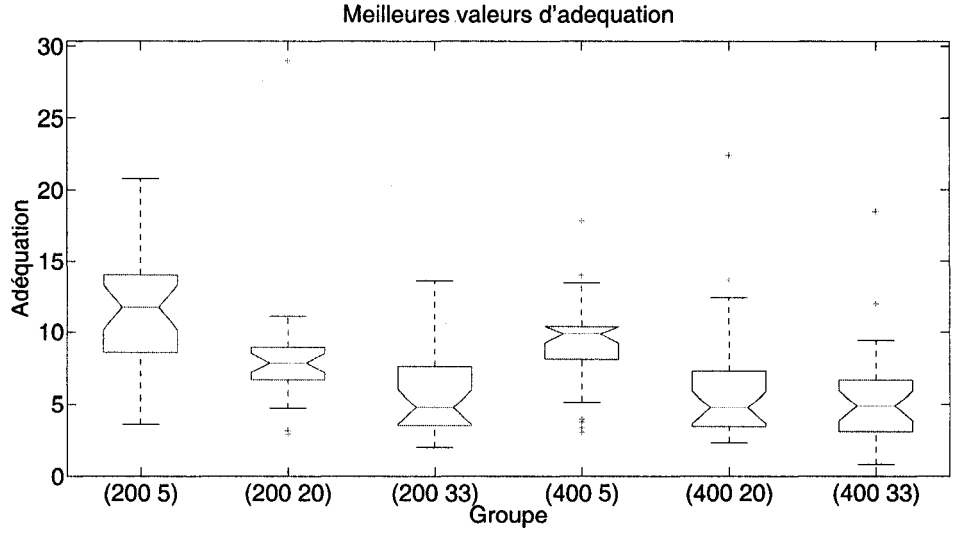


Figure 57 Meilleures valeurs d'adéquation par groupe

Chaque *groupe* contient les valeurs d'adéquation des meilleurs individus des 30 réplications ; les groupes sont présentés sous forme (taille population, élitisme). L'adéquation est la valeur de l'Eq. 3.10 multipliée par $1,000$

Tableau XI

Valeurs d'adéquation (moyennes et écarts types) par groupe

(n, e)	Adéquation	(n, e)	Adéquation
$(200, 5\%)$	11.33 ± 4.23	$(400, 5\%)$	9.27 ± 3.36
$(200, 20\%)$	8.12 ± 4.41	$(400, 20\%)$	5.78 ± 3.79
$(200, 33\%)$	5.60 ± 2.55	$(400, 33\%)$	5.89 ± 3.35

Les meilleures valeurs (les plus petites) furent obtenues pour les populations avec $e = 33\%$, suivies par les populations avec $e = 20\%$, et finalement celles avec $e = 5\%$. Également, les valeurs sont plus regroupées dans les groupes avec $e = 33\%$: les valeurs des écarts types (Tableau XI) et les boîtes montrant le premier quartile (Fig. 57) sont plus petites que pour les autres groupes. En d'autres mots, les populations correspondants à ces groupes font une exploitation de l'espace assez « semblable », les résultats étant donc plus stables.

Il est aussi à remarquer qu'il n'y a pas de différence statistique entre les population avec le même pourcentage d'élitisme et un nombre variable d'individus (donc $(200, 20\%)$ et $(400, 20\%)$ sont égales statistiquement, tout comme $(200, 5\%)$ et $(400, 5\%)$, et $(200, 33\%)$ et $(400, 33\%)$)[6]

Donc, jusqu'ici, tout semble indiquer qu'il serait judicieux de prendre comme la paire performant le mieux $(200, 33\%)$, et de vérifier si ces résultats se retrouvent dans le comportement des populations au complet. La performance des populations peut être intuitivement expliquée par ses valeurs d'adéquations moyennes (Fig. 58) et médianes (Fig. 59).

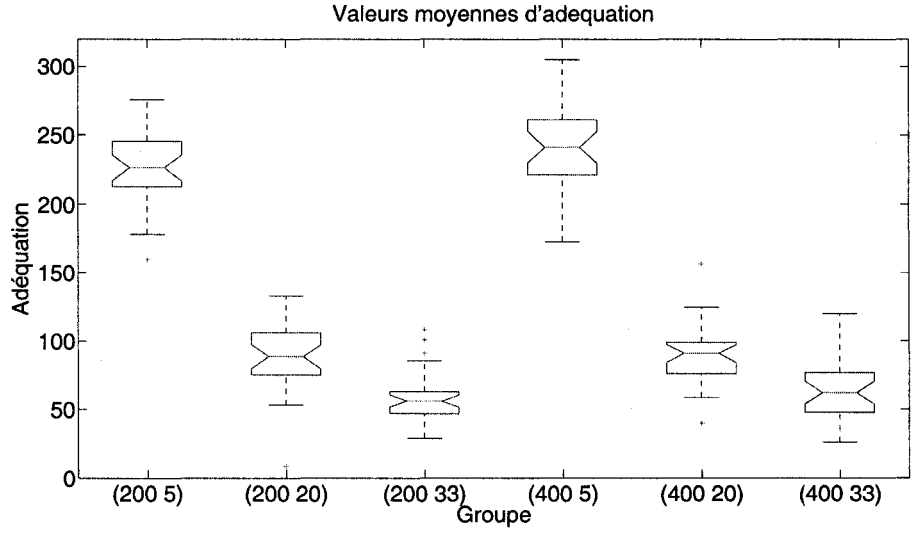


Figure 58 Moyennes des valeurs d'adéquation par groupe

Chaque *groupe* contient les valeurs moyennes d'adéquation des 30 réplications ; les groupes sont présentés sous forme (taille population, élitisme). L'adéquation est la valeur de l'Eq. 3.10 multipliée par $1,000$

La même tendance que pour les meilleures valeurs est retrouvée pour ces mesures : en augmentant e,

[6] La signification statistique de l'hypothèse nulle de non-différence fût déterminée par les ANOVAS non-paramètriques de Kruskal-Wallis à $p = 0,01$

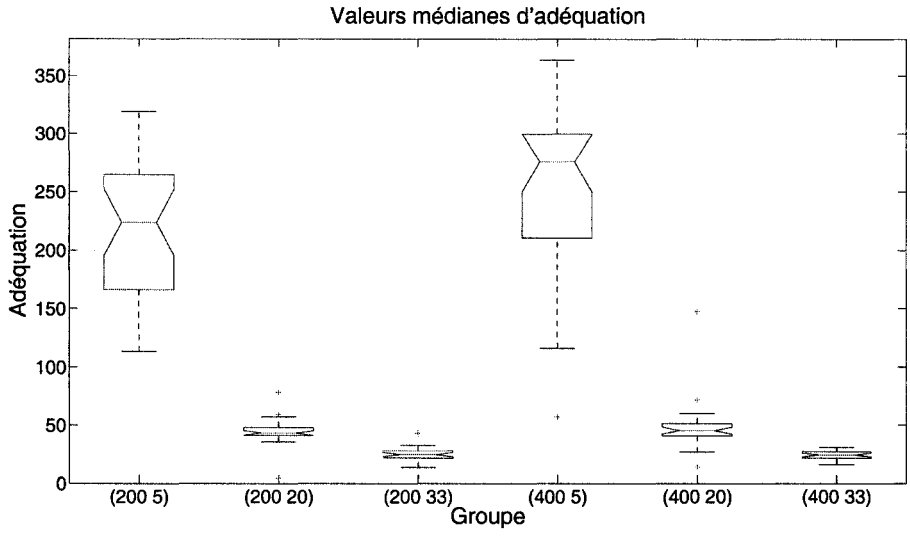


Figure 59 Médianes des valeurs d'adéquation par groupe

Chaque *groupe* contient les valeurs médianes d'adéquation des 30 réplications ; les groupes sont présentés sous forme (taille population, élitisme). L'adéquation est la valeur de l'Eq. 3.10 multipliée par $1,000$

a. la moyenne et la médiane sont meilleures : les populations avec $e = 33\%$ ont une meilleure adéquation que les populations avec $e = 20\%$, qui ont une meilleure adéquation que celles avec $e = 5\%$

b. les valeurs sont plus regroupées (les boîtes représentant les deux quartiles principaux sont plus petites).

Ceci veut clairement dire que *les meilleurs populations produisent les meilleurs élites*, mais aussi soulève un point de fonctionnement de l'algorithme : plus d'élitisme ($\equiv e$ plus grand) implique qu'il y a moins de croisement dans la population (car il faut juste re-créer $(100 - e)\%$ de la population) : les opérateurs de croisement et mutation sont appliqués moins souvent, à moins d'individus. Donc *moins les opérateurs sont appliqués, meilleures sont les solutions* ; ceci est peut-être indicatif que les opérateurs désignés sont trop destructifs ?

Les tendances observées ($e = 33\% \leq e = 20\% \leq e = 5\%$) sont maintenues pendant toute la durée d'une simulation particulière[7]. Les valeurs de l'évolution des valeurs médianes d'adéquation dans une population spécifique, de la génération 100 à la génération 500, sont montrées dans les Fig. 60 (population de taille 200) et Fig. 61 (population de taille 400)

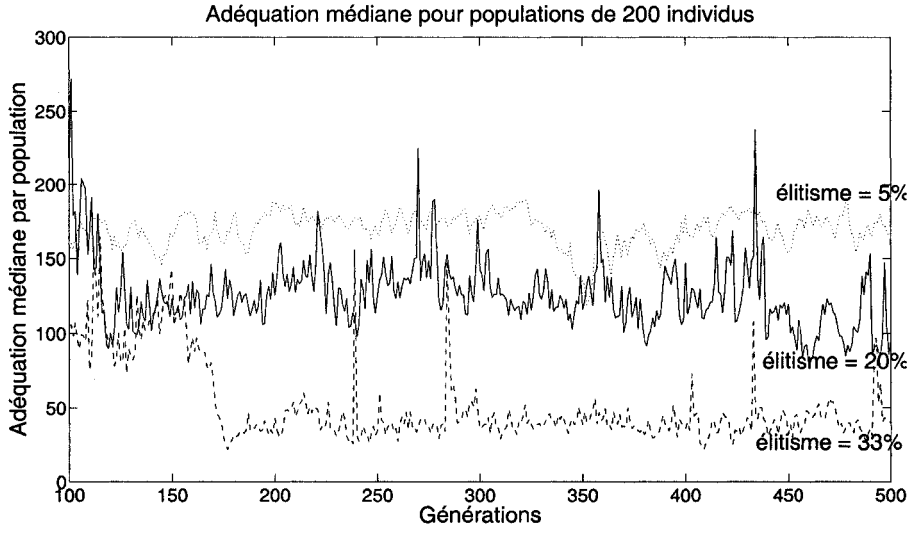


Figure 60 Médiane d'adéquation pour une population de taille 200

Je résume ici certaines autres caractéristiques observées durant l'exécution de ces expériences :

Diversité structurelle des populations Une des caractéristiques recherchées dans la population est d'avoir la capacité « d'échapper » aux minima locaux. Une façon d'assurer ceci est de maintenir une population aussi diverse que possible, minimisant ainsi les chances que la population au complet soit « attrapée » dans une région sous-optimale de l'espace d'adéquation.

La diversité structurelle de la population peut être approximée par la proportion d'individus uniques dans celle-ci ; ces valeurs sont montrées aux diagrammes en boîte de la

[7] Le même comportement qualitatif a été observé pour toutes les 30 instances

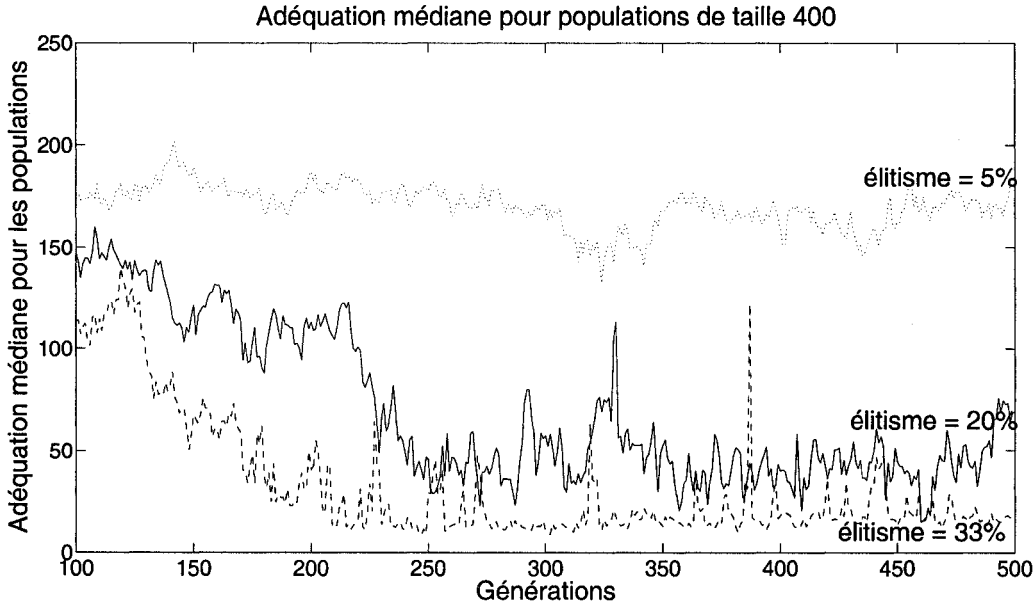


Figure 61 Médiane d'adéquation pour une population de taille 400

Fig. 62. Les valeurs associées sont au Tableau XII ; l'observation principale à faire est que

Tableau XII

Individus uniques (moyennes et écarts types) par groupe

(n, e)	Ind. unique (%)	(n, e)	Ind. unique (%)	(n, e)	Ind. unique (%)
$(200, 5\%)$	94.47 ± 4.33	$(200, 20\%)$	94.83 ± 5.73	$(200, 33\%)$	93.50 ± 9.75
$(400, 5\%)$	92.49 ± 6.21	$(400, 20\%)$	97.25 ± 2.83	$(400, 33\%)$	94.28 ± 7.44

tous les types de population sont (a) également diversifiés (structurellement parlant)[8] et, (b) très diverses (toutes les valeurs sont clairement au dessus de 90%) ; ceci témoigne, intuitivement, d'une bonne exploration de l'espace de possibilités de la part de l'algorithme.

[8] La signification statistique de l'hypothèse nulle de non-différence fût déterminée par les ANOVAS non-paramètriques de Kruskal-Wallis à $p = 0,01$

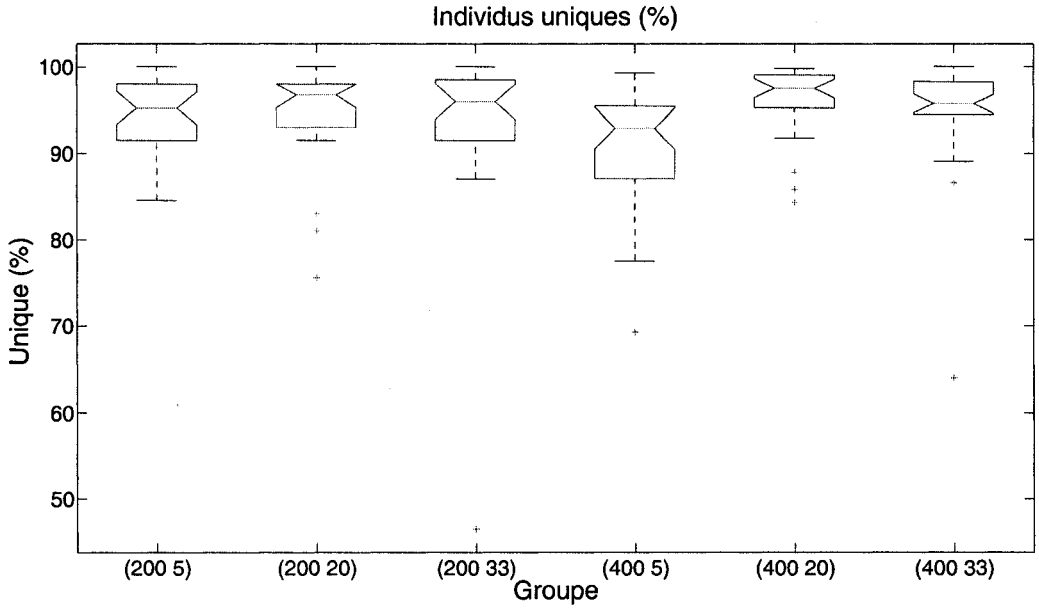


Figure 62 Pourcentage d'individus uniques dans les populations

Chaque *groupe* contient les valeurs d'individus uniques des 30 réplications ; les groupes sont présentés sous forme (taille population, élitisme).

Longueur des génomes générés par l'algorithme Les longueurs moyennes et médianes des populations sont reportées ici sous forme de diagrammes en boîte (Figs. 63 et 64, respectivement).

Nous remarquons que moins il y a d'élitisme, plus le génome est petit (donc, la moyenne de longueur des génomes des populations à 5% d'élitisme est plus petite que celle des populations à 20%, qui est elle-même plus petite que la moyenne pour les populations à 33%). En reprenant un argument présenté ci-haut, moins d'élitisme ($\equiv e$ plus petit) implique que il y a plus de croisement dans la population (car il faut re-créer $(100 - e)\%$ de la population) : les opérateurs de croisement et mutation sont appliqués plus souvent, à un nombre plus élevé d'individus. Donc l'algorithme laissé a lui seul (sans *protéger* les individus par l'élitisme) paraît produire de petits génomes. Discuter de cette caractéristique serait intéressant dans une investigation suivant cette thèse.

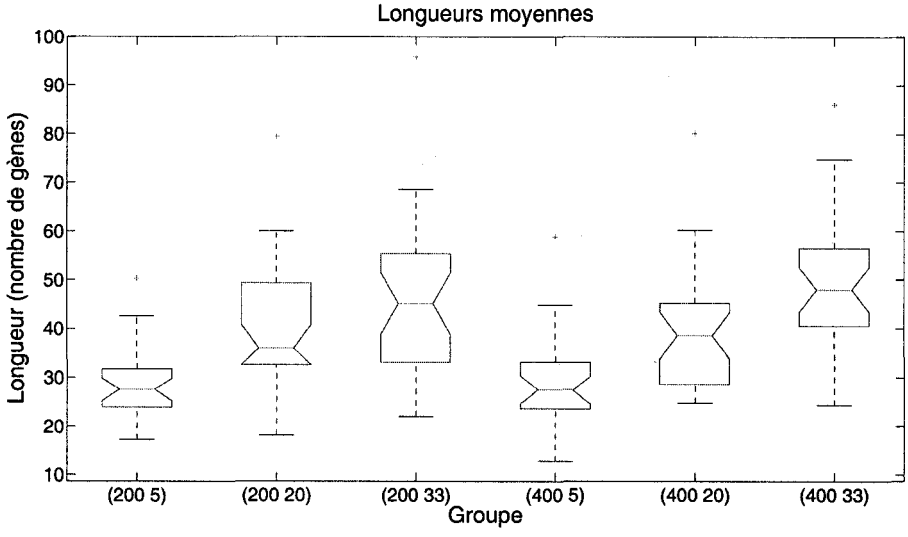


Figure 63 Moyenne des longueurs, par groupe d'expériences

Chaque *groupe* contient les moyennes des longueurs des 30 réplications ; les groupes sont présentés sous forme (taille population, élitisme).

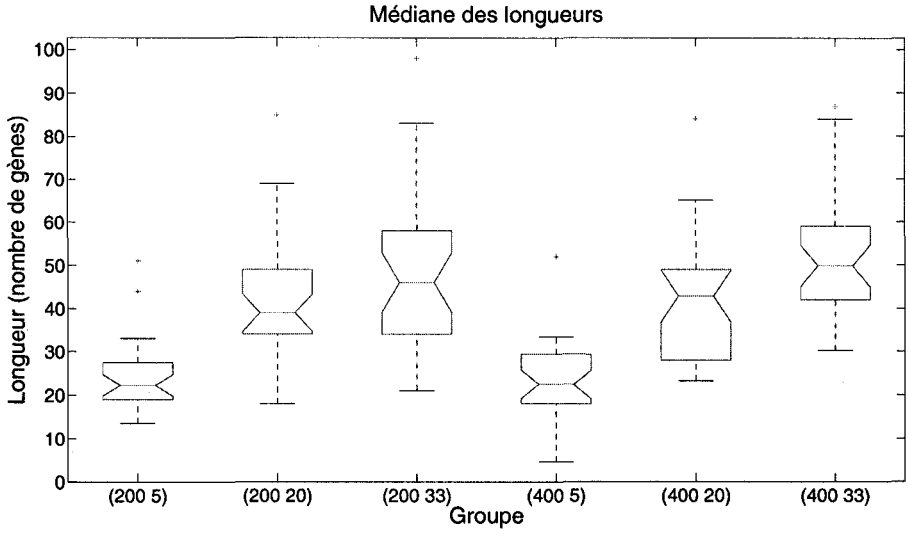


Figure 64 Médiane des longueurs, par groupe d'expériences

Chaque *groupe* contient les médianes des longueurs des 30 réplications ; les groupes sont présentés sous forme (taille population, élitisme).

4.2.2 Deuxième expérience

La deuxième expérience consiste à l'approximation du processus de développement formé par la figure 65 ; cette figure est l'instantané de l'interprétation de la grammaire $\mathbf{G}_2$

$$\mathbf{G}_2 :: \begin{cases} \omega = X \\ r_1 : F \longrightarrow FF \\ r_2 : X \longrightarrow F - [[X] + X] + F[+FX] - X \end{cases}$$

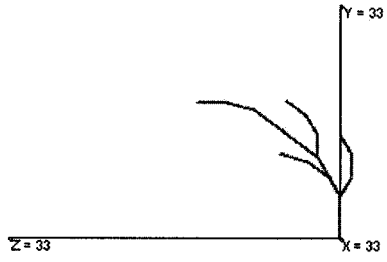


Figure 65 Une (autre) figure simple

Deux (2) itérations de la grammaire $\mathbf{G}_2$ produisent un objet 3D ; projeté selon la convention présentée à la Fig. 25, avec $d = 1$, $w = 100$, $h = 100$, $F = 1$, $B = 1$

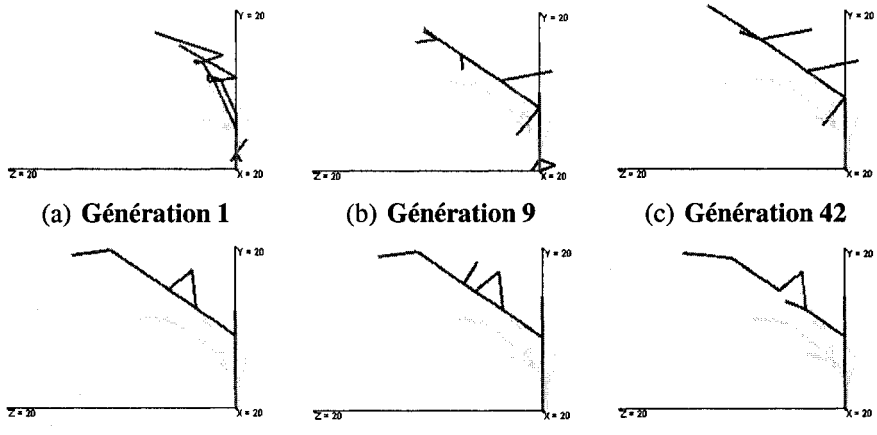
L'exécution de l'algorithme (500 générations) pour ce problème prends de 24 à 36 heures[9] sur une machine[10]. Pour cette raison il n'y a pas assez de données pour dériver des résultats statistiques des valeurs obtenues. Cependant, il est possible de montrer ici (et c'est ce qui est important) le comportement de l'algorithme avec un problème « plus complexe ».

Concrètement, une instance spécifique est présentée, mais toutes ont sensiblement les mêmes caractéristiques. L'algorithme a été éxécuté avec les paramètres spécifiques de

[9] La plage de variation provient du fait que c'est un algorithme non-déterministe, et donc pour une même expérience il y a une variabilité de temps d'exécution qui peut être appréciable.

[10] CPU AMD Athlon 64 3400+, 2.20 GHz, mémoire vive de 1.5 GB, exécutant Windows XP, Version 2002.

la Table X, avec un **pourcentage d'élitisme** de 20% et une **taille de la population** de 200 individus. La Fig. 66 montre les meilleures solutions générées pour l'algorithme évolutif aux générations 1 (a), 9 (b), 42 (c), 99 (d), 357 (e) et 481 (f). Chaque solution a (en arrière plan) la figure cible, donnant ainsi une indication visuelle de sa qualité.



(a) **Génération 1** (b) **Génération 9** (c) **Génération 42**

(d) **Génération 99** (e) **Génération 357** (f) **Génération 481**

Figure 66 Résultats approximation Fig. 65

a. Assez rapidement (gén. 9) la meilleure solution s'ajuste aux caractéristiques « grossières » de la cible : branches à gauche, pas de branches à droite, « petit » tronc. L'algorithme exhibe, encore une fois, une phase d'exploration courte et efficace.

b. Ensuite, nous observons une longue phase d'exploration. Mais les solutions semblent « figées » à ce qui a été découvert pendant la phase d'exploration : la grande branche ne disparaît jamais, il n'y a pas de changements importants dans les solutions, même si, à l'oeil nu, elles ne sont par très semblables à la cible.

c. Il semble que l'algorithme a de la difficulté à générer les branches « courbées ». D'où cela provient-il ?

   (a) de la fonction d'adéquation ?

   (b) de la difficulté du problème (deux itérations, donc explosion exponentielle des possibilités de solution) ?

(c) de la difficulté de la figure (trop courbée) ?

4.2.3 Sommaire

Ayant fait ces deux expériences de façon exhaustive, cela m'a permis de mieux comprendre le comportement de l'algorithme et, en même temps, de choisir des paramètres convenant à la tâche pour laquelle celui-ci est conçu. Concrètement, les point suivants ont été observés :

a. **La meilleure performance est obtenue avec les populations où l'élitisme est** $e = 33\%$. Pour les expériences variant le nombre d'individus ($n \in [200, 400]$) j'ai montré que, statistiquement, la performance n'était pas différente. Donc la paire $(n, e) = (200, 33\%)$ est aussi performante que la paire $(400, 33\%)$, mais prends moins de temps de calcul (car il y a moins d'individus à gérer) ; c'est alors le choix que je fais ici.

   Cependant, il est clair que ces résultats doivent être vus comme très préliminaires ; il faudrait tester avec $n \in \{50, 100\}$ (en plus des $n \in \{200, 400\}$ déjà testés) et $e \in \{0\%, 10\%, 40\%, 50\%\}$ (en plus des $e \in \{5\%, 20\%, 33\%\}$ déjà testés) pour voir jusqu'où les résultats présentés sont valables. À présent, je ne sais pas pas si la valeur d'adéquation continuera de s'améliorer quand e augmente, jusqu'à quelle valeur e_{max}, et je ne sais pas si avec moins d'individus les performances ne sont pas équivalentes (par exemple).

b. Un problème qui est difficile à résoudre est le fait que, avec les mesures prises dans ces expériences, il n'est pas possible de savoir si l'algorithme fait une *bonne* exploration de l'espace de recherche. Avec les expériences simples présentées, j'ai déjà rencontré des inconvénients quand le problème se complique légèrement (voir problème 2) ; est-ce que l'effet observé est dû aux minima locaux, ou à la fonction d'adéquation choisie ? Comme je l'ai dit, pour l'instant, il n'est pas possible de donner une réponse à ces questions.

c. La proportion d'individus uniques dans chacune de ces populations (présentée à la Fig. 62) est élevée ; ceci peut être interprété comme une bonne caractéristique de l'algorithme, qui maintient des individus différents tout au long du temps de calcul.

d. J'ai montré (page 107) comment, avec $e = 5\%$, les populations n'ont pas de bonnes élites (leur meilleur individu n'est pas bon) ; par contre, c'est à 33% d'élitisme que les meilleures solutions se trouvent. Ceci indique (probablement) que l'opérateur de croisement « détruit » les bonnes solutions. Ceci est une question ouverte de recherche, et peut en fait être équivalent au problème avec le même opérateur qui se trouve avec la méthode de programmation génétique.

e. Ces expériences m'ont aussi permis d'évaluer combien de temps prends une expérience, et alors de décider quel type d'expériences pourraient être faites en « temps raisonnable ». C'est pour cela que j'ai décidé de faire des expériences de 500 générations, même si un résultat *parfait* (adéquation = 0) n'avait pas été atteint.

Les paramètres choisis sont résumés au Tableau XIII.

Tableau XIII

Paramètres choisis pour les expériences

Simulation				**Individu**	
Taux de mutation m	0,02	Taille population n	200	Max. symb. dérivables	3
Taux de croisement c	1	Max. générations	500	Gènes par génome	$\in [1, 100]$
Taux d'élitisme e	33%				

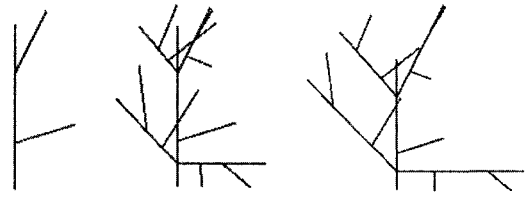
4.3 Deux expériences typiques

Ayant justifié la sélection des paramètres pour les simulations, je présente ici deux résultats typiques des expériences :

- En 4.3.1, une grammaire est développée, et des images instantanées sont prises à trois différents moments de temps, mais du même point de vue.
- Un ensemble de figures est formé des images instantanées prises de trois différents points de vue, à trois instants de temps différents (4.3.2).

4.3.1 Un processus de développement purement temporel

Pour cette expérience, le processus de développement cible consiste en trois (3) instantanés de (l'interprétation 3D) de la grammaire G_2 (Fig. 67). Les trois instantanés ont été pris aux



(a) $it1$ (b) $it2$ (c) $it3$

Figure 67 Processus de développement cible (I)

itérations $i = 1, 2$, et 3 de G_2, **toujours du même point de vue** p_1 ; p_1 est défini comme suit (voir système de coordonnées à la Fig. 25, page 45, et Déf. 3.1, page 45) :

a. p, position de la caméra, $p = (50, 5, 0)$

b. c, « centre d'intérêt », $c = (0, 5, 0)$

c. Les plans avant et arrière de découpage sont $F = 50, B = 50$

La Fig. 67(a) corresponds à l'itération $i = 1$, Fig. 67(b) à $i = 2$, et Fig. 67(c) à $i = 3$.
Il y a certaines caractéristiques présentes dans ces figures que les solutions candidates devraient être capables d'identifier ; celles-ci incluent :

a. La figure représentant l'itération 1 a seulement deux branches, plus un tronc central. Les figures représentant les itérations 2 et 3 ont plus de branches.

b. Ayant un nombre relativement élevé de branches, les figures de l'itération 2 et 3 se distinguent dans le sens que les branches à l'itération 3 sont plus longues, et plus uniformément distribuées que pour l'itération 2. La figure cible de l'itération 2 est plus « compacte » et ses branches sont principalement orientées vers la « droite » de la figure.

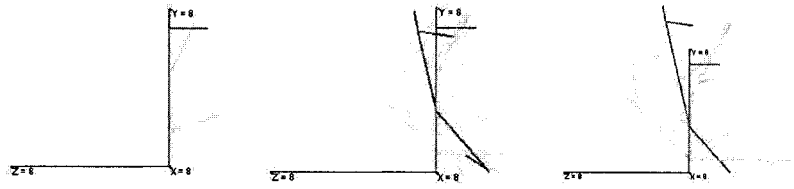
c. Toutes les branches dans les trois figures sont orientées vers le haut, avec l'exception de deux petites branches secondaires à la droite des figures aux itérations 2 et 3.

Les résultats pour cette expérience sont présentés à la Fig. 68 ; chaque ligne d'images présente une génération différente de l'algorithme. Trois générations ont été choisies : 1 (Fig. 68(a-c)), 24 (Fig. 68(d-f)), et 411 (Fig. 68(g-i)). Chaque colonne correspond à une itération de la solution candidate ; les figures cibles sont montrées comme fond des solutions candidates.

La meilleure candidate pour la première génération (Fig. 68(a-c)) est issue de l'initialisation au hasard de la population (et donc l'algorithme n'a pas procédé à la sélection génétique et croisement des individus). Dans ce sens, cette génération est montrée seulement comme référence et ses figures n'adhèrent à aucune caractéristique de la cible.

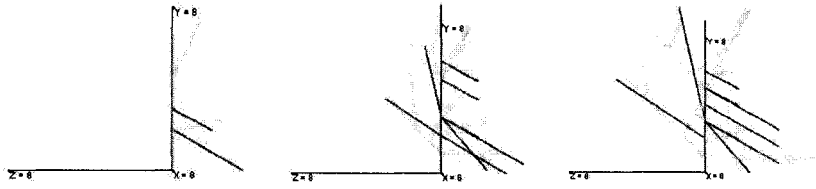
À la génération 24 (Fig. 68(d-f)) la solution candidate s'exprime déjà avec plus de branches : l'itération 1 a deux branches (comme la cible elle-même) et, dans les itérations 2 et 3 les branches de la candidate sont plus nombreuses sur le côté droit (encore une fois, comme dans les figures cibles elle-mêmes). Cependant, en général, les « branches candidates » ne suivent pas le même sens que les branches cibles (voir, par exemple, la partie droite des itérations 2 et 3).

À la génération 411 (Fig. 68(g-i), la meilleure solution obtenue avant la génération 500) les branches candidates suivent mieux le sens et la direction des branches cibles, ainsi que leur nombre ; de façon esthétique, l'algorithme propose de meilleures candidates quand le nombre de générations augmente.



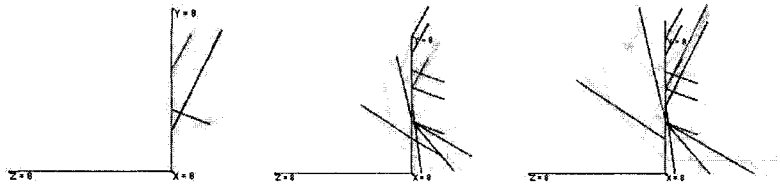
(a) It. 1 (b) It. 2 (c) It. 3

[Génération 1]. Adéquation = 68.26



(d) It. 1 (e) It. 2 (f) It. 3

[Génération 24]. Adéquation pour la génération = 47.54



(g) It. 1 (h) It. 2 (i) It. 3

[Génération 411]. Adéquation = 37.14

Figure 68 Résultats expérience I

Nous pouvons noter, cependant, comment les caractéristiques *morphologiques* des figures cible ne sont pas « mimées » par les solutions candidates ; par exemple, dans la partie inférieure droite des itérations 2 et 3 (Figs. 67(b) et (c)), nous pouvons voir une branche secondaire qui est orthogonale par rapport au tronc, avec deux branches tertiaires allant directement vers le bas. Cette partie des figures est approximée par un ensemble de quatre branches allant vers le bas dans la proposition de la génération 411.

Ceci est fort probablement causé par la façon dont la notion de (di)similarité entre deux images à été choisie (en 4.1.3, page 99) : c'est une mesure « point-par-point » qui ne prend

pas en compte aucune des caractéristiques morphologiques. L'algorithme, en utilisant cette mesure de similarité, génére des figures qui la minimisent, et alors ne prends en compte aucun autre facteur.

4.3.2 Un processus de développement spatio-temporel

Dans cette expérience, nous avons utilisé la même grammaire L-System utilisée en 4.3.1 (page 116) pour générer 9 figures : at $t = 1, 2, 3$, de trois points de vue différents p_1, p_2, p_3. Ces figures sont présentées à la Fig. 69, où les lignes correspondent à des itérations et les colonnes représentent des points de vue. p_1, p_2, et p_3 sont définis de la façon suivante (système de coordonnées de la Fig. 25, page 45, et Déf. 3.1, page 45) : pour les trois

a. c, « centre d'intérêt », $c = (0, 5, 0)$

b. Les plans avant et arrière de découpage sont $F = 50, B = 50$

et l'observateur est positionné tel que spécifié au Tableau XIV.

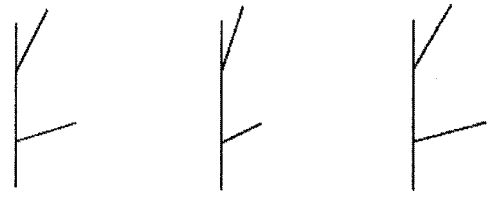
Tableau XIV

Positions de l'observateur, expérience II

	p_1	p_2	p_3
position de la caméra	$(50, 5, 0)$	$(50, 5, 25)$	$(50, 5, -25)$

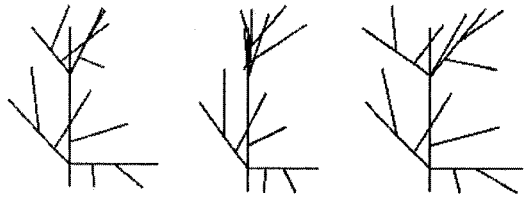
Ayant été générées par la même grammaire que dans 4.3.1, les caractéristiques présentes dans la cible sont les mêmes qu'exprimées avant (surtout par rapport au nombre de branches et leur disposition). Le défi pour l'algorithme dans cette expérience est d'intégrer toute l'information temporelle et géométrique dans un seul modèle.

La meilleure solution candidate a été trouvée à la génération 446, et est montrée à la Fig. 70. Encore une fois, il peut être facilement observé comment l'algorithme réussit à trouver un modèle ayant une interprétation se rapprochant de certaines caractéristiques de



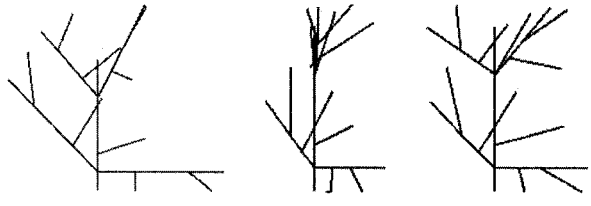
(a) p_1 (b) p_2 (c) p_3

[Itération 1] Points de vue : p_1 en (a), p_2 en (b), et p_3 en (c)



(d) (e) (f)

[Itération 2] Points de vue : p_1 en (d), p_2 en (e), et p_3 en (f)



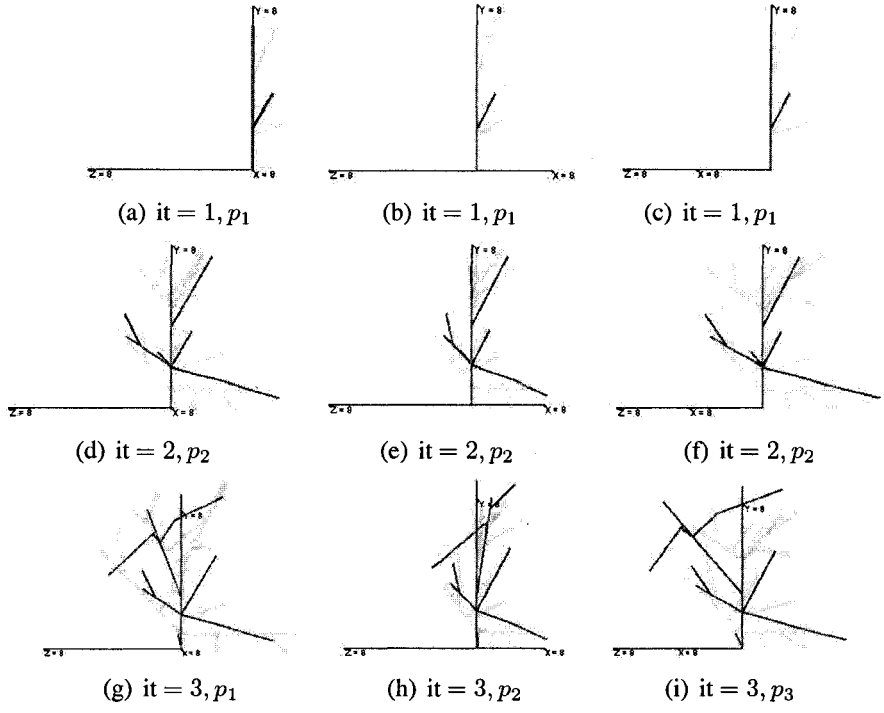
(g) (h) (i)

[Itération 3] Points de vue : p_1 en (g), p_2 en (h), et p_3 en (i)

Figure 69 Processus développement cible (II)

Les points de vue sont définis au Tableau XIV

la cible (comme le nombre de branches et leur disposition) ; cependant, il est encore plus évident ici que les caractéristiques morphologiques ne sont pas prises en compte quand les nouvelles solutions sont évaluées : la Fig. 70 montre une candidate qui remplit l'espace comme le fait la cible, mais avec une façon différente d'arranger ses branches.



(a) it $= 1, p_1$ (b) it $= 1, p_1$ (c) it $= 1, p_1$

(d) it $= 2, p_2$ (e) it $= 2, p_2$ (f) it $= 2, p_2$

(g) it $= 3, p_1$ (h) it $= 3, p_2$ (i) it $= 3, p_3$

Figure 70 Résultats pour l'expérience II

Génération 446, valeur d'adéquation 85.9 (Les figures cibles sont montrées dans le fond de chaque solution.)

Sommaire

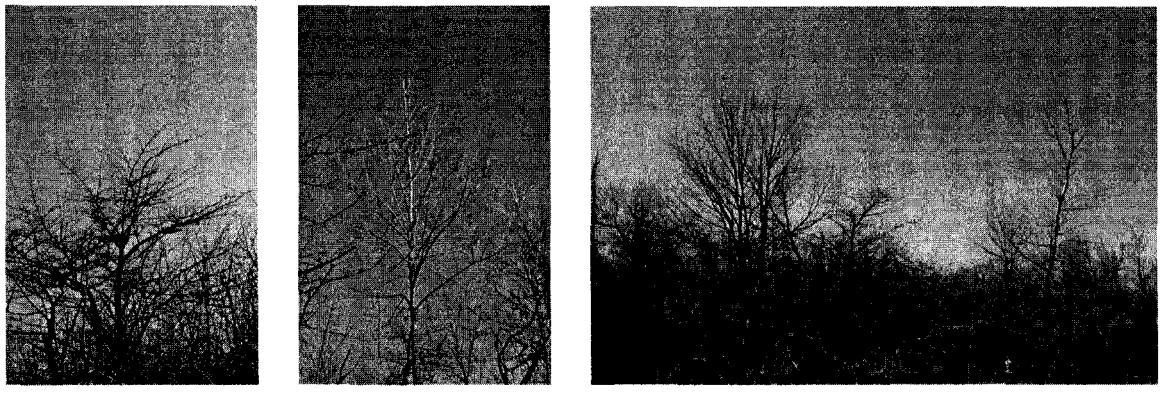
Dans ce chapitre les résultats des expérimentations menées à bien pour comprendre le comportement de l'algorithme ont été présentés ; une discussion par rapport à ceux-ci et au travail en général de la thèse forme le centre du prochain chapitre.

DISCUSSION - PERSPECTIVES FUTURES

Dans ce chapitre je discute des résultats obtenus lors de l'expérimentation, dans un premier temps, pour ensuite commenter sur les avenues futures pour cette recherche.

Résultats obtenus

Mon objectif dans cette thèse était d'explorer la possibilité de synthétiser un modèle fidèle pour le squelette d'une plante (ou d'un arbre) à partir d'un ensemble d'instantanés représentant un processus de développement ; la résolution de ce problème a été divisé en plusieurs parties, la première étant l'**extraction du squelette des arbres présents dans les instantanés**. Cette tâche n'a pas été abordée comme partie du travail présenté ici, car c'est un travail de recherche en lui-même, avec ses difficultés particulières ; je présente, à la Fig. 71, trois cas qui exemplifient ces difficultés.



(a) (b) (c)

Figure 71 Exemples de segmentation difficile

Ces figures représentent des arbres à feuillage caduc, simplifiant ainsi une grande partie du travail : le squelette est libre de feuilles, et donc est, théoriquement, plus facilement identifiable sur chacune des images ; cependant, d'autres difficultés restent : les figures (a) et (b) montrent chacune un seul arbre, mais dont les branches sont *mélangées* avec d'autres

plantes ne correspondant pas à la structure cherchée. Dans le cas de (c), plusieurs arbres sont observables, dont certains d'entre eux sont pratiquement « perdus » dans la végétation qui sert d'arrière-plan.

Une méthode pour obtenir le squelette d'un arbre à feuillage caduc a été implanté dans le cours des recherches au sein du **LIVIA** ; son intégration avec la méthode proposée ici est un pas très important qui doit survenir dans le futur immédiat.

Entretemps, dans cette thèse, la recherche s'est concentrée sur la résolution du problème inverse pour modèles fractals, spécifiquement des modèles fractals désignés pour représenter des figures de plantes et arbres. Les résultats produits par ce travail sont résumés comme suit :

Présentation d'un formalisme pour les modèles de squelettes d'arbres. Une partie importante du travail de réflexion qui a débuté cette recherche a été dédiée à choisir le formalisme le plus adéquat pour le travail à effectuer ; tel qu'argumenté au début du chapitre 3, ce formalisme est le L-System. Cependant, dans la section 3.2.2 il est discuté de la nécessité de définir un sous-ensemble du formalisme des L-Systems pour représenter *spécifiquement* des arbres naturels. Cette discussion se base sur le fait que Lindenmayer, en introduisant les L-Systems à crochets comme outil pour représenter des structures à branches (Lindenmayer, 1968), dit « *toute structure de branches peut être spécifié par un L-System à crochets* » ; mais la partie réciproque (*trouver un formalisme dont toutes les instances représentent des arbres naturels*) n'a pas de solution dans la littérature. Cette thèse en propose une, en introduisant les opérateurs L-System appelés **B**, **Y** et **T** ; des représentations des définitions sont reprises ici : Fig. 72 pour **B**, Fig. 73 pour **Y** et Fig. 74 pour **T**. Les détails sont présentés en section 3.2.2 (page 55) de ce document.

Le point principal fait ici est que **la croissance des arbres naturels peut en fait être représentée par une grammaire L-System combinant ces trois opérateurs avec un axiome (approprié à la forme naturelle à représenter)**. Ce formalisme n'est, pour l'ins-

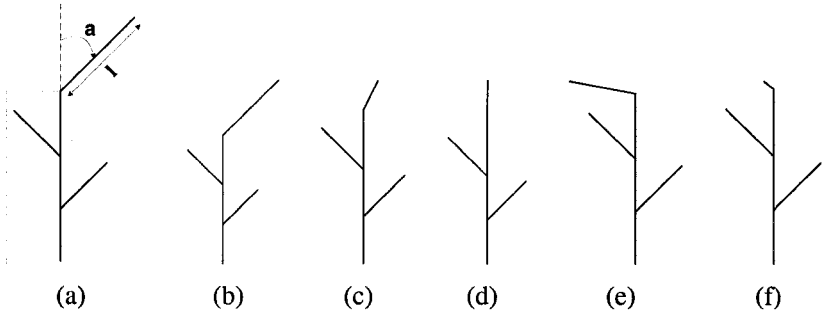


Figure 72 Opérateur B (a) avec des exemples

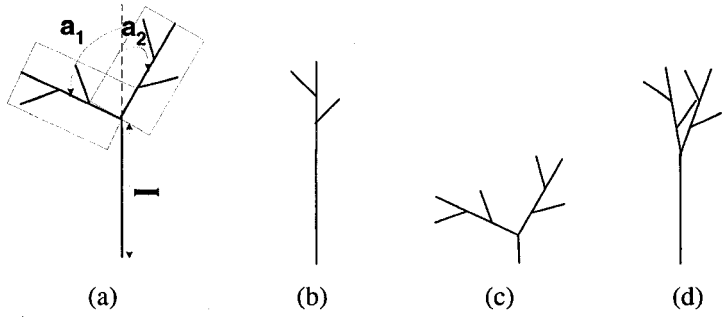


Figure 73 Opérateur Y (a) avec des exemples

tant, qu'une *proposition* fondée sur l'observation des figures d'arbres et l'étude des modèles fractals existants. Il serait sans doute important de pouvoir quantifier l'exactitude et la justesse de celle-ci, en répondant à la (double) question : *est-ce que ce formalisme représente toutes les espèces de plantes ?* et *est-ce que toutes les espèces de plantes sont représentées par ce formalisme ?*.

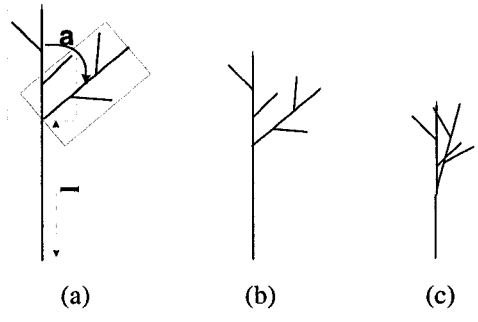


Figure 74 Opérateur T (a) avec des exemples

Le premier pas vers cet objectif serait de trouver un modèle (botanique) déjà existant représentant *toutes* les formes trouvées dans les arbres naturels : une proposition pour cet ensemble est détaillée dans ce document (modèle **HO**, section 1.2, page 17) ; une façon de procéder pour essayer de montrer l'équivalence entre le formalisme **HO** et le formalisme proposé dans cette thèse consisterait à

a. Générer des processus de développement cible pour chacun des 23 modèles idéaux de **HO**

b. Pour chacun de ces processus de développement, appliquer la méthode présentée dans cette thèse pour synthétiser un modèle pour chacun d'entre eux. Ainsi serait démontré l'affirmation *il existe un modèle « thèse » pour tous les modèles* **HO**.

La partie de l'affirmation disant *tous les modèles « thèse » sont des modèles* **HO** est plus délicate à prouver ; en ce moment, je n'ai pas de proposition concernant ce point, sauf de reprendre sérieusement l'ouvrage présentant **HO** (Hallé et al., 1978), et réfléchir à leurs propositions sous la lumière de la méthode présentée dans cette thèse.

Conception d'une méthode pour approximer des processus de développement. Un algorithme pour construire un L-System répondant aux critères d'entrée a été proposé, implanté, et testé dans ce travail. En utilisant le sous-ensemble défini par **B**, **Y**, et **T**, un algorithme basé sur l'algorithme génétique fait l'assemblage de règles suivant ces schémas avec un axiome. La conception de cet algorithme a impliqué l'étude des caractéristiques précises du problème pour ainsi spécifier une façon de procéder en accord avec l'objectif ; ceci implique, entre autres, la génération des grammaires ayant des interprétations « intéressantes » pour le processus cible.

L'algorithme a été exécuté sur un ensemble de tests ; les résultats présentés au chap. 4 montrent que l'algorithme explore l'espace de solutions d'une façon satisfaisante : d'abord une période rapide d'exploration s'installe (longue d'un nombre inférieur à 10% des générations totales), suivie d'une longue période d'exploitation. Cependant on remarque que

sa performance se dégrade avec des figures plus *complexes* (voir 4.2.2, page 112) ; plus d'expériences sont sans doute requises pour comprendre les raisons de ce comportement, mais je pense que cela est fortement lié au choix de la fonction d'adéquation utilisée. La mesure de similarité définie, basée sur une simple distance point-à-point de deux figures, ne donne pas une idée précise de la distance entre deux définitions topologiques. Ce que je voudrais c'est pouvoir décrire de façon fidèle un arbre, où *fidèle* se réfère au sens de proximité visuelle propre à un être humain. La question devient alors *comment* un être humain compare des figures d'arbres ? L'idée de base derrière cette motivation est de pouvoir synthétiser une métrique qui traduise l'exactitude et la subjectivité du système visuel humain. Une option est présentée à l'Annexe III.

Points à développer

Après avoir présenté les résultats, je voudrais parler des points sur lesquels les développements futurs de ce travail devraient se concentrer ; à ce sujet, il y aurait des points à attaquer à courte, moyenne, et longue échéances. Les échelles temporelles correspondent à des points qui sont (a) une validation de cet algorithme (courte échéance), (b) des extensions aux idées (moyenne échéance) et (c) de grands changements dans la façon de procéder (longue échéance).

Les points à résoudre à courte échéance seraient :

Validation avec plus d'expériences des résultats statistiques obtenus. Les résultats présentés en 4.2.1 (page 103 et suivantes) nous donnent un aperçu des capacités de l'algorithme ; cependant, il serait important d'avoir une vue d'ensemble plus complète, en attaquant principalement deux points :

a. En lui proposant des processus de développement plus complexes (du point de vue temporel et aussi géométrique)

b. En faisant un plan d'expériences pour balayer tout l'espace des paramètres de l'algorithme : rang de grandeur pour un génome, valeurs de mutation, croisement, élitisme, et nombre d'individus dans la population, notamment.

Ceci, dans la pratique, ne sera possible qu'en augmentant d'un cran la vitesse de calcul de l'algorithme ; mon avis est qu'il faudrait passer des 12 heures (environ) pour l'expérience la plus simple, à environ 2 heures. De cette façon, environ 10 expériences par jour pourraient être exécutées. Les deux points suivants s'adressent à ce problème de « vitesse ».

Temps d'exécution de l'algorithme. Tout au long de ce travail, ma préoccupation principale n'était pas le *temps* que l'algorithme prends à faire la reconstruction, mais plutôt d'obtenir une méthode pour résoudre le problème. Cependant, le temps d'exécution est aussi important, car il limite (ou permet) l'utilisation pratique de celle-ci. À présent, le temps d'exécution pour les 500 générations de 200 individus pour des expériences très réduites (une image à $t = 1$ étant la seule image dans un processus de développement cible) prend environ 12 heures sur un ordinateur moderne[11]. Ceci est trop lent, et doit vraiment être amélioré.

Implanter l'environnement dans un langage plus moderne. Cet environnement a été programmé en Visual Basic (motivé largement par des raisons de facilité de programmation de l'environnement graphique), avec des modules en **MATLAB** ; je pense cependant qu'il serait profitable d'investir du temps à le reprogrammer en Java, pour pouvoir l'exécuter comme partie d'une page Web ou pour pouvoir l'exécuter dans n'importe quel système d'exploitation sans avoir à le recompiler. Une autre alternative serait de le reprogrammer pour qu'il fonctionne comme un programme parallèle, pour ainsi pouvoir lancer des fils d'exécution légers se communicant entre eux. Je pense que ceci aiderait beaucoup, en soi-même, à améliorer la vitesse du système.

À moyenne échéance, je considère que les points suivants doivent être attaqués :

[11] CPU AMD Athlon 64 3400+, 2.20 GHz, mémoire vive de 1.5 GB, exécutant Windows XP, Version 2002.

L'évaluation des solutions *grammaticales*. Dans ce travail j'ai utilisé une mesure de similarité comme seule indicatrice de la justesse d'une solution : si les images générées par l'algorithme sont proches des images cibles, alors la justesse de la solution est *bonne* ; il serait cependant préférable d'avoir une façon de mesurer le succès d'une reconstruction : *est-ce que le modèle proposé se rapproche du modèle cible ?*.

Bien sûr, cette comparaison est seulement possible si le modèle cible est connu (ce qui, en principe, est rarement le cas, sauf dans des cas comme dans ce travail, où l'algorithme est évalué avec des modèles artificiels). Une proposition dans ce sens consiste à générer n projections (images instantanées) de la figure cible 3D, mais utiliser seulement $n_{tr} < n$ comme entrée à l'algorithme pour générer le modèle. Le succès de cette reconstruction consisterait à mesurer la distance de la figure cible à la figure candidate *sur les autres* $(n - n_{tr})$ projections. Cette idée est utilisée dans le domaine de la classification automatique, où un classifieur est entraîné sur une base de données d'entraînement (taille n_{tr}), et sa performance mesurée comme l'*erreur de généralisation* : l'erreur exprimé sur un problème différent de celui sur lequel il a été entraîné (de taille $n - n_{tr}$).

Spécification de restrictions physiques dans les solutions. Dans les schémas proposés comme base pour une architecture de plante (schémas des opérateurs **B**, **Y**, et **T**) les restrictions physiques des plants naturels ne sont pas exprimées : par exemple, nous savons (par l'observation) que, dans la nature, les branches les plus éloignées du tronc et du sol sont plus petites et moins lourdes que les branches se trouvant plus près du tronc. En insérant ce type de restrictions, l'espace de recherche se rétrécit considérablement, et l'algorithme serait alors plus efficace (car il ne prendrait pas de temps à explorer des solutions inappropriées pour le problème à résoudre).

Insérer de l'information de l'environnement (contexte) dans l'algorithme. Ce modèle, tout comme ceux proposés dans la littérature (notamment en (Prusinkiewicz et Lindenmayer, 1990)), fonctionne sous la supposition que les organismes contrôlent leur propre

développement ; comme mentionné en (Apter, 1966) (cité dans (Prusinkiewicz et Lindenmayer, 1990), page 64) cette simplification doit être acceptée si nous supposons que le modèle mathématique est limité à une plante isolée. Cependant, la forme de plants supérieurs est (largement ?) déterminée par l'environnement, les compétitions entre branches et arbres, et accidents (Zimmerman et Brown, 1971) ; il serait important de réfléchir sur ce point et repenser ce travail à la lumière de cette réflexion.

Finalement, à plus longue échéance, ma proposition est d'**injecter la connaissance de l'expert** qui fait l'expérience : une des raisons pour laquelle cette méthode prends beaucoup de temps à exécuter est parce qu'elle doit naviguer dans l'espace des axiomes et évaluer tout ce qui *paraît* intéressant. Cependant, si l'expérimentateur utilisant cette méthode est en mesure de guider la méthode concernant la *forme* finale de la plante (« *on travaille avec des plants de patates* », par exemple) alors cette information pourrait être utilisée comme direction pour guider l'algorithme. Deux façons de faire ceci sont considérées :

a. **Construire une base de données d'axiomes pour chaque espèce de plants.** Cette base de données serait l'initialisation de la population d'axiomes.

b. **Adapter les opérateurs génétiques pour générer des individus suivant une forme générique particulière.** Maintenant, si deux individus « ayant la forme de plant de patates » se croisent, le résultat ne suivra probablement pas la même forme que les parents.

Cette proposition s'inscrit dans la ligne de pensée qu'un algorithme (évolutif ou pas) doit se spécialiser à résoudre un (certain type de) problème(s) pour être vraiment performant (une petite discussion là dessus est faite à la page 86) ; notre algorithme utilise déjà beaucoup d'information a priori (provenant notamment du modèle proposé pour les arbres naturels) et la définition d'opérateurs génétiques plus spécialisés et des points de départ plus proches des bonnes solutions rendrait l'algorithme (plus) spécifique à notre problème.

Comme paragraphe final, je voudrais commenter que le fait de (a) améliorer la vitesse d'exécution de la méthode, intégrée avec (b) une méthode de segmentation des plants naturels, constituerait un système complet avec lequel des expériences très intéressantes, pratiques, et utiles, pourraient être conduites.

ANNEXE I

Publications de l'auteur

Les résultats préliminaires obtenus dans le développement de la thèse ont été publiés dans 3 travaux principaux : premièrement, le choix et la justification du modèle le mieux adapté à la problématique à résoudre (développé au chapitre 1, pages 7 et suivantes ; aussi en 3.2.1, page 48) apparaissent dans

- Da Costa, Luis ; Landry, J.A. 2003. **On the Applicability of L-Systems and Iterated Functions Systems for Grammatical Synthesis of 3D Models**. Lecture Notes in Artificial Intelligence 2671 : 614-615 2003.

L'environnement de travail (discuté à partir de la page 50) a été présenté dans la conférence Fractals, et est référé comme

- Da Costa, Luis ; Landry, J.A. 2004. **A Convivial Visualization Environment for LSystems**. Fractal 2004 : Complexity and Fractals in Nature. Vancouver, Canada. April 4-7, 2004.

Les premiers résultats des travaux sont presentés dans deux papiers :

- Da Costa, Luis ; Landry, J.A. 2004. **Synthesis of Fractal Models for Plants and Trees : First Results**. Fractal 2004 : Complexity and Fractals in Nature. Vancouver, Canada. April 4-7, 2004.
- Da Costa, Luis and Landry, J.A. 2005. **Generating grammatical plant models with genetic algorithms**. Proceedings of the International Conference on Adaptive and Natural Computing Algorithms. Coimbra, Portugal. March 21-23. SpringerWien, New York. 230-234.

La produit complet de la thèse a été envoyé pour publication dans le journal Fractals :

- Da Costa, Luis and Landry, J.A. **An evolutionary method to solve the inverse problem for bracketed L-Systems**. Submitted on September 8th., 2006, to *Fractals* (published by World Scientific, http://www.worldscinet.com/fractals/fractals.shtml)

Un sous-produit important de l'expérimentation faite pour cette thèse s'exprime par l'application de la technique de la Programmation Génétique à différents problèmes ; en principe, je l'avais utilisée pour synthétiser une fonction de comparaison d'arbres, mais mes

efforts n'ont pas abouti de façon satisfaisante. Cependant, j'ai continué à y travailler avec mon superviseur, Jacques-André Landry, et un de ses ex-étudiants à la Maîtrise (Clément Chion). Les publications suivantes sont le fruit de cette collaboration :

- Chion, Clément, Landry, J.A. and Da Costa, Luis **A Genetic Programming Based Method for Hyperspectral Data Information Extraction : Agricultural Applications** . Submitted on October 24th., 2006, to the *IEEE Transactions on Geoscience and Remote Sensing* (submission ID=TGRS-2006-00660)
- Landry, J.A. ; Da Costa, Luis and Bernier, T. 2006. **Discriminant Feature Selection by Genetic Programming : Towards a Domain Independent Multi-Class Object Detection System**. Journal of Systemics, Cybernetics and Informatics, 3(1).
- Da Costa, Luis and Landry, J.A. 2006. **Relaxed Genetic Programming**. GECCO 2006, eds. M. Keijzer et al. (Seattle, WA, USA. July 8th-12th), pp. 937-938.
- Chion, Clément ; Da Costa, Luis and Landry, J.A. 2006. **Genetic Programming for Agricultural Purposes**. GECCO 2006, eds. M. Keijzer et al. (Seattle, WA, USA. July 8th-12th), pp. 784-789.

ANNEXE II

Documentation Environnement L-Systems

Dans ce chapitre de documentation de l'environnement de travail, je présente deux parties : premièrement la documentation pour l'usager (section AII.1) et ensuite la documentation pour le programmeur (section AII.2).

AII.1 Manuel d'usager

La modélisation, la simulation et la visualisation du développement des plantes est un domaine de recherche passionnant ; il combine des idées basées sur l'informatique (infographie, théorie des langages et simulation), ainsi que des idées venant de la Vie Artificielle et de sciences *pures*, comme biologie, mathématique et physique. En outre, « . . . *la modélisation des plantes a aussi une saveur artistique, car la beauté des plantes pose un défi constant pour créer des modèles visuellement attractifs* » (Prusinkiewicz et Lindenmayer, 1990).

Pour exploiter l'intuition et la puissance obtenue des L-Systems il est utile de travailler dans un environnement qui permettrait l'interaction simultanée avec la définition grammaticale de l'objet désigné et avec son interprétation graphique. La création la plus importante faite jusqu'à date correspond au développement de *Virtual Laboratory*, un environnement interactif pour faire des expériences en simulation (Prusinkiewicz et Lindenmayer, 1990). *Virtual Laboratory* se spécialise en applications graphiques des L-Systems, en mettant l'emphase sur la génération des fractales et la modélisation des plantes.

Pour cette thèse j'ai senti la nécessité d'avoir un environnement qui soit (a) intuitif, simple à comprendre et à naviguer, et (b) suffisamment complexe pour nous permettre de faire des bonnes visualisations et maniements de L-Systems. Ceci m'a motivé pour faire le design et l'implantation d'un environnement pour la visualisation et l'interaction avec des L-Systems. Ses caractéristiques principales sont :

- Grammaires L-System de différents types (paramétriques ou non-paramétriques, déterministes ou stochastiques, indépendantes ou sensibles au contexte) peuvent être définies.
- Les modèles sont spécifiées en 3 dimensions.
- Les structures sont montrées dans l'écran suivant une métaphore d'une caméra dans l'espace : tous les paramètres de visualisation peuvent être changés par l'usager. L'environnement envoie continuellement des *indices* concernant la position dans l'espace de la caméra.
- L'environnement a un module pour projeter ces objets en 2D (équivalent à *prendre des « photos »* (des instantanées) d'une certaine position ; ces projections peuvent alors être manipulées comme désiré. Un module de visualisation permet de manipuler les photos acquises.
- Code source à l'adresse : http://www.livia.etsmtl.ca/people/costa

Une présentation complète de l'environnement (sous forme de *manuel d'usager*) est présenté ici, avec des réflexions sur comment ce travail m'a aidé à mener à bien mes objectifs de recherche.

AII.1.1 Description de l'environnement

L'environnement est divisé en trois parties majeures : (1) le **module de description 3D**, qui permet la définition d'un objet avec un L-System (section AII.1.1.1), (2) le **module de visualisation**, qui permet la visualisation de « *instantanées* » de ces objets (les projections 2D) (section AII.1.1.2) et (3) le **module de synthèse** (section AII.1.1.3).

AII.1.1.1 Module de description 3D

L'idée de ce module est de présenter un environnement graphique et facile à utiliser pour permettre à l'usager de définir, avec des grammaires L-System, des modèles pour des objets en trois dimensions et, en même temps, de pourvoir une façon de visualiser comment

cet objet change avec le *temps*. La présentation graphique de ce module est illustré à la Fig. 75, et les différentes sections de cette présentation (marquées [1] à [8] à la Fig. 75) sont décrites ici.

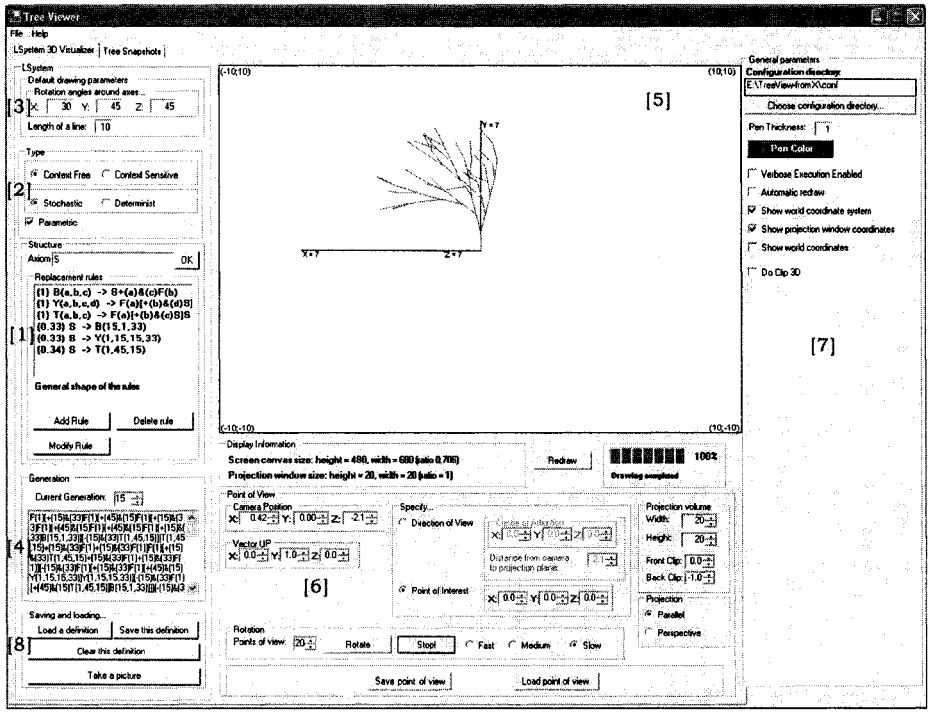


Figure 75 Module de définition 3D de l'environnement

- **Définition grammaticale** : Fig. 76 (correspondant à la Fig. 75, section [1])montre comment une description basée en L-Systems est entrée dans le système par l'usager, suivant une syntaxe standard (Prusinkiewicz et Lindenmayer, 1990). La forme de chaque règle (et de la grammaire comme un tout) est guidée par le choix du *type* de la grammaire par l'usager.
- **Choix de types de L-System** : Comme montré à la Fig. 77 (et Fig. 75, section [2]), l'usager peut choisir entre différents types de L-Systems : paramétrique ou non-

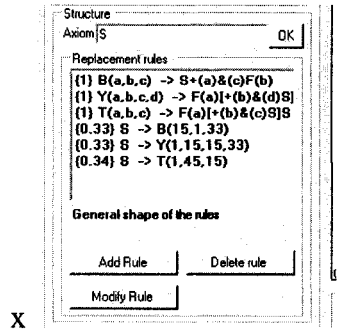


Figure 76 Description d'un objet 3D basé en L-System

paramétrique, déterministe ou stochastique et dépendante ou indépendante du contexte (Prusinkiewicz et Lindenmayer, 1990).

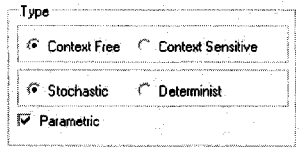
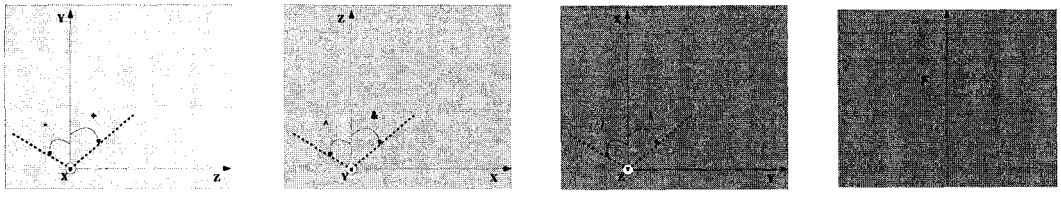


Figure 77 Types de L-Systems valides

- **Paramètres par défaut pour un L-System** : L'interprétation graphique 3D d'un L-System utilise la définition de 4 paramètres par défaut (Fig. 78, correspondant à Fig. 75, section [3]) :
  a. Angle par défaut pour les commandes + et -
  b. Angle par défaut pour les commandes \ et /
  c. Angle par défaut pour les commandes & and ∧
  d. Longueur par défaut d'une ligne (commande **F**)

  Ces paramètres sont définis par l'usager à la section [3] de la Fig. 75, montrée en détail à la Fig. 79
- **Dérivation des itérations de la grammaire** La dérivation du LString pour une itération donnée est montrée à la section [4] de la Fig. 75 (aussi montré à la Fig. 80)



(a) $+$ et $-$: rotation de (oY) à (oZ)

(b) & et $\wedge$: rotation de (oZ) à (oX)

(c) $\backslash$ et / : rotation de (oX) à (oY)

(d) F

Figure 78 Commandes avec ses paramètres par défaut

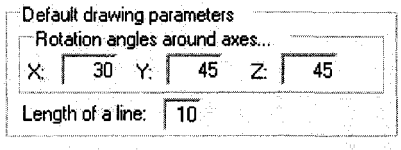


Figure 79 Paramètres par défaut d'un L-System

– **Visualisation d'une figure 3D en un espace 2D (*l'écran d'un ordinateur*)** Le LString généré est montré sur le canevas graphique (section [5] of Fig. 75, et Fig. 81) basé sur la projection guidée par la métaphore d'une *caméra* : l'idée est de représenter la figure 3D générée comme si elle était vue d'une certaine position de l'espace. Pour la métaphore de la caméra dans l'espace 3D (correspondant à un *observateur*) l'objet est supposé être à un point $(0, 0, 0)$ d'un système d'axes 3D. (Foley et al., 1990, chap. 6) L'ensemble complet de paramètres est montré à la Fig. 82, et spécifié dans l'environnement à la Fig. 75, section [6] (aussi Fig. 83).

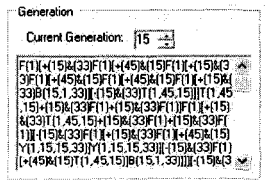


Figure 80 LString dérivé d'une grammaire

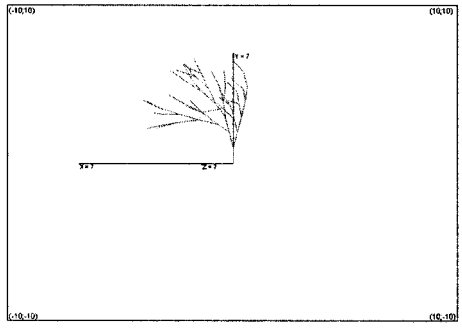


Figure 81 Canevas pour la projection

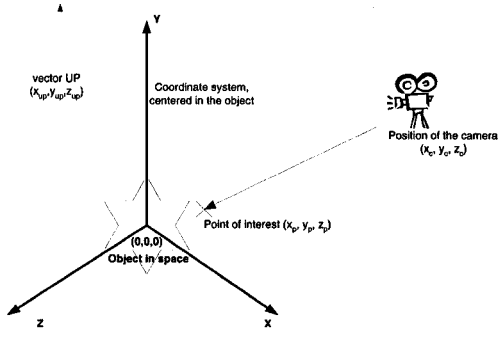


Figure 82 Métaphore d'une caméra

– **Autres paramètres** Différents paramètres, utilisées surtout pour le *débogage* et faire de la sauvegarde et récupération de données sont spécifiés à la Fig. 84 (Fig. 75, section [7]).

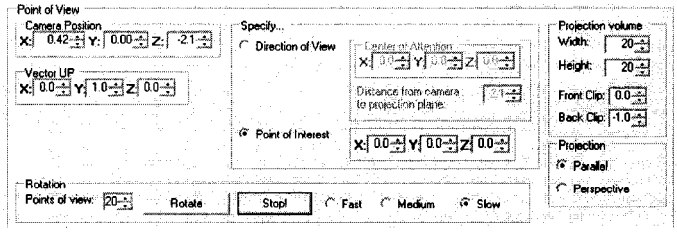


Figure 83 Position de la caméra et paramètres de visualisation 2D

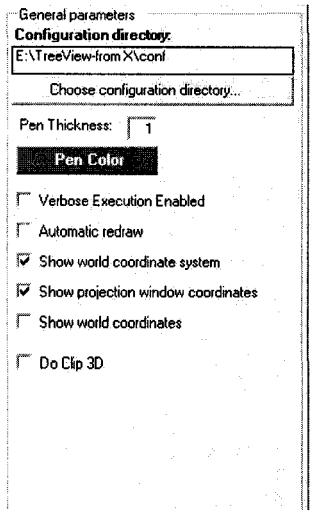


Figure 84 Autres paramètres pour l'environnement

– **Prendre des *instantanées* d'un objet** Une fonctionalité permettant de prendre des « photos » d'un objet est présentée à la Fig. 75, section [8] (aussi Fig. 85). Une description simple de la figure (la photo !) est gardée dans un fichier texte, dans un répertoire racine choisi par l'usager.

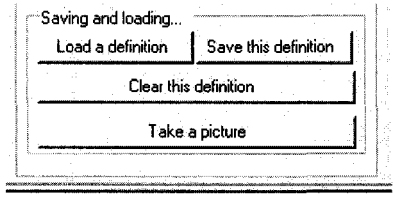


Figure 85 Description L-System de l'objet en 3D

AII.1.1.2 Module de visualisation

En utilisant un répertoire racine pour une figure 3D, ce module charge et fait la visualisation des figures 2D gardées dans celui-ci. Les parties principales de cette section sont montrées à la Fig. 86.

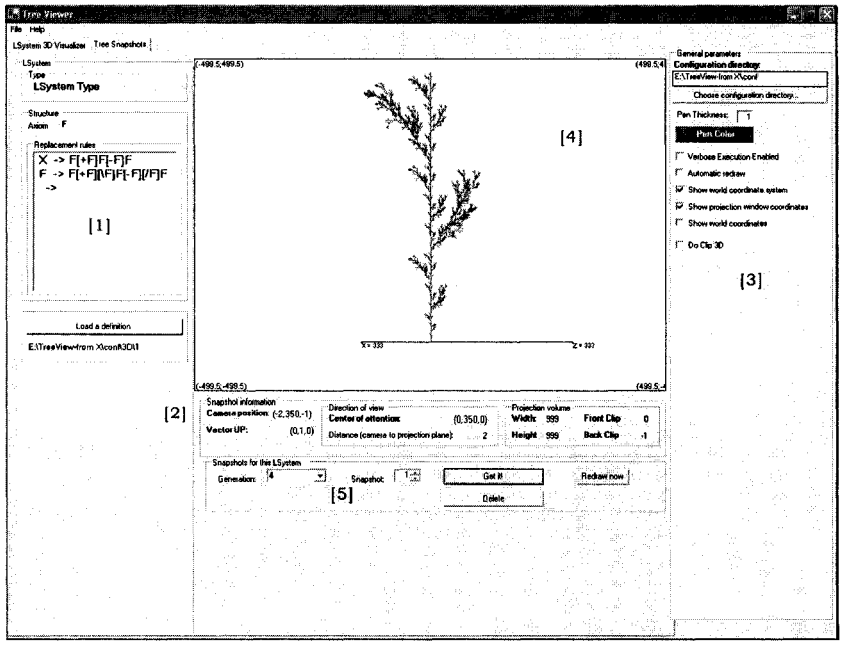


Figure 86 Environnement de visualisation

La section [1] contient la définition de L-System 3D qui a génére la photo ; la section [2] montre la position de la caméra quand la photo a été prise. Ces deux section sont « lecture seulement » : elles sont montrées seulement comme référence pour l'usager.

La section [3] contient les mêmes options que pour le module 3D (voir AII.1.1.1) et le canevas (section [4]) est défini de la même façon. Section [5] contient la génération et une identification pour l'instantanée qui est montrée.

AII.1.1.3 Module de synthèse

Fichier de configuration ; en ordre (% indique un commentaire dans le fichier) :

a. **Port de communication** pour les deux modules (TCP/IP) (pour sécurité, je prends n'importe quel port ≥ 9100

b. **Étiquette** indiquant quel type d'algorithme il faut exécuter : **ALGO LAZY** indique l'algorithme décrit dans cette thèse.

c. **Nombre de générations** pour l'algorithme.

d. **Nombre de symboles dérivables** dans les individus de la population.

e. **Probabilité de mutation d'un gène** dans les individus de la population ($\in [0, 1]$)

f. **Pourcentage d'élite** pour la population ($\in [0, 100]$)

g. **Nombre d'individus** dans la population

h. **Nombre minimum de gènes** dans les individus de la population

i. **Nombre maximum de gènes** dans les individus de la population

j. **Liste des fichiers** avec les caractéristiques des cibles. Ceci est une suite des paramètres suivants (pour chaque figure du processus cible, TOUTES ces données doivent être spécifiées) :

   (a) La liste des points sur l'écran conformant l'image

   (b) Le point de vue de l'observateur

   (c) « L'âge » de la figure (itération a laquelle elle a été obtenue)

   (d) L'histogramme des segments

   (e) L'histogramme des angles

AII.1.2 Conclusions

Le développement de cet outil simple est d'importance capitale pour atteindre les objectifs de ce travail de recherche ; il est difficile de visualiser des objets en trois dimensions, et

encore plus difficile d'imaginer ce que la projection en deux dimensions de cet objet représente. Avec cet environnement la définition grammaticale d'un objet 3D a immédiatement un sens grammatical, et l'intuition de la difficulté d'une reconstruction peut être évaluée.

Dans le même ordre d'idées, la visualisation des figures prises des objets 3D permet d'avoir une représentation (très simple, bien sûr) de ce qui serait le point de départ pour une reconstruction en trois dimensions basé en deux dimensions. Le défi ultime de ce projet est d'avoir des photos naturelles directement (au lieu d'avoir des figures générées par une grammaire et projetées sur un plan) mais avec ce point de départ simplifié, les méthodes qui doivent être développées seront plus facilement testées.

AII.2 Manuel de programmeur

Ce projet a été développé en Visual Studio .NET 2003 pour les parties d'interface et génération de l'interprétation des L-Systems, et en **MATLAB** pour le calcul des caractéristiques des figures étant comparées par l'algorithme. Ici il est question de la documentation pour le(s) futur(s) programmeur(s) travaillant dans l'extension de ce travail.

La meilleure façon de générer de la documentation de qualité pour un projet est de documenter pendant que l'implantation est faite. Cependant, documenter le code est souvent la dernière chose faite sur un cycle de projet typique ; même si l'importance de bien documenter est soulignée dans tous les ouvrages de Génie Logiciel, les développeurs normalement investissent la meilleure partie du temps à construire des nouvelles fonctionalités, et finalement faire la documentation à la fin.

Ceci a aussi été mon cas pour cette thèse ; mais j'ai voulu quand même bien faire : en Visual Studio .NET 2003 la documentation du code avec un format **XML** est supportée pour **C#**, mais non pour Visual Basic .NET. Cependant, il est possible d'installer un *plug-in* gratuit appelé **VBCommenter PowerToy** ; ceci permet de documenter avec format XML dans tout Visual Studio .NET 2003. VBCommenter peut être récupéré à www.gotdotnet.

com/workspaces/workspace.aspx?id=112b5449-f702-46e2-87fa-86bdf39a17dd (voir article http://msdn.microsoft.com/vbasic/default.aspx?pull=/library/en-us/dv_vstechart/html/VBGeneratingDocs.asp) Après avoir écrit la documentation XML, j'ai utilisé NDoc (http://ndoc.sourceforge.net/), un générateur de documentation pour .NET. NDoc est un outil gratuit, avec licence de code source libre, qui génére la documentation en différents formats, incluant le style MSDN pour HTML (.chm), le format Visual Studio .NET (HTML Help 2), et le style MSDN-Online pour pages Web.

Toute la documentation pour le projet est comprise dans le CD accompagnant cette thèse.

ANNEXE III

Fonction d'adéquation basée sur les réseaux de neurones

En botanique et en d'autres contextes agronomes (par exemple, en horticulture et en études forestières) il est nécessaire de quantifier la différence et la variabilité à l'intérieur d'un ensemble de plantes. Mais comparer des branchements (qui représentent naturellement l'architecture de la structure) d'une façon intuitive pour un être humain est un problème délicat. Ici, je discute de la pertinence d'utiliser un réseau de neurones pour apprendre des perceptions humaines sur la différence entre deux architectures de plantes. Un ensemble de caractéristiques est définie pour une figure de plante et les données pour les expériences sont obtenues à l'aide d'un ensemble d'usagers à travers un système interactif et en ligne. Les résultats initiaux et les analyses correspondantes sont présentées.

AIII.1 Introduction et travail antérieur dans le domaine

La modélisation de plantes, et particulièrement de leur structure/fonction, est très importante sous différents contextes : en sciences forestières, par exemple, où la production et la qualité du bois sont estimées à travers des variables comme *diamètre de la branche*, *volume de la couronne* et *géométrie de la branche* (Ferraro et al., 2004). Le même argument tient pour des domaines comme l'horticulture et la botanique ; et, comme la structure d'une plante est étroitement liée à sa fonction et sa santé, être capable de comprendre cette structure peut guider vers des méthodes efficientes d'évaluation de plantations[1].

Il est coutumier de décrire l'architecture des plantes en s'appuyant sur la représentations des structures topologiques avec des graphes (Godin, 2000). Avec cette représentation, un algorithme à complexité finie pour calculer une distance entre des graphes en forme d'arbre a été construit (Zhang, 1993), (Zhang, 1996) ; il a postérieurement été adapté à des architectures de plantes (Ferraro et al., 2004). Le résultat est une distance d'édition, définie comme le coût minimal de la séquence d'opérations élémentaires appliquées à un arbre pour obtenir l'autre. Cette idée de distance forme la base d'une *mesure de dissimilarité* pour comparer des axes d'arbres (Runqiang et al., 2002).

[1]Cet argument est la base de ce travail de thèse.

Une façon directe de définir cette fonction de distance est en comparant deux images avec une métrique point-par-point : c'est essentiellement une généralisation de la distance Euclidienne pour un ensemble de points, et a été implantée pour cette thèse (voir 3.3.2.2, page 75) ; cependant, je n'ai pas été satisfait de la métrique (quelle est la signification de *la distance entre fig1 et fig2 est 35.76 ?* Est-ce une bonne mesure ? Pas trop ?) La question alors c'est de savoir quand nous, comme êtres humains, considérons qu'un arbre *est similaire* à un autre ? La motivation est de réussir à synthétiser une métrique qui traduise la justesse (et subjectivité !) du système visuel humain.

Le reste de cet annexe est organisé comme suit : le design expérimental est présenté à la section AIII.2, avec tous les détails relevants pour l'implantation ; les résultats dérivés de ces expériences sont présentés et discutés en section AIII.3. Les conclusions et réflexions pour le travail futur sont présentés en section AIII.4.

AIII.2 Design expérimental

Étant intéressé à générer une métrique mimant le système visuel humain, l'idée est de lier des résultats obtenus des votes d'usagers par rapport à la ressemblance de paires de figures avec les caractéristiques associées avec ces paires. Pour atteindre ce but, j'ai construit un environnement de travail composé de 3 composantes :

a. **Une façon de décrire une image** : un ensemble de caractéristiques est défini en AIII.2.1.

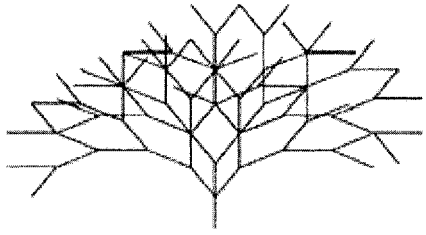
b. **Un système de vote en ligne** pour permettre aux usagers de donner leur opinion sur *la distance entre deux figures* (AIII.2.2).

c. **Un environnement de synthèse d'une fonction basé sur des réseaux de neurones** : un réseau de neurones est entraîné pour prédire ces valeurs (AIII.2.4).

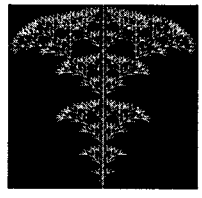
Une base de données d'images fut construite pour expérimenter avec cette idée. Sa description et quelques considérations pratiques sont décrites en AIII.2.3.

AIII.2.1 Caractéristiques d'une figure

Une liste de mesures décrivant une figure générique ont été choisies pour décrire un squelette d'un arbre ; une brève explication pour chacune d'entre elles est présentée ici-bas. Quelques illustrations de ces mesures, appliquées aux Figs. 87(a) and (b), sont aussi montrées (où possible).



(a) Fig. exemple 1



(b) Fig. exemple 2

Figure 87 Exemples de squelettes d'arbres

a. **Disposition des branches** Une caractéristique importante d'un arbre est combien de branches il a, et comment elles sont arrangées sur la figure de l'arbre ; une évaluation à ce sujet est obtenue en mesurant les segments de lignes qui font les branches, et les angles qu'elles font entre elles. Ces deux caractéristiques sont appelées *histogrammes de segments* et *histogrammes d'angles.*

b. **Dimension fractale** : rapidement[2] la dimension fractale d'une forme c'est la portion de l'espace Euclidéen qui est *rempli* par la forme. Celle-ci est appelée *fractale* si cette dimension est non-entière ; la caractéristique essentielle des fractales est la façon dont la structure es (statistiquement) identique, indépendamment de l'échelle sur laquelle elle est vue - une propriété connue comme auto-similarité (Barnsley, 1993).

c. **L'aire** : le nombre de pixels dans la figure.

d. **le centroide** : les coordonnées (x, y) du centre de masse de la région.

[2]Plus de détails en ce document, section 1.1.

e. **La boîte enveloppante** : le plus petit rectangle qui peut contenir la figure (le fonds noir dans les Figs. 87).

f. **Largeur de l'axe majeur** : la largeur (en pixels) de l'axe majeur de l'ellipse qui a le même moment d'ordre 2 que la région.

g. **Largeur de l'axe mineur** : la largeur (en pixels) de l'axe mineur de l'ellipse qui a le même moment d'ordre 2 que la région.

h. **Excentricité** : l'excentricité de l'ellipse qui a le même moment d'ordre 2 que la région. L'excentricité est le ratio de la distance entre le foyer de l'ellipse et son axe majeur.

i. **Orientation** : l'angle (en degrés) entre l'axe des x et l'axe majeur de l'ellipse qui a le même moment d'ordre 2 que la région.

j. **Aire remplie** : dans la région définie par l'image, tous les « trous » sont remplis ; le nombre de pixels dans cette nouvelle image (« *remplie* ») sont comptés (voir Fig. 88).

k. **L'aire convexe** : nombre de pixels dans le plus petit polygone convexe (l'*enveloppe convexe*, *convex hull* en anglais) qui contient la région (Fig. 88

(a) Fig. 87(a) filled (b) Fig. 87(a) convex hull : (c) Fig. 87(b) filled (d) Fig. 87(b) convex hull :

Figure 88 Aires remplies et aires convexes pour les images de la Fig. 87

l. **Diamètre équivalent** : le diamètre d'un cercle avec la même aire que la région. Calculée comme $\sqrt{(4 * Area)/(\pi)}$.

m. **Solidité** : la proportion de pixels dans la coque convexe qui sont aussi dans la région.

n. **Amplitude** : la proportion de pixels dans la boîte enveloppante qui sont aussi dans la région.

AIII.2.2 Système de vote en ligne

Un site web a été construit pour montrer des paires d'images, pris sur un ensemble S (présenté en AIII.5), et pour permettre aux usagers de les comparer. La page principale du site est http://alpamayo.livia.etsmtl.ca:8080/welcomeMetricVote.php[3]. L'usager est demandé d'évaluer la *distance* entre chaque paire d'images présentée ; chaque *vote* est un chiffre entier dans l'intervalle $[0, 10]$, où 0 veut dire *ces 2 images sont très différentes* et 10 veut dire *ces 2 images sont pratiquement les mêmes*. Ces valeurs sont gardées dans un base de données ; la page pour voter est montrée à la Fig. 89

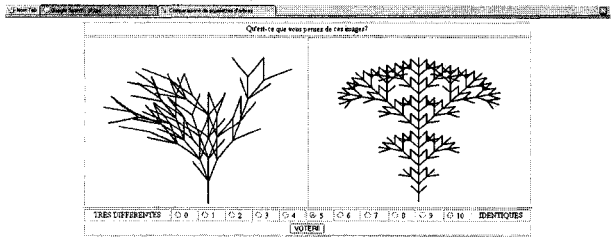


Figure 89 Page principale de votation

La votation consiste à évaluer la (dis)similarité entre les caractéristiques d'un arbre ; un calcul des différences entre les caractéristiques est alors nécessaire. Cette différence est directe pour presque toutes les caractéristiques ; comment cette différence est calculée pour les histogrammes (caractéristique 1) est expliqué en AIII.2.1. Ce processus est calculé en 3 étapes :

a. Normalisation des deux histogrammes en un même nombre de cases (*bins*)

[3] Cette adresse peut ne plus être valide, car l'auteur de cette thèse ne travaille plus au LIVIA. Plus d'information devrait être disponible à http://www.livia.etsmtl.ca/people/costa

b. Calcul d'un histogramme de différences (DH) : les valeurs dans chaque case sont soustraites, et normalisées par la valeur maximale entre les deux. Ceci amplifie les petites différences.

c. Une fois que l'histogramme de différences est calculé, une valeur de distance est calculé : c'est la somme signée au carré des cases dans l'histogramme des différence normalisé.

$$\sum_{bin=1}^{maxBins} sign(DH(j)) * \frac{DHj^2}{MaxValue(j)}$$

AIII.2.3 La base de données d'images

Les images utilisées pour cette expérience ont été générées avec l'environnement de manipulation pour L-Systems implanté pour cette thèse. Vingt (20) images ont été générées avec différentes grammaires, chacune générant différentes formes (voir AIII.5). L'ensemble de caractéristiques présenté en AIII.2.1 est calculé pour chaque image.

Une analyse de composantes principales a été appliquée sur les données pour détecter des régularités ; le résultat est montré au Tableau XV . On peut observer que les 6 composantes

Tableau XV

Résultats ACP

Nombre de composantes	Variabilité expliquée
1	45%
2	69%
3	78%
4	81%
5	88%
6	92%

principales expliquent plus de 90% de la variabilité des données, ce qui comprime de façon considérable les caractéristiques.

AIII.2.4 Prédiction par réseaux de neurones

Les résultats en AIII.2.3 permettent de faire une réduction de dimensionalité des nombres de caractéristiques aux 6 composantes principales ; à partir de cet ensemble, le but est d'inférer un indice de similarité. Un réseau de neurones a été choisi comme outil pour faire cette tâche de régression. Plus précisément, un réseau de neurones perceptron multicouche, avec une couche d'entrée de 6 neurones, 1 couche cachée ($\mathcal{H}$ neurones) et une couche de sortie avec 1 neurone (car le réseau prédit des valeurs de similarité) ; la fonction d'activation pour les neurones dans la couche cachée est la fonction sigmoïde et une fonction linéaire pour le neurone dans la couche de sortie. Le réseau est entraîné avec l'algorithme standard de rétropropagation (McClelland et Rumelhart, 1986).

Trois questions importantes sont à répondre en travaillant avec ce type d'architecture :

a. Combien de neurones forment la couche cachée ? ($\mathcal{H}$)

b. Comment être sûr que les résultats obtenus après entraînement du réseau ne sont pas le résultat d'un minimum local ?

c. Comment entraîner le réseau pour que l'erreur de généralisation soit aussi petit que possible ?

Une façon de résoudre les questions (2) et (3) se retrouve dans la littérature qui traite des problèmes de classification (Efron et Tibshirani, 1993). Il y a deux méthodes fondamentales, appelées *validation croisée* et *amorçage* (*bootstrapping*, en anglais) :

a. **Validation croisée** : échapper à une mauvaise généralisation ; la base de données de votes $\mathcal{D}$ est divisé en 2 ensembles : un *ensemble d'apprentissage* $\mathcal{T}$ et un *ensemble de validation* $\mathcal{V}$. Le réseau est alors entraîné sur $\mathcal{T}$, ayant une erreur t, et testé sur $\mathcal{V}$, ayant une erreur v. L'optimisation du réseau est basé sur la minimisation de t et v.

Une règle rapide pour choisir la taille de chaque ensemble est de donner à $\mathcal{T}$ $2/3$ des éléments en $\mathcal{D}$, et le restant $1/3$ à $\mathcal{V}$.

b. **Amorçage** : dénicher des minima locaux. La performance du réseau peut être très influencée par les choix de $\mathcal{T}$ et $\mathcal{V}$, si ces ensembles ne sont pas des bons représentants du problème complet. Bien comprendre l'influence de ces choix est important ; l'amorçage, ou *bootstrapping*, se réfère au choix des combinaisons de $\mathcal{T}$ et $\mathcal{V}$, répéter les mesures, et alors étudier les propriétés statistiques de chaque expérience.

C'est aussi important de voir qu'une performance non-représentative du réseau peut être le résultat de l'initialisation de ses poids ω, car l'algorithme de rétropropagation ne fait qu'ajuster ces poids. Ces expériences doivent alors être répétées avec différentes initialisations, et les résultats étudiés sous une loupe statistique ; les résultats de cet ensemble divers d'expériences (avec ω, $\mathcal{T}$, et $\mathcal{V}$) sont regroupés comme montré au Tableau XVI. La

Tableau XVI

Amorçage (I)

$\mathcal{H}$					
	seed (ω)				
		amorçage 1	amorçage 2	$\cdots$	amorçage n_b
1		erreur	erreur	$\cdots$	erreur
	1	$(t_1^{(1)}, v_1^{(1)})$	$(t_1^{(1)}, v_1^{(1)})$	$\cdots$	$(t_1^{(1)}, v_1^{(1)})$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	n_s	$(t_{n_s}^{(1)}, v_{n_s}^{(1)})$	$(t_{n_s}^{(1)}, v_{n_s}^{(1)})$	$\cdots$	$(t_{n_s}^{(2)}, v_{n_s}^{(2)})$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
n_h					
	1	$(t_1^{(n_h)}, v_1^{(n_h)})$	$(t_1^{(n_h)}, v_1^{(n_h)})$	$\cdots$	$(t_1^{(n_h)}, v_1^{(n_h)})$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	n_s	$(t_{n_s}^{(n_h)}, v_{n_s}^{(n_h)})$	$(t_{n_s}^{(n_h)}, v_{n_s}^{(n_h)})$	$\cdots$	$(t_{n_s}^{(n_h)}, v_{n_s}^{(n_h)})$

première colonne indique les valeurs de $\mathcal{H}$ étudiées ($\mathcal{H} \in [1, n_h]$) ; chacune de ces valeurs définit un réseau en particulier, dont les poids ω sont initialisés au hasard en n_s façons (deuxième colonne). Pour chacune des paires $(\mathcal{H}, \omega)$, n_b choix différents de $\mathcal{T}$ et $\mathcal{V}$ sont

faits (troisième ligne du tableau). Les résultats sont reportés dans les cellules, en indiquant l'erreur d'entraînement t et l'erreur de généralisation v. Une façon alternative de montrer ces résultats est présentée au Tableau XVII, où les résultats individuels obtenus pour la génération de ω sont regroupés par ses valeurs statistiques de base (meilleure -minimum- valeur, moyenne, et médiane). Ceci permet de faire un choix pour « le meilleur » $\mathcal{H}$:

Tableau XVII

Amorçage (II)

$\mathcal{H}$				
	amorçage 1	amorçage 2	$\cdots$	amorçage n_b
1	$v_{min}^{(1)}, v_{avg}^{(1)}, v_{median}^{(1)}$	$v_{min}^{(1)}, v_{avg}^{(1)}, v_{median}^{(1)}$	$\cdots$	$v_{min}^{(1)}, v_{avg}^{(1)}, v_{median}^{(1)}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
n_h	$v_{min}^{(n_h)}, v_{avg}^{(n_h)}, v_{median}^{(n_h)}$	$v_{min}^{(n_h)}, v_{avg}^{(n_h)}, v_{median}^{(n_h)}$	$\cdots$	$v_{min}^{(n_h)}, v_{avg}^{(n_h)}, v_{median}^{(n_h)}$
	$\Downarrow$	$\Downarrow$	$\cdots$	$\Downarrow$
	$\mathcal{H}_{opt}^{(b=1)}$	$\mathcal{H}_{opt}^{(b=2)}$	$\cdots$	$\mathcal{H}_{opt}^{(b=n_b)}$

$\mathcal{H}_{opt}$; supposons que ce choix se fasse par la minimisation de l'erreur de généralisation moyenne. Alors une lecture des colonnes au Tableau XVII produit un *gagnant* pour chaque amorce, montré dans la dernière ligne de la table. Une valeur de $\mathcal{H}$ devrait apparaître plus fréquemment que d'autres : ceci serait un choix clair pour $\mathcal{H}_{opt}$.

Il serait aussi possible de faire le choix de $\mathcal{H}_{opt}$ en lisant la table par lignes, en calculant une valeur sur celle-ci (par exemple, l'erreur moyenne de généralisation) et choisissant alors la valeur de $\mathcal{H}_{opt}$ comme celle qui a la meilleure de ces valeurs. Les deux façons étant conceptuellement équivalentes, la valeur de $\mathcal{H}_{opt}$ devrait être la même, indépendamment de la méthode pour la choisir. Si elles ne le sont pas, nous pouvons soupçonner que des effets locaux affectent les résultats.

AIII.3 Résultats

Vingt (20) images sont gardées dans la base de données du système ; elles sont montrées à la Fig. AIII.5. Le système prend 2 figures au hasard, et les montre à l'usager ; avec 20 images, $(20) * (21)/2 = 210$ combinaisons sont possibles[4]. Chaque valeur de vote fut transformé à sa valeur z (dans l'équation suivante, x est une valeur et z_x est sa valeur z)

$$z_x = \frac{x - \mu}{\sigma} \tag{III.1}$$

μ et σ sont les moyennes et les écarts types, respectivement, des séquences de données d'où x provient.

Avec cette base de données, un ensemble d'expériences fut mis en place pour choisir un réseau de neurones, de $\mathcal{H}_{opt}$ neurones cachées, qui aurait la meilleure (*plus petite*) erreur de généralisation v. Les paramètres pour chaque expérience sont :

a. *Numéro d'expériences d'amorçage* : nombre de différentes combinaisons pour $\mathcal{T}$ et $\mathcal{V}$. n_b dans le Tableau XVI.

b. $\mathcal{H}$: nombre de neurones cachées pour chaque réseau de neurones

c. $\mathcal{M}_{epochs}$: nombre d'époques maximum sur lequel l'algorithme de rétropropagation s'exécute.

d. $\delta_{\mathcal{G}}$: erreur minimale admissible.

Les mesures pour chaque expérience sont (1) erreur moyenne de validation (sur toutes les expériences d'amorçage) μ_v, et (2) l'écart type de cette erreur σ_v.

AIII.3.1 Premier ensemble d'expériences

[4] Pour cette première approche, (a) seulement une personne a voté dans le système, ce qui m'a permis de tester la robustesse et la faisabilité de l'approche, et (b) chaque paire peut être présentée plusieurs fois au même testeur. 2,581 votes furent enregistrés de cette façon

Ce premier ensemble a comme paramètres $n_b = 10$, $\mathcal{M}_{epochs} = 3000$ et $\delta_{\mathcal{G}} = 0,33$ Les résultats sont montrés au Tableau XVIII Deux points très intéressants ressortent de ces

Tableau XVIII

Résultats (I)

$\mathcal{H}$	μ_v	σ_v	$\mathcal{H}$	μ_v	σ_v
1	1.411	0,035	12	0,666	0,035
2	0,921	0,155	14	0,662	0,035
4	0,767	0,045	16	0,643	0,028
6	0,731	0,047	18	0,592	0,030
8	0,700	0,037	20	0,602	0,031
10	0,696	0,046	22	0,589	0,026

résultats ; premièrement, il est impératif de commenter la justesse et la bonne performance du réseau de neurones : les erreurs sont pour la plupart sous 1 (donc miment l'usager assez bien). Ceci est un point important, et surprenant en même temps.

Deuxièmement, nous pouvons observer comment l'erreur de validation croisée (μ_v) diminue quand $\mathcal{H}$ augmente ; donc, suivant les résultats du Tableau XVIII, la valeur optimale pour $\mathcal{H}$ est $\mathcal{H}_{opt} = 22$, avec $\mu_v = 0,589$

C'est cependant intuitif que, plus il y a de neurones cachés, plus grand est le risque de surentraînement. Pourquoi l'erreur de validation la plus petite est-elle obtenue avec le nombre le plus grand de neurones cachés ? La réponse pourrait se trouver dans le fait que l'algorithme essaie d'approcher les valeurs d'entraînement aussi près que $\delta_{\mathcal{G}} = 0,33$; ceci influence la « vitesse » de convergence, car l'algorithme de rétropropagation est un algorithme de descente de gradient. En mettant plus de pression sur ce point-là (par exemple, en mettant $\delta_{\mathcal{G}} < 0,33$) ceci peut causer un comportement final différent pour les réseaux étudiés ; ceci est l'idée pour le prochain ensemble d'expériences.

AIII.3.2 Deuxième ensemble d'expériences

Les paramètres sont, maintenant : $n_b = 10$, $\mathcal{M}_{epochs} = 3000$ and $\delta_{\mathcal{G}} = 0$ Les résultats sont présentées au Tableau XIX

Tableau XIX

Résultats (II)

$\mathcal{H}$	μ_v	σ_v	$\mathcal{H}$	μ_v	σ_v
12	0,662	0,036	18	0,667	0,038
14	0,662	0,036	20	0,668	0,037
16	0,663	0,037	22	0,667	0,039

Ces résultats confirment l'hypothèse ; même en obtenant des performances similaires, le réseau avec moins de neurones ($\mathcal{H} = 12$, $\mu_v = 0,662$) est le gagnant. Ceci, bien sûr,

Ceci, bien sûr, contredit les résultats du Tableau XVIII, et me fait penser à un problème de surentraînement. Dans cette expérience, où les réseaux qui peuvent avoir ce problème sont implicitement pénalisés, le réseau avec moins de noeuds cachés émerge comme le gagnant.

AIII.3.3 Architecture à utiliser

Ces résultats produisent les données nécessaires pour compléter le design de l'architecture du réseau de neurones prédictif : il a 6 neurones d'entrée (les *valeurs-z* des 6 composantes principales des données), $\mathcal{H} = 12$ neurones cachées, et 1 variable de sortie.

AIII.4 Discussion et travail futur

AIII.4.1 Contribution de chaque caractéristique aux composantes principales

Tout en profitant de la compression des variables d'entrée que nous donne l'analyse par composantes principales, je serai intéressé à savoir lesquelles des variables originales ex-

pliquent mieux les résultats. Ceci pourrait en fait être important d'un point de vue pratique : si il y a une variable qui ne contribue pas *suffisamment* à l'explication de la variabilité, il n'y a pas besoin de la calculer. Pour cela, les contributions des 18 caractéristiques de composantes principales sont calculées et montrées au Tableau XX.

Tableau XX

Pourcentage de contribution

Caractéristique	Nombre des composantes principales (variabilité expliquée) 6 (90%)	4 (+80%)	3 (+75%)	2 (~70%)	1 (~45%)
Distribution des segments	6.37	7.08	5.49	7.05	3.58
Distribution des Angles	6.30	7.19	5.59	7.10	3.69
Aire	3.66	3.95	4.63	5.99	**8.53**
Largeur axe majeur	5.69	7.22	5.20	6.85	5.52
Largeur axe mineur	5.87	5.09	5.72	4.60	7.69
Excentricité	8.87	**9.05**	6.39	4.97	4.34
Orientation	7.89	9.66	8.07	2.75	2.08
sin(Orientation)	6.62	4.81	5.74	4.00	3.77
cos(Orientation)	**5.66**	0	0	0	0
Aire convexe	4.22	5.21	6.11	7.41	7.34
Aire remplie	4.25	5.54	6.41	4.86	7.82
Diamètre équivalent	3.30	3.39	4.28	5.17	8.74
Solidité	5.91	5.87	6.73	7.66	5.08
Extension	5.33	4.54	5.26	7.38	4.63
Dimension Fractale	5.19	3.94	4.61	3.55	6.52
X du Centroide	5.50	6.04	7.56	6.89	6.85
Y du Centroide	5.21	6.80	6.57	6.58	6.51
Boîte enveloppante	4.16	4.61	5.63	7.19	7.31

Quelques observations pourront être faites de ces données :

a. L'importance des caractéristiques **Aire ,Aire convexe, Aire remplie, Diamètre équivalent** et **Boîte enveloppante** paraissent jouer un rôle plus important pour les premières composantes principales (2 et 3, jusqu'à 75% d'explication de la variabilité) ; elles paraissent être moins importantes quand plus de composantes sont choisies.

b. Les caractéristiques **Excentricité** et **cos(Orientation)** paraissent être importantes pour expliquer l'explication fine de la variabilité, car elles commencent à être importantes quand la variabilité expliquée est plus grande que 80%

AIII.4.2 Travail futur et en cours

Bien que le travail présenté ici est une recherche en cours, j'ai été surpris de la bonne performance montrée par le réseau de neurones. Comme observé dans les données générées par le système de vote, la variabilité de ceux-ci pour un ensemble de caractéristiques peut être importante, même dans ce cas réduit, où juste 1 personne vote ! En d'autres mots, en considérant l'espace multidimensionnel défini par les 18 caractéristiques, 2 points appartenant à une zone spécifique de l'espace peuvent recevoir des votes assez différents. Alors j'ai espéré avoir des résultats intéressants, mais j'ai été surpris par la bonne capacité de prédiction obtenue avec cette méthode.

D'un autre côté, ceci peut seulement être pris pour ce qu'il est : des résultats préliminaires. Il est impératif d'en faire l'extension avec plus d'usagers, votant un nombre plus grand de fois, pour obtenir une vue statistique cohérente de la base de données. Ceci va aussi permettre d'entraîner le réseau avec plus de données.

Une idée à laquelle je travaille est d'intégrer le système de vote en ligne avec le prédicteur ; dans ce sens, chaque fois qu'un vote est régistré par une personne, le réseau de neurones pourrait aussi *voter*, et les valeurs comparées. Ceci permettrait au réseau d'être re-entraîné quand les données insérées dans la base de données diverge de façon substantielle des prédictions, ou d'implanter un système d'alarmes pour alerter les chercheurs du type de votes qui y sont générés.

AIII.5 Les images

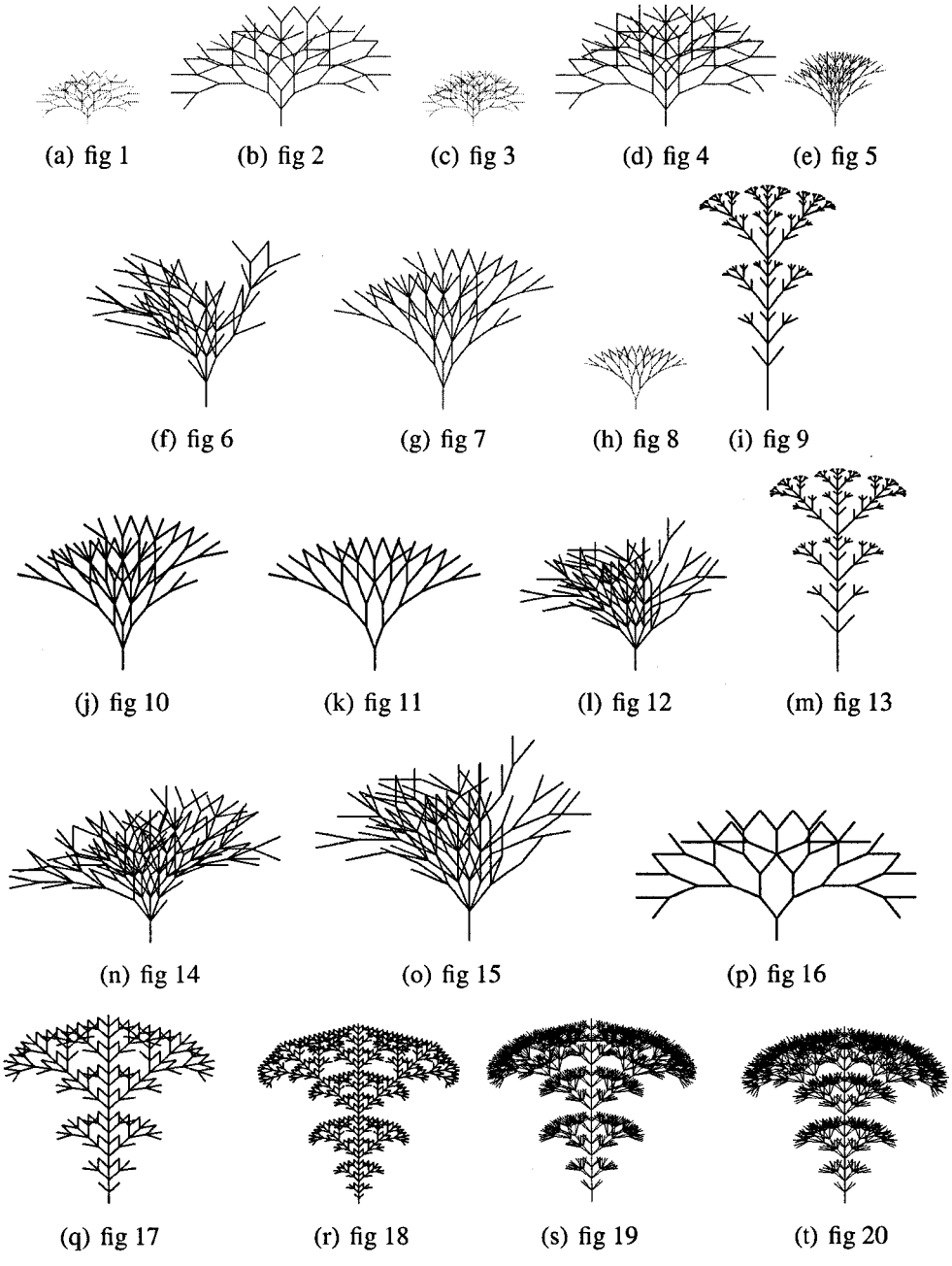


Figure 90 Figures d'arbres pour les expériences

ANNEXE IV

Dimensionnalité de l'espace de recherche pour l'algorithme

Comme décrit dans le chapitre 3 de ce document, pour générer les grammaires L-System candidates à modéliser un certain processus de développement cible P il nous suffit de lancer l'algorithme 1 avec un ensemble d'axiomes (car les règles de production de chacune des grammaires est défini par le schéma (3.6), page 60). Donc chercher les solutions pour approximer un certain processus de développement cible P se fait en 2 étapes :

a. Calculer/choisir un ensemble d'axiomes $\mathcal{E}_a$.

b. Pour chaque axiome, générer les grammaires « intéressantes » et évaluer leur distance géométrique par rapport au processus cible.

Dans cette partie du document je calcule le cardinal de l'espace de recherche exploré par l'Alg. 1. Les deux sous-sections correspondent au calcul du nombre d'axiomes sur lequel $\mathcal{E}_a$ est choisi (section AIV.1) et le calcul du nombre de grammaires générées par un des axiomes $\in \mathcal{E}_a$ (section AIV.2)

AIV.1 Espace de recherche pour l'algorithme génétique

Le cardinal de l'ensemble d'axiomes disponible pour l'algorithme est défini par 2 paramètres des axiomes conformant les populations génétiques :

a. La longueur des individus, mesuré en nombre de symboles

b. Le nombre de symboles dérivables que ces individus ont

Pour cela, nous allons définir les individus de la population comme ayant une longueur dans le rang $[1, l_m]$; chaque individu aura une quantité de symboles dérivables dans le rang $[1, n_m]$

Examinons le cas de 1 symbole s appartenant à un axiome ; ce symbole est soit un symbole dérivable, soit il ne l'est pas.

a. Si il est dérivable, il peut prendre une des 3 valeurs **B**, **T** ou **Y** ; chacun de ces symboles a des paramètres qui lui sont propres. Je ferai le cas de **B**, et les résultats seront appliqués à **Y** et **T** sans aucun problème.

   - La règle associée à **B** est $B(a,b,c) \rightarrow S+(a)\&(c)F(b)$ Donc b est une longueur, a et c des angles ; par la façon comme nous avons fait l'implantation, les angles peuvent prendre une valeur dans $[0, 180]$ degrés, avec des intervalles de 15 degrés. Donc chaque paramètre représentant un angle peut prendre $180/15 = 12$ valeurs. Également, une longueur peut prendre une valeur dans l'intervalle $[1, 5]$, donc 5 valeurs. Au total, pour chaque symbole **B** nous avons $5*12*12 = 5*12^2$ combinaisons.
   - Par le même raisonnement, nous calculons que pour chaque symbole **T** nous avons $5 * 12^2$ combinaisons, et pour chaque symbole **Y** nous avons $5 * 12^3$ combinaisons.

   Donc, s étant dérivable implique que il appartient à l 'une de ces $(5*12^2)*(5*12^2)*(5*12^3) = 5^3 * 12^7$ combinaisons.

b. Si s est non-dérivable, il peut prendre une de 7 valeurs possibles : F, $+$, $-$, $/$, $\backslash$ et $\wedge$. Ces symboles : $+$, $-$, $/$, $\backslash$ et $\wedge$ ont 1 paramètre, un angle ; par l'argument fait ci-dessus, cela veut dire que nous aurons un ensemble de 12 possibilités pour chacun d'eux. F prend 1 paramètre de longueur, donc son ensemble de possibilités est 5. Donc, s étant non-dérivable implique que il appartient à l 'une de ces $12^6 * 5$ combinaisons.

Maintenant, prenons le cas de un axiome S, de longueur l_s, d_s symboles dérivables ($d_s \ll l_s$) Les d_s symboles dérivables peuvent se retrouver de $C\begin{pmatrix} l_s \\ d_s \end{pmatrix}$ façons différentes dans S, et nous aurons alors $(5^3 * 12^7)^{d_s}$ possibilités pour les choisir. Les autres $l_s - d_s$ possibilités

correspondent à $(12^6 * 5)^{(l_s - d_s)}$ combinaisons. Donc nous pouvons choisir cet axiome S de $\mathcal{C}\begin{pmatrix} l_s \\ d_s \end{pmatrix} * (5^3 * 12^7)^{d_s} * (12^6 * 5)^{(l_s - d_s)}$ façons différentes.

Ce chiffre est assez imposant; un exemple numérique est montré dans le tableau XXI Pour mesurer ces valeurs, disons que si une expérience utilise une population de taille

Tableau XXI

Cardinal de la population d'axiomes

l_s	d_s	combinaisons
10	1	$\sim 2,4 * 10^{55}$
10	2	$\sim 5,2 * 10^{60}$
10	3	$\sim 6,64 * 10^{65}$
20	1	$\sim 2,4 * 10^{105}$
20	2	$\sim 1,09 * 10^{111}$
20	3	$\sim 3,16 * 10^{116}$
50	2	$\sim 8,86 * 10^{260}$

100, pendant 500 générations (évaluant donc $100 * 500 = 5 * 10^4$n axiomes), elle aura navigué, au maximum, $\sim 10^{-49}\%$ de l'espace de recherche (avec $l_s = 10, d_s = 1$, ce qui est une expérience trop petite pour être utile).

Commentaire Dans ce calcul j'ai omis de mettre le facteur que ces axiomes peuvent avoir des crochets (car je travaille avec des L-Systems à crochets)... Il faut refaire ce calcul dans ce sens.

AIV.2 Étant donné un axiome, combien de grammaires ?

Comme décrit au chapitre 3, pour pouvoir évaluer l'adéquation d'un certain axiome ω_x par rapport à un certain problème P, nous faisons l'expansion de ω_x avec un algorithme déterministe. Cet algorithme calcule toutes les combinaisons possibles de solutions qui peuvent émaner de ω_x, et les évalue dans les différents points temporels spécifiés par P (nommons ceux-ci $\{t_1, t_2, \cdots, t_{np}\}$, où , $t_i < t_j, \forall i < j$) Une optimisation majeure

de l'algorithme consiste à ne pas évaluer, à une profondeur donnée t_i, les solutions qui seraient dérivées de solutions *mal adaptées* aux points t_j où $t_j < t_i$.

Chaque solution (une grammaire L-System) est représentée par l'évaluation graphique de son état (un LString) ; donc se demander *combien de solutions l'algorithme doit évaluer ?* est équivalent à se demander *combien de LString différents la méthode proposée génére ?* Ainsi, je vais faire ici ce dernier calcul.

AIV.2.1 Sous-problème

Pour cela, je vais commencer par une version modifiée de la question, consistant à prendre (a) un seul point temporel t_n et (b) un LString $\mathcal{S}_0$ quelconque (pas nécessairement l'axiome), pour calculer le nombre de LString générés. $\mathcal{S}_0$ a été généré, après i itérations, par une grammaire G ayant comme axiome $\omega = \omega_x$.

Par définition (voir schéma (3.6), page 60), les symboles qui génèrent d'autres chaînes sont les symboles dérivables (B, Y, et T, car des règles de production ayant un côté gauche avec ces symboles peuvent exister dans G) et le symbole d'axiome, ω, car il y a toujours une règle implicite dans les grammaires disant $\omega \longrightarrow \omega_x$ Appelons $\mathcal{D}(\mathcal{S}_0)$ le nombre de symboles dérivables dans $\mathcal{S}_0$, $\mathcal{N}(\mathcal{S}_0)$ le nombre de symboles ω présents en $\mathcal{S}_0$, et d_ω le nombre de symboles dérivables dans l'axiome.

Je vais définir avec une récurrence, basée sur la valeur de t_n, combien de différentes chaînes peuvent être générées à partir de $\mathcal{S}_0$; les cas de base sont simples à voir :

a. $t_n = 0$: aucune itération à faire, donc $\mathcal{S}_0$ n'a pas changé.

b. $t_n = 1$: chaque symbole dérivable peut ou bien être dérivé, ou bien ne pas l'être. Donc le nombre de différents chaînes est $2^{\mathcal{D}(\mathcal{S}_0)}$.

Pour compter le cas récursif $t_n = n$ il faut savoir la structure des LString générés à l'itération $n-1$; je montre la structure générée à l'itération 1 (donc des $2^{\mathcal{D}(\mathcal{S}_0)}$ combinaisons) :

- 1 chaîne où aucun des $\mathcal{D}(\mathcal{S}_0)$ symboles ne sont dérivés. Cette chaîne aura 0 symboles ω, et $\mathcal{D}(\mathcal{S}_0)$ symboles dérivables.
- $\mathcal{D}(\mathcal{S}_0)$ chaînes où 1 des $\mathcal{D}(\mathcal{S}_0)$ symboles sont dérivés Ces chaînes auront 1 symbole ω, et $(\mathcal{D}(\mathcal{S}_0) - 1)$ symboles dérivables.
- $\vdots$
- $\mathcal{C}\begin{pmatrix} \mathcal{D}(\mathcal{S}_0) \\ k \end{pmatrix}$ chaînes où k des $\mathcal{D}(\mathcal{S}_0)$ symboles sont dérivés. Ces chaînes auront $\mathcal{C}\begin{pmatrix} \mathcal{D}(\mathcal{S}_0) \\ k \end{pmatrix}$ symboles ω, et $(\mathcal{D}(\mathcal{S}_0) - \mathcal{C}\begin{pmatrix} \mathcal{D}(\mathcal{S}_0) \\ k \end{pmatrix})$ symboles dérivables.
- $\vdots$
- 1 chaîne où **tous** les $\mathcal{D}(\mathcal{S}_0)$ symboles sont dérivés. Cette chaîne aura $\mathcal{D}(\mathcal{S}_0)$ symboles ω, et 0 symboles dérivables.

Donc, si j'appelle $\aleph(d_\omega, \mathcal{D}(\mathcal{S}_0), \mathcal{N}(\mathcal{S}_0), n-1)$ le nombre de LString générés à partir de $\mathcal{S}_0$ après $n-1$ itérations, le nombre de LString à la génération n sera

$$\sum_{k=0}^{\mathcal{D}(\mathcal{S}_0)} \mathcal{C}\begin{pmatrix} \mathcal{D}(\mathcal{S}_0) \\ k \end{pmatrix} \aleph(d_\omega, \mathcal{N}(\mathcal{S}_0) * d_\omega + (\mathcal{D}(\mathcal{S}_0) - k), k, n-1)$$

Cette récurrence est donc définie par l'équation (IV.1)

$$\aleph(d_a, d, s, n) = \begin{cases} 1 & \text{, si n} = 0 \\ 2^d & \text{, si n} = 1 \\ \sum_{k=0}^{d} \mathcal{C}\begin{pmatrix} d \\ k \end{pmatrix} \aleph(d_a, s * d_a + (d-k), k, n-1) & \text{, sinon} \end{cases} \quad \text{(IV.1)}$$

Faisons un exemple pour voir l'ordre de magnitude de ces valeurs. Supposons que nous avons une grammaire avec un axiome ω_x

$$\omega_x = F(1)\mathbf{B}(a,b,c)F(2)\mathbf{Y}(1,2,3,4)/(100) + (15)$$

(donc $d_\omega = 2$) ; dans une de ses dérivations, une chaîne $\mathcal{S}_0$ est générée ;

$$\mathcal{S}_0 = F(1)\omega\mathbf{B}(a,b,c)F(2)\mathbf{Y}(1,2,3,4)$$

(donc $\mathcal{D}(\mathcal{S}_0) = 2, \mathcal{N}(\mathcal{S}_0) = 1$) L'objectif est de calculer combien de chaînes seront générées en n itérations ; ces valeurs de $\aleph(d_\omega, \mathcal{D}(\mathcal{S}_0), \mathcal{N}(\mathcal{S}_0), n) = \aleph(2,2,1,n)$ sont indiquées dans le tableau XXII pour différentes valeurs de n.

Tableau XXII

(Exemple I) Nombre de solutions générées à partir de $\mathcal{S}_0$

n	$\aleph(2,2,1,n)$	n	$\aleph(2,2,1,n)$
1	4	4	12,544
2	36	5	1,081,600
3	441	6	435,348,225

Le même calcul fait avec un autre axiome ω_x ayant $d_\omega = 3$ et un autre $\mathcal{S}_0$ ayant $\mathcal{D}(\mathcal{S}_0) = 3$ est présenté au Tableau XXIII.

Tableau XXIII

(Exemple II) Nombre de solutions générées à partir de $\mathcal{S}_0$

n	$\aleph(3,3,1,n)$	n	$\aleph(3,3,1,n)$
1	8	4	73,034,632
2	216	5	$1,41 * 10^{14}$
3	35,937	6	$4,56 * 10^{23}$

AIV.2.2 Le sous-problème augmenté

Nous savons maintenant compter, grâce à l'équation (IV.1), combien de chaînes valides sont générées par l'algorithme à partir d'une chaîne intermédiaire $\mathcal{S}_0$. Dans cette section nous nous posons la question : *comment j'augmente ce résultat pour le cas où l'algorithme évalue en n itérations $(t_1, t_2, \cdots, t_n)$?*

L'algorithme s'éxécute de la facon suivante :

(1) faire l'expansion de toutes les possibles valeurs est faite pour t_1

(2) choisir le meilleur

(3) remplacer $\mathcal{S}_0$ par ce meilleur, et retourner au point (1)

Ce qui est intéressant à voir ici c'est que nous ne pouvons pas, sans plus d'information sur le problème a résoudre, savoir quel est le point qui va « gagner » à chaque profondeur. Donc nous ne pouvons pas connaître, a priori, la valeur des paramètres $\mathcal{D}(\mathcal{S}_0)$ et $\mathcal{N}(\mathcal{S}_0)$. Ce que nous devons faire c'est calculer **tous** les cas, et faire de statistiques sur ces valeurs. De cette façon, nous pourrons estimer les valeurs de l'algorithme, avec une certaine confiance.

Dans le tableau XXIV sont montrées quelques valeurs pour l'axiome.

Une fois ce chiffre obtenu, il est facile de voir que le nombre médian d'évaluations m faite dans une expérience est $m = n_a * v_m * n_g$, où n_a est le nombre d'axiomes dans la population, v_m le nombre de solutions généré, et n_g le nombre de générations que l'algorithme roule. Dans le tableau XXV sont montrées quelques valeurs pour une population d'axiomes. $n_a = 100$, $n_g = 500$.

AIV.2.3 Détails d'implantation

Dans cette section je présente les appels spécifiques aux fonctions **MATLAB** qui m'ont permis de faire les calculs présentés ci-dessus.

Tableau XXIV

Nombre de solutions à évaluer (par individu)

d_ω	itérations	nombre médian d'évaluations
2	$\{t_1 = 2\}$	9
2	$\{t_1 = 3\}$	49
2	$\{t_1 = 4\}$	256
3	$\{t_1 = 2\}$	27
3	$\{t_1 = 3\}$	1,331
3	$\{t_1 = 4\}$	54,872
2	$\{t_1 = 2, t_2 = 4\}$	28
3	$\{t_1 = 2, t_2 = 4\}$	649

Tableau XXV

Nombre de solutions à évaluer (population)

		nombre d'évaluations médian	
d_ω	itérations	(1 solution)	(expérience)
2	$\{t_1 = 2\}$	9	$4,5 * 10^5$
2	$\{t_1 = 3\}$	49	$2,45 * 10^6$
2	$\{t_1 = 4\}$	256	$1,28 * 10^7$
3	$\{t_1 = 2\}$	27	$1,35 * 10^6$
3	$\{t_1 = 3\}$	1,331	$6,66 * 10^7$
3	$\{t_1 = 4\}$	54,872	$2,74 * 10^9$
2	$\{t_1 = 2, t_2 = 4\}$	28	$1,4 * 10^6$
3	$\{t_1 = 2, t_2 = 4\}$	649	$3,25 * 10^7$

Premièrement, pour ce que j'ai appelé le **sous-problème** (AIV.2.1 de cet annexe), les calculs des Tableaux XXII et XXIII ont été effectués par la fonction appelée *calculate-NumberOfDerivations*. En particulier, ils ont été obtenus en tapant dans l'environnement (pour $n = 5, d_\omega = 2, \mathcal{D}(\mathcal{S}_0) = 2, \mathcal{N}(\mathcal{S}_0) = 1$) :

```
>> n = calculateNumberOfDerivations(2, 2, 1, 5)
n = 1081600
```

Deuxièmement, pour le **Sous-problème augmenté** (AIV.2.2 de cet annexe), les calculs sont menés à bien avec une fonction suivant le nom de *allEvaluationsNumber* (l'implantation consistant en un appel répété à l'équation présentée avant, avec un contrôle de la récursion). Les paramètres cette fois consistent en les mêmes que spécifiés pour $\mathcal{S}_0$, plus un vecteur consistant des valeurs $\{t_1, t_2 - t_1, t_3 - t_2, \cdots, t_n - t_{n-1}\}$ Ceci retourne un vecteur avec tous les nombres d'évaluations qui ont été effectuées en prenant les divers chemins (possibilités).

Par exemple, si nous voulons savoir la distribution du nombre des évaluations de $\mathcal{S}_0$, qui a comme valeurs celles présentées au tableau XXII, avec $\{t_1 = 2, t_2 = 4\}$, nous faisons :

```
>> vEvals = allEvaluationsNumber(2, 2, 1, [2 2])
vEvals =
    82         109         109         109         109
   145         145         145         145         145
   145         193         193         193         193
   257         244         325         325         325
   433         433         433         577         244
   325         325         325         433         433
   433         577         730         973         973
  1297
>>
```

Nous voudrions trouver, par exemple, la médiane et la moyenne de cet ensemble. Ces valeurs seraient 349.4 pour la moyenne, 291 pour la médiane.

Le cas spécial de l'axiome (donc, A = $\mathcal{S}_0$) s'écrit $\aleph(d_\omega, d_\omega, 0, n)$ pour n itérations, et l'évaluation à différentes profondeurs pour $d_\omega = 2$ donne le résultat suivant (profondeurs = $\{t_1 = 2, t_2 = 4\}$)

```
>> vEvals = allEvaluationsNumber(2, 2, 0, [2 2])
vEvals =
    10    13    13    17    28    37    28    37    82
>> mean(vEvals)
   29.4444
>> median(vEvals)
    28
```

BIBLIOGRAPHIE

Apter, M. J. (1966). *Cybernetics and development*, volume 29 of *International Series of Monographs in Pure and Applied Biology/Zoology Division*. Pergamon Press, Oxford.

Arber, A. (1950). *Natural philosophy of plant form*. Cambridge University Press.

Barnsley, M. F. (1993). *Fractals everywhere*. Academic Press Limited, London, 2 edition.

Barnsley, M. F. et Demko, S. (1985). Iterated function systems and the global construction of fractals. *Proc. Royal Society London*, Ser. A(399) :243–275.

Barthélémy, D. (1991). Levels of organization and repetition phenomena in seed plants. *Acta Biotheoretica*, 39 :309–323.

Bell, A. (1991). *Plant form : An illustrated guide to flowering plants*. Oxford University Press, Oxford.

Bentley, P. J. (1999). *Evolutionary Design By Computers*. Morgan Kaufmann, San Francisco, CA.

Borchert, R. et Honda, H. (1984). Control of development in the bifurcating branch system of tabebuia rosea : A computer simulation. *Botanical Gazette*, 145(2) :184–195.

Cantor, G. (1883). Uber unendliche, lineare punktmannigfaltigkeiten v. *Mathematische Annalen*, 21 :545–591.

Cart, J. C. (2003). Direct manipulation of recurrent models. *Computer and Graphics*, 27(1).

Cerovic, Z., Samson, G., Morales, F., Tremblay, N., et Moya, I. (1999). Ultraviolet-induced fluorescence for plant monitoring : present state and propects. *Agronomie*, 19 :543–578.

Chomsky, N. (1956). Three models for the description of languages. *IRE Transactions on Information Theory*, 2(3) :113–124.

Collet, P., Schoenauer, M., Lutton, E., et Louchet, J. (2000). Polar ifs + parisian genetic programming = efficient ifs inverse problem solving. *Genetic Programming and Evolvable machines Journal*, 1(4) :339–361.

de Reffye, P., Dinouard, P., et Barthélémy, D. (1990). Architecture et modélisation de l'orme du japon zelkova serrata (thunb.) makin (ulmaceae) : la notion d'axe de référence. *De la forêt cultivée à l'industrie de demain : 3e colloque Sciences et Industries du Bois, Arbora*, pages 351–352, Bordeaux, France.

Efron, B. et Tibshirani, R. J. (1993). *An Introduction to the Bootstrap*. Chapman and Hall.

Feder, J. (1968). Languages of encoded line patterns. *Information and Control*, 13 :230–244.

Ferraro, P., Godin, C., et Prusinkiewicz, P. (2004). A structural method for assessing self-similarity in plants. *Fourth International Workshop on Functional-Structural Trees Models*, pages 56–61.

Fogel, D. B. (1999). Some recent important foundational results in evolutionary computation. Miettinen, K., Makela, M. M., Neittaanmaki, P., et Periaux, J., editors, *Evolutionary Algorithms in Engineering and Computer Science*, pages 55–73. John Wiley and Sons Ltd, England.

Foley, J., van Dam, A., Feiner, S., et Hughes, J. (1990). *Computer graphics : Principles and Practice*. The Systems Programming Series. Addison-Wesley, Reading, Massachussets, 2nd edition.

Frijters, D. et Lindenmayer, A. (1974). A model for the growth and flowering of aster novae-angliae on the basis of table (1,0)l-systems. Rozenberg, G. et Salomaa, A., editors, *LSystems, Lecture Notes in Computer Science 15*, pages 24–52. Springer-Verlag, Berlin.

Frijters, D. et Lindenmayer, A. (1976). Developmental descriptions of branching patterns with paracladial relationships. Lindenmayer, A. et Rozenberg, G., editors, *Automata, Langages, Development*, pages 57–73, North-Holland, Amsterdam.

Gatsuk, L., Smirnova, O., Zaugolnova, L., et Zhukova, L. (1980). Age states of plants of various growth forms : a review. *Journal of Ecophysiology*, 68 :675–696.

Godin, C. (2000). Representing and encoding plant architecture : a review. *Annals of Forest Science*, 57 :413–438.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Massacusetts.

Hallé, F., Oldeman, R., et Tomlinson, P. (1978). *Tropical Trees and Forest, an Architectural Analysis*. Springer-Verlag, New-York.

Hanan, J. (1992). *Parametric L-Systems and their application to the modelling and visualization of plants*. Ph.d. thesis, University of Regina.

Hart, J. C., Baker, B., et Michaelraj, J. (2003). Structural simulation of tree growth and response. *The Visual Computer*, 19 :151–163.

Heisel, F., Sowinska, M., Miche, J., Lang, M., et Lichtenthaler, H. (1996). Detection of nutrient deficiencies of maize by laser induced fluorescence imaging. *Journal of Plant Physiology*, 148 :622–631.

Herman, G. T., Lindenmayer, A., et Rozenberg, G. (1975). Description of developmental languages using recurrence systems. *Mathematical systems theory*, 8(4) :316–341.

Holland, J. (1992). *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, Massachusetts, 2 edition.

Honda, H. (1971). Description of the form of trees by the parameters of tree-like body. *Journal of Theoretical Biology*, 31 :331–338.

Hutchinson, J. (1981). Fractals and self-similarity. *Indiana Univ. J. Math.*, 30(5) :713–747.

Janssen, J. M. et Lindenmayer, A. (1987). Models for the control of branch positions and flowering sequences of capitula in mycelis muralis (l.) dumont (compositae). *New Phytologist*, 105 :191–220.

Kokai, G., Toth, Z., et Vanyi, R. (1998). Applications of genetic algorithms with more populations for lindenmayer systems. Alpaydin, E. et Fyfe, C., editors, *Proceedings of the International Symposium on Engineering of Intelligent Systems, EIS'98*, pages 324–331.

Koza, J. (1993). Discovery of rewrite rules in lindenmayer systems and state transition rules in cellular automata via genetic programming.

Lindenmayer, A. (1968). Mathematical models for cellular interaction in development. parts i and ii. *Journal of Theoretical Biology*, 18 :280–299 and 300–315.

Lindenmayer, A. (1974). Adding continuous components to l-systems. Rozenberg, G. et Salomaa, A., editors, *LSystems, Lecture Notes in Computer Science*, volume 15, pages 53–68. Springer-Verlag, Berlin.

Lindenmayer, A. (1977). Paracladial relationships in leaves. *Ber. Deutsch. Bot. Ges. Bd*, 90(287–301).

Linsen, L., Karis, B. J., McPherson, E. G., et Hamann, B. (2005). Tree growth visualization. *Proceedings of WSCG 2005*. UNION Agency - Science Press, Plzen, Czech Rebublic.

Luck, H. B. et Luck, J. (1994). Approche algorithmique des structures ramifiées acrotone et basitone des végétaux. Vérine, H., editor, *La biologie théorique à Solignac*, pages 111–148. Polytechnica, Paris.

Luck, J., Luck, H. B., et Bakkali, M. (1990). A comprehensive model for acrotonic, mesotonic, and basitonic branching in plants. *Acta Biotheoretica*, volume 38, pages 257–288.

Lutton, E. (1999). Genetic algorithms and fractals. Miettinen, K., Makela, M. M., Neittaanmaki, P., et Periaux, J., editors, *Evolutionary Algorithms in Engineering and Computer Science*, pages 327–350. John Wiley and Sons Ltd, England.

Mandelbrot, B. B. (1982). *The fractal geometry of nature*. W. H. Freeman, San Francisco.

Mandelbrot, B. B. (1989). *Les objets fractals : forme, hasard et dimension*. Flammarion, 3 edition.

McClelland, J. et Rumelhart, D. (1986). *Parallel Distributed Processing : Explorations in the Microstructure of Cognition*. MIT Press, Cambridge, MA.

McKenna, D. (1994). Squarecurves, e-tours, eddies, and frenzies : Basic families of peano curves on the square grid. *The Lighter Side of Mathematics : Proceedings of the Eugene Strens Memorial Conference on Recreational Mathematics and Its History*. Mathematical Association of America.

Mech, R. et Prusinkiewicz, P. (1996). Visual models of plants interacting with their environment. *Proc Siggraph 96*, pages 397–410. ACM Press, New York.

Mitchell, M. (1996a). *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA.

Mitchell, T. (1996b). *Machine Learning*. McGraw Hill, New York.

Naur, P. et Backus (1960). Report on the algorithmic language algol 60. *Communications ACM*, 3(5) :299–314.

Peitgen, H.-O., Jurgens, H., et Saupe, D. (1992). *Chaos and Fractals : New Frontiers of Science*. Springer-Verlag, New York.

Penuelas et Fillela (1998). Technical focus : visible and near-infra-red reflectance techniques for diagnosing plant physiological status. *Trends Plant Sci.*, 3 :151–156.

Prusinkiewicz, P. (1986). Graphical applications of l-systems. *Proceedings of Vision Interface '86*, pages 247–253. Canadian Information Processing Society (Toronto, Ontario), Vancouver, British Columbia, Canada.

Prusinkiewicz, P. (1987). Applications of l-systems to computer imagery. Ehrig, H., Nagl, M., Rosenfeld, A., et Rozenberg, G., editors, *Graph grammars and their applications to computer science; Third international workshop*, pages 101–145. Springer-Verlag.

Prusinkiewicz, P. (1993). Modeling and visualization of biological structures. *Proceedings of Graphics Interface '93*, pages 128–137, Toronto, Ontario (Canada).

Prusinkiewicz, P., Federl, P., Mech, R., et Karwowski, R. (2003). *L-systems and beyond.* Course notes, SIGGRAPH 2003 (available online at *http ://algorithmicbotany.org/papers/sigcourse.2003.html*).

Prusinkiewicz, P., Hammel, M., Hanan, J., et Mech, R. (1996). L-systems : from the theory to visual models of plants. *Proceedings of the 2nd CSIRO Symposium on Computational Challenges in Life Sciences*, pages 1–27. CSIRO Publishing, Brisbane, Australia.

Prusinkiewicz, P., Hammel, M., Mech, R., et Hanan, J. (1995). The artificial life of plants. *ACM SIGGRAPH*, volume 7 of SIGGRAPH '95, pages 1–38.

Prusinkiewicz, P. et Hanan, J. (1989). *Lindenmayer Systems, Fractals, and Plants*, volume 79 of *Lecture Notes in Biomathematics*. Springer-Verlag, New York.

Prusinkiewicz, P. et Lindenmayer, A. (1990). *The Algorithmic Beauty of Plants*. Springer-Verlag, New York.

Prusinkiewicz, P., Mundermann, L., Karwowski, R., et Lane, B. (2001). The use of positional information in the modeling of plants. SIGGRAPH, editor, *SIGGRAPH Proceedings, Annual Conference Series, 2001.*, pages 289–300, Los Angeles, CA.

Raun, W., Solie, J., Johnson, G., Stone, M., Mullen, R., Freeman, K., Thomason, W., et Lukina, E. (2002). Improving nitrogen use efficiency in ceral grain production with optical sensing and variable rate application. *Agronomy Journal*, 94 :815–820.

Robinson, D. (1996). A symbolic framework for the description of tree architecture models. *Botanical Journal of the Linnean Society*, 121 :243–261.

Ross, J. K. (1981). *The radiation regim and the architecture of plant stands*. Junk W. Pubs, The Hague, Netherlands.

Runqiang, B., Chen, Y.-P. P., Burrage, K., Hanan, J., Room, P., et Belward, J. (2002). Derivation of l-system models from measurements of biological branching structures using genetic algorithms. Eiben, A., Baeck, T., Schoenauer, M., et Schwefel, H.-P., editors, *15th international conference on Industrial and engineering applications of artificial intelligence and expert systems*, Lecture Notes in Computer Science, pages 514–524. Springer-Verlag.

Sierpinski, W. (1915). Sur une courbe dont tout point est un point de ramification. *C. R. Acad. Paris*, 160 :302–305.

Sierpinski, W. (1916). Sur une courbe cantorienne qui contient une image biunivoque et continue de toute courbe donnée. *C. R. Acad. Paris*, 162 :629–632.

Siromoney, R. et Subramanian, K. G. (1983). Space-filling curves and infinite graphs. Ehrig, H., Nagl, M., et Rozenberg, G., editors, *Graph Grammars and their application to computer science ; Second International Workshop. Lecture Notes in Computer Science 153.*, pages 380–391. Springer-Verlag, Berlin.

Smith, A. R. (1984). Plants, fractals, and formal languages. *Computer Graphics (ACM Siggraph)*, 18(3) :1–10.

Szilard, A. L. et Quinton, R. E. (1979). An interpretation for d0l systems by computer graphics. *The Science Terrapin*, 4 :8–13.

Vanyi, R., Kokai, G., Toth, Z., et Reto, T. (2000). Grammatical retina description with enhanced methods. *Proceedings of Genetic Programming 2000.*

von Koch, H. (1905). Une méthode géométrique élémentaire pour l'étude de certaines questions de la théorie des courbes planes. *Acta Mathematica*, 30 :145–147.

Wagon, S. (1991). Biasing the chaos game : Barnsley's fern. *Mathematica in Action*, pages 156–163. W.H. Freeman, New York.

Wolpert, D. H. et Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1) :67–82.

Zhang, K. (1993). A new editing-based distance between unordered trees. *4th CPM'93*, pages 254–256, Padala, Italy.

Zhang, K. (1996). A constrained edit distance between unordered labeled trees. *Algorithmica*, 15 :205–222.

Zimmerman, M. H. et Brown, C. L. (1971). *Trees - structure and function*. Springer-Verlag, Berlin.