$N°$d'ordre : 3508

THÈSE

présentée à

L'UNIVERSITÉ BORDEAUX I

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET INFORMATIQUE

par Fanny CHEVALIER

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : Informatique

Reconnaissance de motifs dans des graphes : heuristique et applications

---

Soutenue le : 12 décembre 2007

Après avis de :

MM.	Yves CHIRICOTA	Professeur	
	Jean-Marc FÉDOU	Professeur	**Rapporteurs**

Devant la commission d'examen formée de :

MM.	Yves CHIRICOTA	Professeur	**Rapporteur**
	Jean-Marc FÉDOU	Professeur	**Rapporteur**
	Maylis DELEST	Professeure	**Directrice**
	Jean-Philippe DOMENGER	Professeur	**Directeur**
	Serge DULUCQ	Professeur	**Examinateur**
	Alexandru TELEA	Professeur	**Examinateur**
	Céline ROZENBLAT	Professeure	**Invitée**

– 2007 –

Reconnaissance de motifs dans des graphes : heuristique et applications

Fanny CHEVALIER

Remerciements

Je remercie les membres du jury, Serge Dulucq, qui me fait l'honneur de présider le jury de cette thèse, Yves Chiricota et Jean-Marc Fédou, les rapporteurs de ce travail de thèse, pour leurs remarques qui m'ont permis d'améliorer ce document et d'appréhender ce travail avec plus de recul, Alexandru Telea et Céline Rozenblat qui ont accepté de participer à ce jury.

Je tiens à remercier très chaleureusement mes directeurs de thèse Maylis Delest et Jean-Philippe Domenger pour leur disponibilité et les moyens qu'ils m'ont accordés. Je conserverai de ces trois années un souvenir très agréable grâce à leurs grandes qualités humaines. Je leur suis également reconnaissante pour la richesse des enseignements scientifiques qu'ils m'ont transmis. Je remercie tous les enseignants rencontrés au cours de mon cursus scolaire qui m'ont donnée l'envie de suivre cette voie. Je pense particulièrement à Jenny Benois-Pineau qui m'a fait confiance très tôt et m'a encouragée à poursuivre dans la recherche. Je remercie Nicolas Hanusse qui m'a été d'une aide précieuse. Ses remarques et conseils, ainsi que le temps qu'il m'a consacré ont très largement contribué à l'aboutissement de ce travail. Ce travail a été réalisé au sein du Laboratoire Bordelais de Recherche en Informatique, dirigé par Serge Dulucq. Je suis reconnaissante des moyens mis à ma disposition ainsi que de l'aide reçue de la part des membres de ce laboratoire, que ce soit dans le cadre des enseignements que j'ai eu le plaisir d'assurer ou pour mes démarches administratives. Je pense en particulier à Isabelle Dutour, Michel Marcus, Serge Dulucq, Eric Sopena, Cathy Roubineau, Philippe Biais, Fabienne Clairand, ...

Je pense également à Anthony Don, Romain Bourqui, David Auber, J.P. Domenger, Guy Mélançon, Noël Novelli pour leur imperturbable bonne humeur qui a su entretenir une ambiance de travail extrêmement agréable. Un salut amical à mes collègues Anthony, Romain, Matthias, Jocelyn, Florian, Maxime, Aïda, Charles, Mickaël, Alexandre, Pierre, Pascal, Omer, Mamadou, Martin et tous ceux avec qui j'ai partagé de bons moments.

Un petit coucou à mes amis, Tom et Do, Ji et Nina, Carine et Cédric, Angèle et Nicolas. Une mention spéciale à Al Yen qui saura certainement apprécier ce document.

Une très chaleureuse pensée pour ma famille : mes parents, mes frères adorés Florian, Benoît, J.B. et Séba, sans oublier Chantal, Caro, Erika et mes très chers Martin et Elsa. Je pense aussi à Valérie, Flo, Éléa, Françoise et Gérard. Un clin d'oeil à mon cousin Jean-Charles.

Je termine par le plus important : merci à David qui fait mon bonheur chaque jour et sans qui je n'en serai pas là aujourd'hui.

Reconnaissance de motifs dans des graphes : heuristique et applications

Résumé :

Cette thèse s'inscrit dans la problématique de la comparaison de graphes. Nous proposons dans la première partie de ce manuscrit plusieurs algorithmes de recherche de motifs similaires dans de grands graphes. La seconde partie de cette thèse traite de l'étude des arborescences de fichiers, structure de données pour laquelle la méthode initiale de reconnaissance de motifs a été développée. Nous présentons un modèle stochastique pour la génération aléatoire de structure arborescentes, basé sur les observations de l'étude statistique des données réelles. Enfin, nous détaillons deux adaptations de l'algorithme de reconnaissance de motifs similaires à des applications particulières. La première concerne la reconnaissance d'objets extraits de la video basse résolution pour l'indexation grossière. La deuxième application a été développée pour la visualisation de l'évolution de projets logiciels.

Mots-clef : comparaison de graphes, reconnaissance de motifs, génération aléatoire, indexation multimedia, évolution de projet logiciel, visualisation

Discipline : Informatique

Pattern recognition in graphs : heuristic and applications

Abstract :

The topic of this thesis is graph comparison. In the first part of this manuscript we propose several algorithms for finding similar patterns in large graphs. The second part of this thesis deals with the study of user file systems as the underlying data on which the initial pattern recognition methods were developed. We propose a stochastic model for random generation of trees, based on observations of the statistical study of real data. In the third part, we detail two adaptations of the algorithm for recognizing similar pattern in graphs to specific applications. The first one concerns the recognition of objects extracted from low resolution video for rough indexing. The second application has been developed to visualize software evolution.

Keywords : graph comparison, pattern recognition, random generation, multimedia indexing, software evolution, visualization

Field : Computer Science

Table des matières

Introduction

La représentation des données sous forme de graphe est très répandue. Structure de données universelle et outil puissant, les graphes sont très fréquemment utilisés dans les applications informatiques. L'étude des réseaux d'interactions et structures de graphes recouvre des domaines aussi variés que la sociologie, la génétique, le domaine des transports, l'informatique ou encore la médecine. On peut citer par exemple le métabolisme cellulaire [14] le graphe du Web en informatique [76], les réseaux sociaux [126]. Autant d'exemples qui montrent que les graphes sont une structure de données qui préoccupe une large part de la communauté scientifique. Les réseaux sont depuis longtemps étudiés à travers les mathématiques ou les sciences. On peut considérer Euler comme le précurseur en théorie des graphes avec la résolution du fameux problème des ponts de Königsberg en 1736. Outre son rôle de langage pour décrire les modèles abstraits, la théorie des graphes est devenue un outil pour l'étude des données empiriques. Au début des années cinquante, les mathématiciens ont commencé à considérer les graphes comme des objets stochastiques au sein desquels un phénomène peut se propager (en particulier l'information ou la maladie) [122]. On ne connaissait pas les processus de formation de ces réseaux complexes difficiles à observer dans leur intégralité, et les structures de graphes sous-jacentes ont longtemps été associées à des réseaux aléatoires.

Il existe un très large panel de méthodes et algorithmes d'analyse, manipulation, navigation, dessin, visualisation et autres pour l'étude et la représentation des graphes. L'une des tâches les plus importantes liées à l'utilisation des graphes est leur comparaison. Ainsi, le problème de mise en correspondance de graphes et de sous-graphes font l'objet d'une recherche intensive depuis les quatre dernières décennies. Le problème d'isomorphisme de graphes et de sous-graphes étant complexe, un grand nombre de techniques proposées exploitent les spécificités des données du domaine d'application pour limiter les comparaisons. D'autre part, selon l'application visée, il est souvent pertinent de détecter dans des structures de graphes des éléments qui se ressemblent fortement, sans pour autant être identiques, comme le montrent la plupart des exemples mentionnés dans le paragraphe qui suit. En chimie [143] ou en biologie [7] par exemple, il est fréquent d'observer des similitudes entre des composés chimiques ou des structures d'ARN. En multimédia, un même objet peut varier dans le temps (vidéo) ou apparaître dans des contextes et sous un angle de vue différents (image ou vidéo) [40, 57]. Bien qu'une correspondance parfaite n'existe pas entre les graphes, la détection de motifs ressemblants que l'on appellera *motifs quasi-similaires* peut se révéler une source d'information importante. Nous trouvons ainsi un grand nombre d'approches pour la mise en correspondance de graphes et de sous-graphes pour la résolution de problématiques diverses.

En vision par ordinateur, Lladós *et al.* [81] ont proposé une méthode de reconnaissance de symboles graphiques ou de diagrammes manuscrits. Dans leur approche, un graphe d'adjacence des régions est associé à chaque symbole. La reconnaissance s'effectue par comparaison de graphes en utilisant une méthode basée sur le calcul d'un coût d'édition. En chimie, on peut citer les travaux de Yan *et al.* [143] pour la recherche de sous-structures dans des molécules chimiques. Chaque molécule est dans un premier temps associée à un graphe. Ce graphe est ensuite décomposé en un ensemble de motifs élémentaires prédéfinis. Ces motifs servent d'index pour retrouver une molécule dans une base de données. Dans le domaine du multimédia, de nombreux travaux sont basés sur les techniques de mise en correspondance de graphes. On peut citer Gomila *et al.* [57] qui utilisent des méthodes de relaxation probabiliste pour le suivi d'objets segmentés dans la vidéo. Demirci *et al.* [40] proposent une mise en correspondance des sommets telle que plusieurs régions d'une image requête (associées à plusieurs sommets dans le graphe d'adjacence des régions), peuvent être mises en correspondance avec une seule et même région (donc un unique sommet dans le graphe d'adjacence) de l'image cible. En maintenance de code informatique, les arbres syntaxiques correspondant au code source sont utilisés pour la détection de duplication de code. Baxter *et al.* [17] ont proposé une méthode basée sur la classification des sommets par similarité (au vu de la composition du sous-arbre). Cette classification implique que deux sommets classés dans la même catégorie correspondent à deux sommets racines de sous-arbres similaires. Dans le domaine de la visualisation de l'information, Auber *et al.* [7] ont proposé une technique de visualisation permettant d'identifier visuellement les motifs quasi-similaires entre deux arborescences de fichiers. La méthode, que nous détaillerons dans le paragraphe suivant, s'appuie sur une heuristique utilisant les caractéristiques structurelles des sommets pour la mise en correspondance. Le résultat de la mise en correspondance de motifs quasi-similaires est présenté comme une coloration des sommets telle que deux motifs de la même couleur sont des motifs quasi-similaires.

Au cours de cette thèse, nous nous sommes intéressés à l'heuristique de recherche de motifs quasi-similaires dans des arborescences proposée par Auber *et al.* [7]. Cette méthode a été développée initialement lors du concours Infovis 2003 [52] pour répondre à différentes tâches concernant la comparaison de structures arborescentes.

La première contribution de ce travail de thèse concerne la formalisation de l'heuristique généralisée aux structures de graphes.

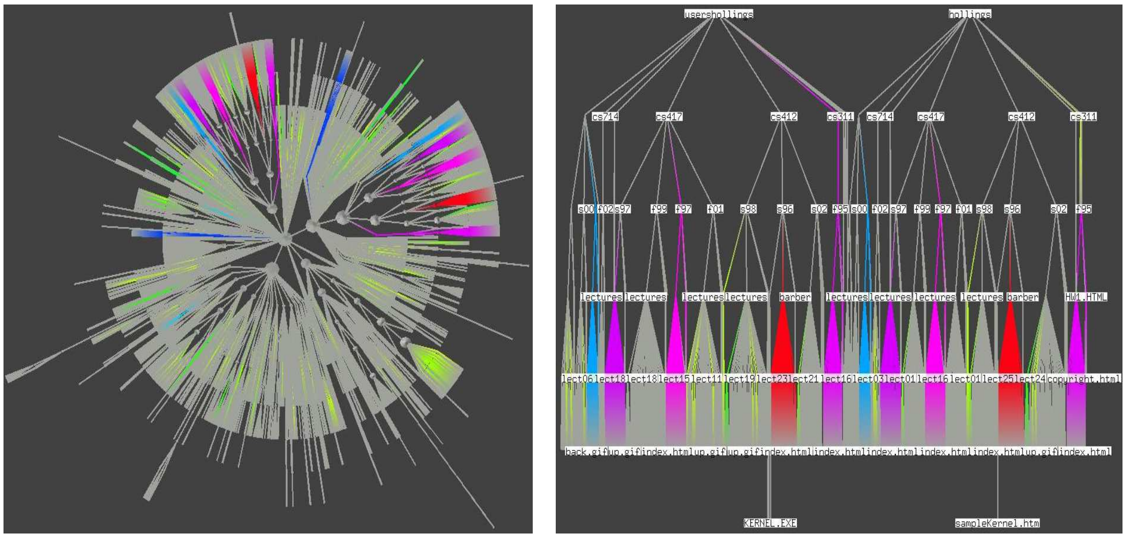
Pour cerner les raisons des performances de cette approche, nous avons choisi d'étudier les structures pour lesquelles la méthode a initialement été développée, à savoir les arborescences de fichiers utilisateurs. Nous avons ainsi proposé un modèle de génération aléatoire permettant de reproduire des arbres aux mêmes propriétés afin de valider la méthode sur un plus large jeu de données.

Enfin, nous avons adapté la méthode à deux problématiques très différentes : la reconnaissance d'objets dans la vidéo à basse résolution et le suivi de blocs de code similaires dans un projet logiciel.

L'heuristique de détection de motifs quasi-similaires

Cette méthode a été développée pour répondre aux tâches concernant la comparaison d'arborescences de fichiers. L'objectif fixé par le concours Infovis 2003 [52] était de proposer une méthode d'évaluation permettant de détecter visuellement les similarités et les changements entre deux versions différentes d'une même arborescence.

Ainsi, la première application utilisant la méthode heuristique de détection de motifs similaires, que nous détaillerons dans la section 2.1 de ce manuscrit, est la solution logicielle EVAT [7], interface de visualisation permettant à l'utilisateur d'identifier visuellement, par la coloration des sommets et arêtes des arborescences, si des changements ont eu lieu entre deux versions d'un système arborescent de fichiers. La figure 0.1 montre un exemple de comparaison de structures arborescentes au sein d'un même système de fichiers issu du concours Infovis 2003 [52]. Une même couleur suggère des motifs quasi-similaires. Sur la figure 0.1.a, on remarque la présence de deux sous-arbres proches (des motifs rouges, roses et bleus sont visibles dans les deux sous-arbres). En montrant le détail de ces deux sous-arbres (Fig. 0.1.b) on constate en effet que les répertoires nommés `hollings` et `usershollings` sont très proches en terme de structure et de contenu.

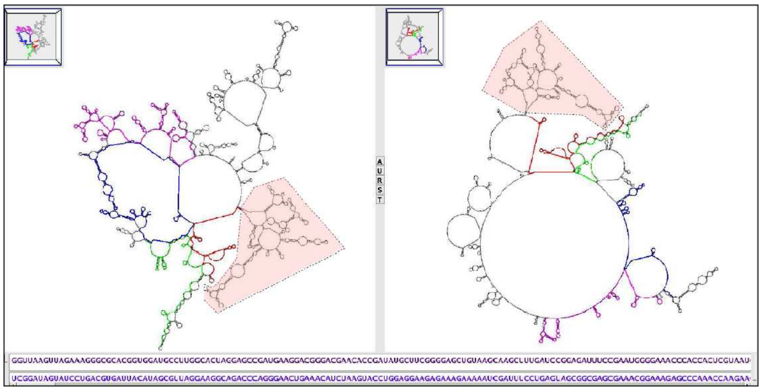


(a) (b)

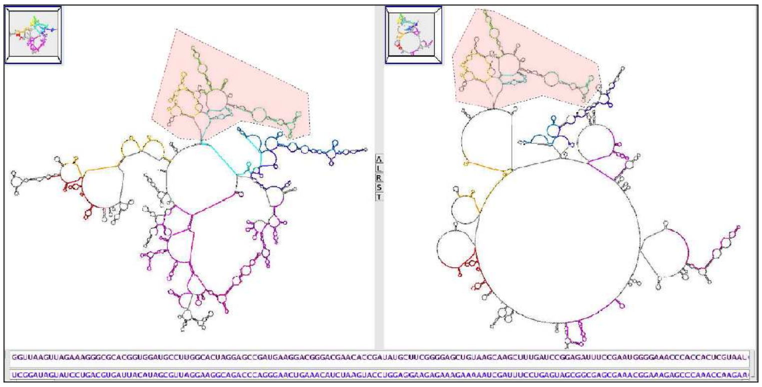
FIG. 0.1: EVAT : Visualisation de motifs quasi-similaires dans un système de fichiers.

Par la suite, Auber *et al.* ont repris la méthode dans un algorithme pour le dessin de structure secondaires d'ARN [6]. Le principe est d'identifier le plus grand motif similaire entre les deux structures à comparer afin d'utiliser un même plongement dans le plan des éléments qui composent le motif ressemblant. Ainsi, la partie commune étant représentée avec la même orientation dans les deux dessins, la comparaison visuelle sur le reste de la structure est facilitée pour l'utilisateur.

Un exemple de cette application est montré sur la figure 0.2. Le plus grand motif quasi-similaire est représenté par les zones en rouge. La comparaison sur les structures avant rotation (Fig. 0.2.a) est moins évidente que sur le dessin prenant en compte les motifs similaires (Fig. 0.2.b).



(a)



(b)

Fig. 0.2: Dessin de structures ARN secondaires utilisant la reconnaissance de motifs similaires. Dessin avant rotation (a) et après rotation (b).

Contributions

Nous proposons dans le chapitre 2 une formalisation de cette heuristique généralisée aux graphes. Pour la reconnaissance de motifs quasi-similaires, la méthode procède en deux phases. Dans un premier temps, les sommets des graphes sont classifiés en différentes familles de sommets partageant les mêmes caractéristiques. La deuxième phase concerne la construction des motifs quasi-similaires : l'algorithme se base sur la comparaison des étiquetages des voisinages des sommets à comparer.

Nous avons introduit l'arbre de couverture [18] pour optimiser la phase de construction des familles de sommets aux mêmes propriétés.

D'autre part, nous définissons ici trois variantes pour la reconnaissance de motifs quasi-similaires. Ces variantes se distinguent par la stratégie de comparaison du voisinage. L'ensemble des sommets considérés pour la comparaison de motifs, selon la variante, est constitué de :
- une partie du voisinage direct des sommets
- l'intégralité du voisinage direct des sommets
- le sous-graphe issu des sommets.

Dans les chapitres 4 et 5, nous présentons une adaptation de l'heuristique pour la mise en correspondance d'objets dans la vidéo (Chap. 4) et l'évolution de code dans un projet logiciel (Chap. 5). Dans ces applications, les données extrinsèques sont utilisées pour aider à la mise en correspondance.

Nous avons également testé l'heuristique, sans introduire d'information propre aux données, pour la mise en correspondance de motifs quasi-similaires dans des graphes représentant le trafic aérien, ou encore des graphes de migrations alternantes (Chap. 2). La figure 0.3 montre un exemple de recherche de motifs quasi-similaires entre deux versions d'un réseau de trafic aérien. Les sommets coloriés en vert représentent les sommets qui ont été mis en correspondance. La figure 0.4 représente le détail d'une partie de la figure 0.3.

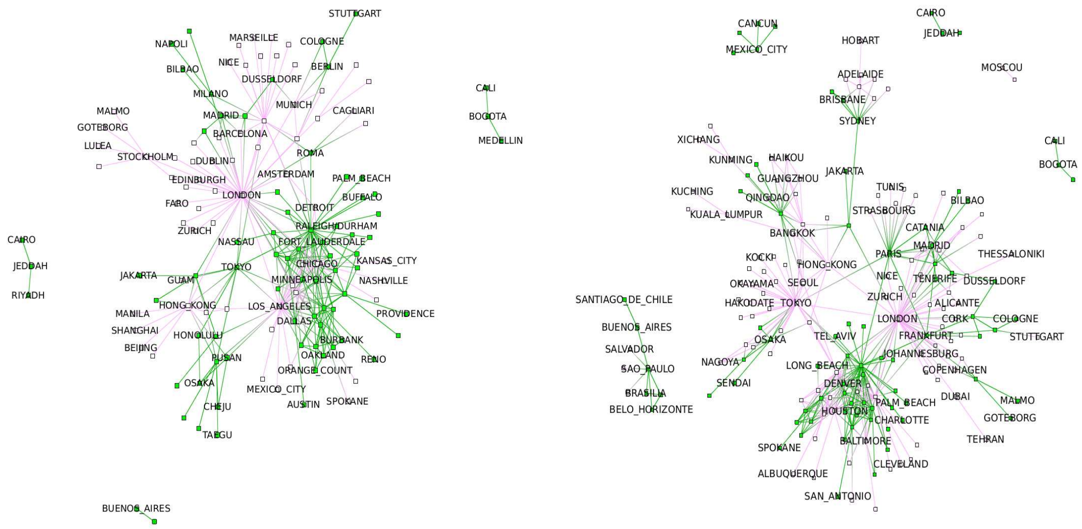


FIG. 0.3: Motifs quasi-similaires (en vert) entre le réseau de trafic aérien en 2000 et en 2004.

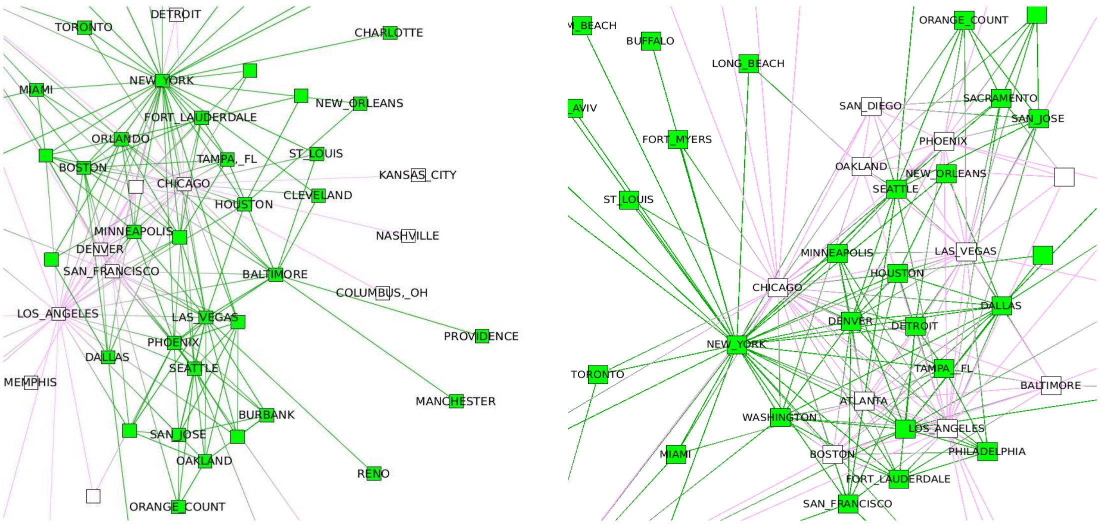


FIG. 0.4: Motifs quasi-similaires (en vert) entre le réseau de trafic aérien en 2000 et en 2004. Détail de la figure 0.3.

Sur cette deuxième figure (Fig. 0.4), nous pouvons identifier de nombreuses similitudes. Le réseau aérien des États-Unis a été identifié comme motif commun entre les deux périodes. Nous pouvons constater que de nombreuses villes (New-York, Toronto, Minneapolis, Seattle, Dallas, San Jose, ...) sont effectivement présentes dans les deux motifs.

Génération aléatoire

La génération aléatoire de graphes et structures arborescentes a fait l'objet d'une recherche très active dans les dernières décennies [97]. En effet, la génération aléatoire depuis un modèle est fréquemment utilisée dans le but de valider, d'évaluer les performances des algorithmes. Bien souvent, l'acquisition de données réelles est difficile pour diverses raisons (moyens matériels, droit à la propriété, ...) et les jeux de données obtenus sont limités en taille. Le fait de disposer d'un modèle de génération aléatoire permet de construire des jeux de données aux propriétés proches de la réalité et ainsi de pallier à ces problèmes.

Dans les dernières décennies, l'apparition des ordinateurs personnels dans les foyers et l'émergence de l'Internet ont suscité de nouveaux problèmes pour la recherche dans le domaine des graphes [76, 97]. Par exemple, de nombreuses études dans le but de comprendre l'architecture et les propriétés sous-jacentes des réseaux complexes particuliers que l'on peut rencontrer dans la vie courante ont été possibles grâce au progrès des moyens informatiques. Ces études ont permis entre autres de révéler que la plupart des graphes issus des réseaux réels obéissent à certaines lois de structuration organisée alors que la première hypothèse suggérait un comportement aléatoire. En effet, cette hypothèse découle des travaux de 1959 des mathématiciens Paul Erdös et Alfréd Rényi [46] qui, pour décrire les réseaux réels, ont suggéré un modèle de génération aléatoire de graphes qui connecte les sommets en plaçant des arêtes aléatoirement uniformément. Ce premier modèle de génération aléatoire de graphes a été considéré comme unique modèle pendant de nombreuses années pour simuler les graphes aléatoires.

Plus récemment, dans la fin des années 80, la recherche en synthèse d'images a soulevé le problème de la génération aléatoire de structures permettant de modéliser des plantes réalistes dans un décor en trois dimensions. L'utilisation de modèles paramétrés (ou de modèles différents) est nécessaire pour générer à la fois des arbres dont la structure ressemble à celle d'un chêne avec beaucoup de ramifications, donnant une impression d'arbre “touffu”, et des arbres qui ont les caractéristiques des peupliers par exemple qui sont des arbres plutôt filiformes. Par conséquent, pour répondre au besoin de générer des arborescences aux caractéristiques diverses, de nombreux modèles de génération aléatoire de structures arborescentes ont été proposés, dont une synthèse peut être consultée dans [109].

L'étude des propriétés des structures de graphes réels et surtout la reproduction des structures aux mêmes propriétés est donc un thème essentiel qui concerne différents domaines de recherche.

Nous proposons dans ce manuscrit un modèle pour la génération aléatoire de systèmes de fichiers utilisateurs.

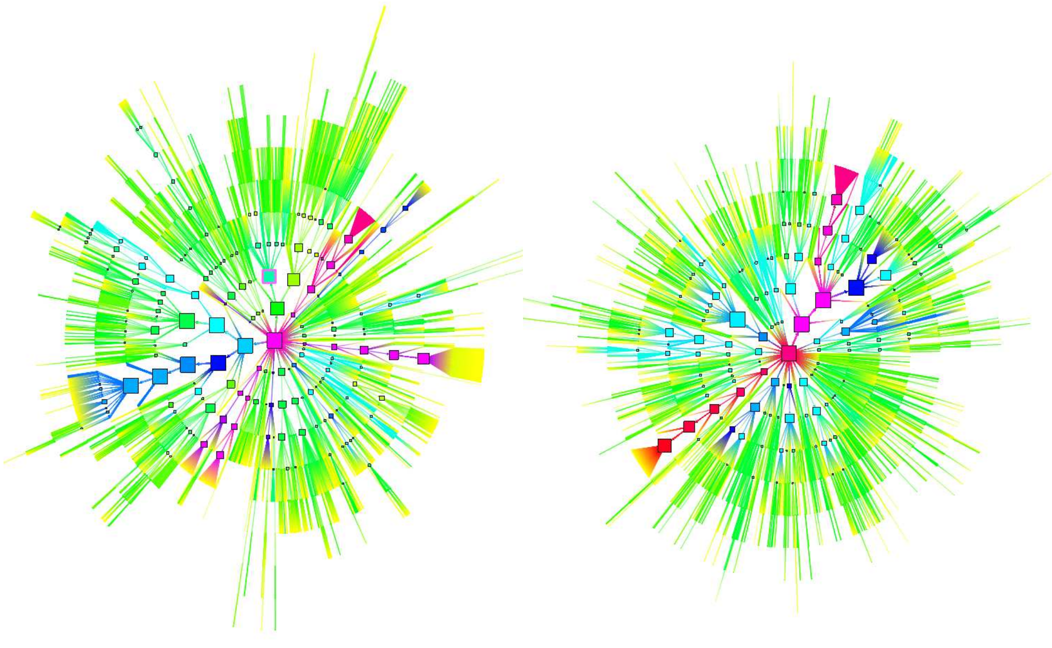
Contributions

Le chapitre 3 présente une étude statistique des arborescences de fichiers réelles qui a permis d'extraire un certain nombre de propriétés sur ces structures arborescentes particulières.

À partir de cette étude, nous avons développé un modèle de génération aléatoire basé sur les observations, permettant de produire des arbres proches des systèmes de fichiers réels. Le modèle intègre l'ensemble des propriétés que nous avons observées sur les données réelles.

Nous avons défini deux versions du modèle : la version de base consiste à construire une série d'arbres indépendants aux propriétés proches des arborescences de fichiers réelles. Dans la version étendue, nous avons cherché à modéliser les différents "comportements utilisateurs" pour la génération d'un système de fichiers. Le profil de l'utilisateur influera sur la structure arborescente obtenue. Le modèle proposé permet ainsi de générer des systèmes de fichiers complets, comprenant des utilisateurs aux profils variés.

La figure 0.5 montre le résultat de la détection de sous-arbres quasi-similaires entre le système de fichiers de référence (Fig. 0.5.a) et le système de fichiers artificiel obtenu par génération aléatoire (Fig. 0.5.b). Nous pouvons observer sur la visualisation que de multiples motifs ont été mis en correspondance entre les deux structures. Le détail de l'exemple de la figure 0.5 est décrit dans la section 3.3.6.1.



(a) (b)

Fig. 0.5: Motifs quasi-similaires entre le système de fichiers de référence (a) et un système de fichier généré avec notre modèle (b).

Ces résultats tendent à valider les performances de l'heuristique pour la recherche de motifs quasi-similaires dans des arborescences de fichiers. D'autre part, le fait de détecter des sous-arbres quasi-similaires entre le système de fichiers de référence et le système de fichiers généré tend à montrer que le modèle de génération aléatoire proposé reproduit des structures arborescentes proches de la réalité.

Indexation de documents multimédia

La recherche d'images indexées basée sur le contenu ("Content-based image retrieval" (CBIR)) désigne l'ensemble des techniques permettant la recherche d'images et de documents audiovisuels dans de grandes bases de données. Le terme "indexation basée sur le contenu" signifie que le processus de recherche utilise des informations extraites du contenu des images elles-mêmes plutôt que des méta-données (mot-clés, descriptions) ajoutées manuellement.

Du point de vue de l'utilisateur, un système CBIR idéal, devrait permettre d'effectuer des requêtes sémantiques telles que "trouve des images contenant une voiture rouge". Mais ce type de requêtes est très difficile à mettre en oeuvre à cause de la diversité des instances de chaque concept dans les images. Les systèmes actuels reposent essentiellement sur l'analyse des caractéristiques bas niveau issues de l'image telles que les couleurs, les textures et les formes. Cependant, il est encore extrêmement difficile d'exprimer des concepts sémantiques à l'aide de caractéristiques bas niveau. On parle de "fossé sémantique" ("semantic gap" [59]). Certains systèmes utilisent des caractéristiques de plus haut niveau telles que la présence de visages dans l'image ou la reconnaissance de l'orientation de l'image (portrait/paysage) pour enrichir l'indexation par le contenu.

En plus des techniques d'analyse et d'indexation, de nouveaux concepts d'interaction sont requis pour permettre l'accès à des contenus visuels [19]. Les requêtes des utilisateurs peuvent prendre différentes formes :

- les requêtes par l'exemple ("query by example"), pour lesquelles l'utilisateur fournit une image représentative des images recherchées. Cette image peut être produite par l'utilisateur ou issue d'un échantillon de la base d'images. Le système recherche ensuite les images de la collection les plus similaires à la requête selon les caractéristiques indexées [23, 78]
- les requêtes par croquis ("query by sketch") pour lesquelles l'utilisateur crée un dessin du document recherché. L'interface utilisateur doit alors permettre de dessiner les zones qui composent l'image et de fixer leurs attributs de couleur, texture ou trajectoire [27, 54, 78]
- d'autres méthodes utilisent directement un vecteur constitué de valeurs numériques associées aux caractéristiques de l'image recherchée ("query by features"). Par exemple, son histogramme de couleurs [54].

Des techniques d'interaction sont également utilisées pour aider l'utilisateur à formuler et reformuler ses requêtes. Les techniques de bouclage de pertinence ("relevance feedback") impliquent l'utilisateur dans un cycle de raffinements de la requête initiale. À partir d'un ensemble d'éléments qui lui sont présentés, l'utilisateur indique la pertinence de chaque élément. La requête est alors adaptée pour tenir compte de ces informations [23, 90].

La recherche de documents multimédia par le contenu est donc devenue l'un des thèmes de recherche les plus actifs à l'heure actuelle.

Contributions

Dans le chapitre 4, nous proposons deux nouvelles méthodes pour la mise en correspondance d'objets dans la vidéo à basse résolution spatiale et temporelle [31, 30].

Dans un tel contexte, les données sont fortement dégradées et les techniques de reconnaissance doivent être adaptées à cette dégradation.

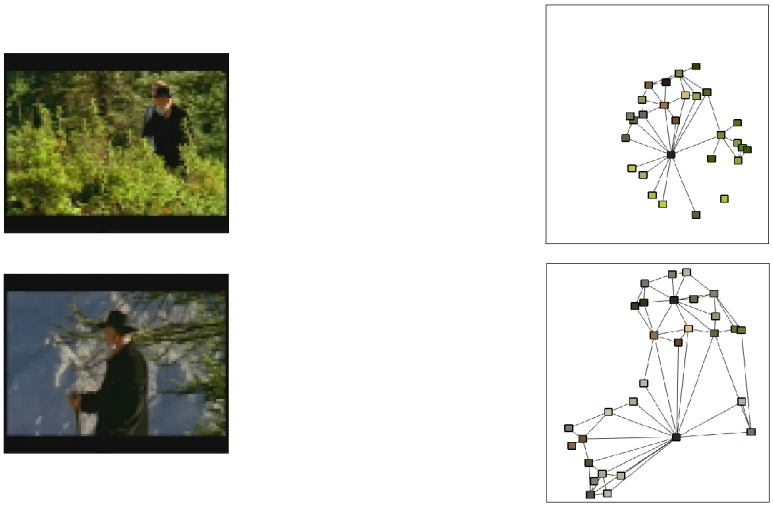
La première méthode est basée sur les techniques de relaxation probabiliste [111, 68]. Après avoir calculé une mesure de similarité initiale entre les régions qui constituent les objets à comparer, l'algorithme corrige la similarité en tenant compte de la cohérence du voisinage local. Pour cette approche, nous avons proposé d'une part une nouvelle mesure de similarité entre les régions basée sur des descripteurs adaptés à la résolution des données. D'autre part, nous avons défini une nouvelle fonction de mise à jour de la similarité en fonction de la cohérence du voisinage. Nous avons également introduit une stratégie de mise en correspondance progressive des régions selon leur surface : en considérant les régions les plus grandes dans un premier temps, nous limitons les problèmes liés à la faible résolution des données.

La deuxième méthode est une adaptation de la méthode heuristique de reconnaissance de motifs quasi-similaires présentée dans ce manuscrit. Le principe est de modéliser les objets de la vidéo sous forme de graphes d'adjacence des régions et de comparer ces derniers pour identifier les parties quasi-similaires. Dans cette approche, les propriétés extrinsèques aux données telles que la couleur ou la surface des régions sont utilisées dans la comparaison des structures de graphes.

La figure 0.6 montre un exemple de représentation de deux objets extraits d'une vidéo sous forme de graphe d'adjacence des régions. Les deux méthodes de mise en correspondance proposées s'appuient sur cette représentation. Sur la figure 0.7, on montre le résultat de la recherche d'objets similaires dans une base de données vidéo via la méthode par relaxation (Fig. 0.7.a) et via l'heuristique (Fig. 0.7.b). La requête (encadrée de rouge) est dans chacun des cas retournée comme objet le plus similaire.

Aide à la maintenance logicielle

Le développement d'un logiciel informatique nécessite un important effort de conception et de maintenance. La réalisation de projets logiciels d'envergure est le plus souvent effectuée par des équipes ayant un nombre important de développeurs. Les problèmes dus à des logiciels de taille de plus en plus grande ont été résolus par le développement d'outils pour aider à la coordination des travaux des différents développeurs. Ces outils ont pour objectif d'aider à la maintenance du projet logiciel.



(a) (b)

Fig. 0.6: (a) Images vidéos originales et les graphes d'adjacence des régions correspondants.

Fig. 0.7: Recherche par l'exemple dans une base d'objets extraits de la vidéo.

La plupart des outils d'aide à la maintenance logicielle exploitent les données du gestionnaire de versions. Le gestionnaire de versions (CVS, Subversion, ...) permet de conserver constamment un historique des différentes versions communes à tous les développeurs travaillant sur le projet. Le rôle du gestionnaire est de centraliser les travaux des différents programmeurs et de conserver un historique des modifications apportées à chaque mise à jour de l'un ou l'autre développeur. Les gestionnaires de versions constituent un puissant outil pour le développement d'un projet logiciel d'envergure. Cependant, ils ne permettent pas d'interpréter facilement les modifications qui ont eu lieu dans le projet dans sa globalité.

Ce n'est que dans les dernières années que l'on voit apparaître des outils pour l'aide à la maintenance logicielle. Parmi les techniques les plus répandues, on peut citer les techniques de détection de duplication de code dans un projet logiciel. La duplication de code est un phénomène très fréquemment observé dans la plupart des projets logiciels : il est classique de recopier des portions de codes pour les réutiliser après avoir effectué de légères modifications. Les raisons de la présence de nombreux clones dans le code sont multiples : la difficulté de factoriser, la nécessité d'éviter les dépendances entre les modules ou encore la pratique usuelle de la production de code par l'exemple (copié-collé). L'existence de clones dans le code introduit de nombreuses instabilités dans une application logicielle. Ainsi, la détection de ces derniers en vue d'éliminer les duplications par une refactorisation de code constitue l'une des tâches essentielles de la maintenance logicielle. Un grand nombre de travaux ont été proposés pour résoudre le problème de détection de clones dans le code : comparaison textuelle des lignes de codes [45], comparaison des propriétés syntaxiques de blocs de code [89] ou encore la comparaison d'arborescences construites à partir de l'analyse syntaxique du code [17, 137, 73, 50].

Par ailleurs, des applications exploitant l'information des gestionnaires de versions pour l'aide à la maintenance logicielle ont émergé dans la dernière décennie [82, 136, 134, 135, 53]. L'idée est de proposer à l'utilisateur une interface de visualisation qui met en évidence les différents évènements importants de l'évolution ayant eu lieu dans le code au fil des versions. On peut citer par exemple l'ajout ou la suppression d'un bloc de code au niveau de détail de la ligne de code, les changements importants dans l'équipe de développeurs, *etc.* L'utilisation d'interfaces de visualisation pour faciliter l'interprétation de l'évolution d'un projet logiciel permet ainsi aux développeurs et responsables d'avoir une connaissance plus étendue du projet.

Contributions

Dans le chapitre 5, nous proposons une nouvelle méthode pour l'identification et la visualisation de l'évolution de blocs de code quasi-similaires dans les différentes versions d'un projet logiciel. La méthode est utilisée pour détecter et analyser les changements de petite et moyenne échelle entre les différentes versions d'un code source.

La méthode se base sur une comparaison des arbres formés à partir d'une analyse syntaxique du code. Nous utilisons la méthode heuristique pour la reconnaissance de code quasi-similaire. Nous proposons de visualiser les sous-arbres mis en correspondance en se basant sur la représentation dense des arbres et des techniques de rendu améliorant la lisibilité.

La figure 0.8 montre un exemple de reconnaissance de code similaire entre deux versions d'un même fichier source. Les tubes colorés indiquent des blocs de code stables entre les versions. On note ici des suppressions de code (bloc C) et une inversion des blocs A et B.

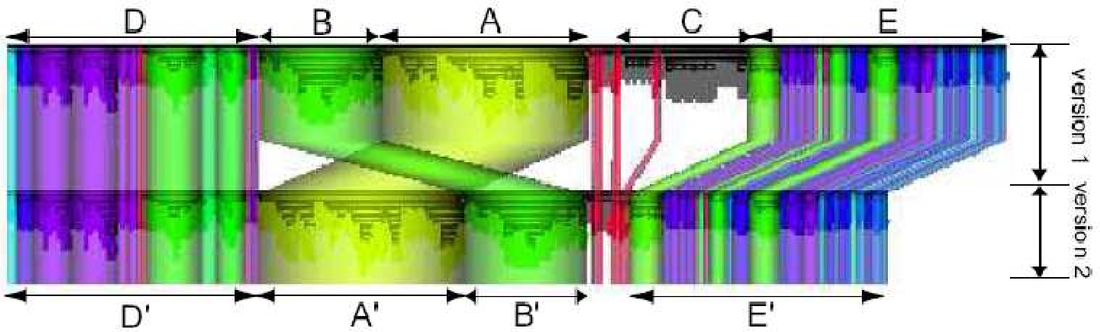


FIG. 0.8: Deux versions d'un fichier source de 850 lignes de code environ.

Chapitre 1
Présentation des domaines

Dans ce chapitre, sont présentés les domaines que nous aborderons dans ce manuscrit. Ce chapitre comporte un certain nombre de définitions et notations essentielles à la compréhension des chapitres suivants de cette thèse. Dans la première section, nous introduisons les définitions et notations générales sur les structures de graphes que nous manipulerons tout au long de ce manuscrit. La deuxième section présente la problématique de la mise en correspondance de graphes et le détail de quelques algorithmes permettant de résoudre ce problème. Nous présentons ensuite le domaine de la génération aléatoire de graphes pour lequel nous avons défini un nouveau modèle. Dans les sections 1.5 et 1.6, nous introduisons la notion d'indexation de documents multimédia (Sec. 1.5), et un aperçu des techniques pour l'aide à la maintenance de projets logiciels (Sec. 1.6), domaines pour lesquels s'inscrivent certaines de nos contributions.

1.1 Définitions et notations générales sur les graphes

Dans cette section sont introduites les notations et définitions relatives aux graphes que nous utiliserons dans cette thèse. Le lecteur pourra également se référer à différents ouvrages complets du domaine [20, 41].

Définition 1 (Graphe)
On appelle graphe le couple $G = (V, E)$ formé par un ensemble V d'éléments appelés sommets et par un ensemble E de paires de sommets $\{u, v\}$. Les éléments de E sont appelés les arêtes de G.

Les graphes sont appelés ainsi car ils peuvent être représentés graphiquement.

Définition 2 (Boucle)
Soit $G = (V, E)$ un graphe. On dit que l'arête $e = \{u, v\}$ est une boucle si $u = v$.

Définition 3 (Multi-arête)
Soit $G = (V, E)$ un graphe et soient e et e' deux arêtes distinctes de l'ensemble E. On dit que $e = \{u, v\}$ et $e' = \{u', v'\}$ sont des multi-arêtes si $\{u, v\} = \{u', v'\}$.

Définition 4 (Graphe fini)
Un graphe est dit fini si les ensembles V et E sont finis.

Définition 5 (Graphe simple)
Un graphe simple est un graphe qui ne contient ni boucle ni multi-arête.

Par la suite, le terme graphe désigne un graphe simple fini. Le graphe de la figure 1.1.a contient une boucle (e_6) et 2 multi-arêtes (e_3 et e_4). Le graphe de la figure 1.1.b quant à lui, est un exemple de graphe simple.

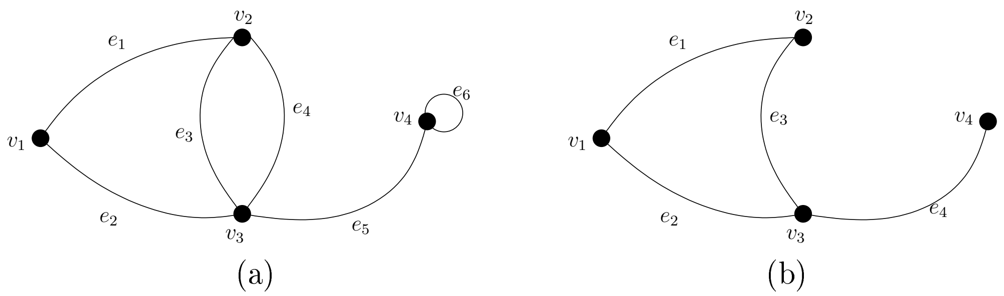


FIG. 1.1: Exemples de graphes : (a) un multigraphe et (b) un graphe simple.

Définition 6 (Graphe planaire)
Un graphe est dit planaire s'il existe un plongement dans le plan sans croisement d'arêtes.

Définition 7 (Sous-graphe)
Soient $G = (V, E)$ et $G' = (V', E')$ deux graphes. G' est un sous-graphe de G, noté $G' \prec G$, si les trois conditions suivantes sont vérifiées :
- $V' \subseteq V$
- $E' \subseteq E$
- $\forall \{u, v\} \in E'$, *on a* $u \in V'$ *et* $v \in V'$.

Définition 8 (Brin)
Le couple (e, u) est un brin d'arête si l'arête e a pour extrémité le sommet u.

Définition 9 (Arc)
Un arc est une paire ordonnée de sommets (u, v) : c'est une arête orientée.
Soit $e = \{u, v\}$ une arête. On notera (u, v) l'arc résultant de l'orientation de e depuis le sommet u vers le sommet v en distinguant le sommet source u noté $src(e)$ du sommet destination v noté $dst(e)$.
On a $(u, v) \neq (v, u)$ si $u \neq v$.

Définition 10 (Graphe orienté)
Un graphe orienté est un couple (V, E) avec $V \subseteq E \times E$.

Définition 11 (Graphe complet)
On dit que le graphe G est un graphe complet si on a $E = \wp_2(V)$ (i.e. l'ensemble de ses sommets sont reliés deux à deux).

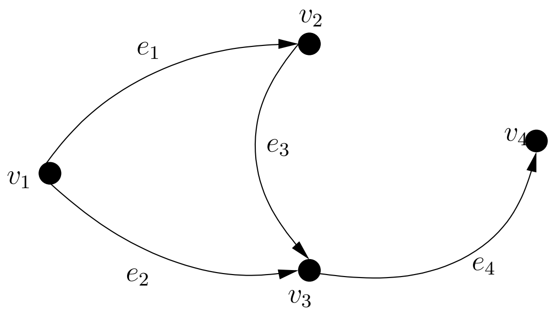


Fig. 1.2: Un graphe orienté.

Définition 12 (Clique)
On appelle clique du graphe G un sous-graphe complet de G.

Définition 13 (Prédécesseur, successeur, voisin)
Soient G un graphe orienté et $v \in V$. L'ensemble des prédécesseurs (resp. successeurs) de v noté $\mathcal{N}^-(v)$ (resp. $\mathcal{N}^+(v)$) est l'ensemble des sommets $\{u \mid (u,v) \in E\}$ (resp. $\{u \mid (v,u) \in E)$. On appelle voisins de v, ou sommets adjacents à v, l'ensemble $\mathcal{N}(v) = \mathcal{N}^-(v) \cup \mathcal{N}^+(v)$.

Sur l'exemple de la figure 1.2, on a $\mathcal{N}^-(v_3) = \{v_1, v_2\}$ et $\mathcal{N}^+(v_3) = \{v_4\}$.

Définition 14 (Degré)
Soient G un graphe orienté et $v \in V$. La cardinalité $|\mathcal{N}^-(v)|$ (resp. $|\mathcal{N}^+(v)|$, resp. $|\mathcal{N}(v)|$) est appelé degré entrant (resp. degré sortant, resp. degré) du sommet v et est noté $d^-(v)$ (resp. $d^+(v)$, resp $d(v)$).

Définition 15 (Chaîne, Chemin)
Une chaîne, appelée aussi parcours (resp. chemin) est une séquence de sommets $(v_1, v_2, \cdots, v_n)$ telle que $\forall i \in [1, \cdots, n-1], \{v_i, v_{i+1}\} \in E$ (resp. $(v_i, v_{i+1}) \in E$). La longueur de la chaîne (resp. chemin) correspond au nombre d'arêtes (resp. arcs) qui la composent.

Définition 16 (Cycle)
On appelle cycle ou circuit tout chemin $(v_1, v_2, \cdots, v_n)$ tel que $v_1 = v_n$.

Définition 17 (DAG)
*Soit G un graphe orienté. On dit que G est un graphe acyclique (*DAG*, en référence à la terminologie anglaise "directed acyclic graph") s'il ne contient aucun cycle.*

Définition 18 (Graphe connexe, fortement connexe)
Un graphe G est dit connexe (resp. fortement connexe) si, pour toute paire de sommets $\{u, v\}$ il existe une chaîne (resp. chemin) dont les deux extrémités sont u et v.

Définition 19 (Composante connexe)
Soit G un graphe. On appelle composantes connexes de G les plus grands sous-graphes connexes G' de G.

Définition 20 (Feuille)
On appelle feuille d'un graphe orienté tout sommet v tel que $d^+(v) = 0$.

Définition 21 (Racine)
On appelle racine d'un graphe orienté (resp. non orienté) tout sommet $v_\perp$ tel que, $\forall v \in V \mid v \neq v_\perp$ il existe un chemin (resp. une chaîne) de $v_\perp$ à v.

Définition 22 (Arbre orienté)
Soit G un graphe acyclique orienté connexe. G est un arbre s'il existe un unique sommet $v_\perp$ racine (Déf. 21) et, $\forall v \in V$, tel que $v \neq v_\perp$, on a $d^-(v) = 1$.

Un arbre $T = (V, E)$ vérifie $|E| = |V| - 1$, et $d^-(v_\perp) = 0$.

Le graphe de la figure 1.3 est un arbre orienté. Sa racine (Déf. 21) est le sommet v_1 et ses feuilles (Déf. 20) sont les sommets v_4, v_5 et v_6.

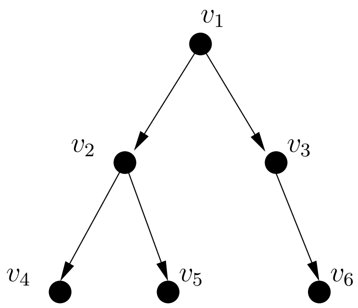


FIG. 1.3: Un arbre orienté.

Définition 23 (Arbre binaire, arbre k-aire)
Soit $T = (V, E)$ un arbre, T est un arbre binaire si tout sommet v de V possède au plus deux successeurs (i.e. $\forall v \in V, d^+(v) \leq 2$). Plus généralement, on appelera arbre k-aire tout arbre vérifiant la propriété : $\forall v \in V, d^+(v) \leq k$.

L'arbre de la figure 1.3 est un arbre binaire.

Définition 24 (Forêt)
Soit $G = (V, E)$ et $W = \{G_1, \cdots, G_k\}$ la décomposition en composantes connexes de G. On dit que G est une forêt si tous les G_i sont des arbres.

Il est clair que tout arbre (resp. forêt) est un graphe simple.

Définition 25 (Profondeur)
Soient T un arbre et v un sommet de T. La profondeur de v, notée $prof_T(v)$ est la longueur du chemin de la racine à v.

Dans l'arbre de la figure 1.3, la profondeur du sommet v_4 est de 2.

Définition 26 (Hauteur)
On appelle hauteur d'un arbre T, notée $h(T)$ la longueur du plus long chemin dans T. La hauteur est définie par

$$h(T) = \max_{v \in V} \ prof_T(v)$$

L'arbre de la figure 1.3 a une hauteur de 2.

Définition 27 (Arbre couvrant)
Soit $G = (V, E)$ un graphe connexe non orienté. Un arbre couvrant $T = (V_T, E_T)$ de G est un sous-graphe de G tel que $V_T = V$, $E_T \subseteq E$, et T est un arbre.

La figure 1.4 montre un graphe G et deux de ses arbres couvrants possibles.

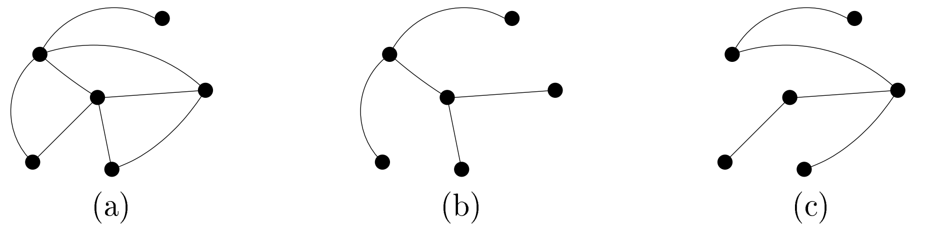


FIG. 1.4: Exemple d'arbres couvrants. (a) Un graphe G. (b) et (c) sont des arbres couvrants de G.

Définition 28 (Représentation par la matrice d'adjacence)
La matrice d'adjacence M d'un graphe orienté $G = (V, E)$ est la matrice de taille $|V| \times |V|$ telle que :

$$M(v, v') = \begin{cases} 1 & \textit{s'il existe un arc } e \textit{ de } E \textit{ tel que } e = (v, v') \\ 0 & \textit{sinon} \end{cases}$$

La figure 1.5 montre la matrice d'adjacence associée au graphe de la figure 1.2.

	v_1	v_2	v_3	v_4
v_1	0	1	1	0
v_2	0	0	1	0
v_3	0	0	0	1
v_4	0	0	0	0

FIG. 1.5: Matrice d'adjacence associée au graphe de la figure 1.2.

Le nombre de Strahler

Le nombre de Strahler a été introduit par Strahler [123] sur les arbres binaires pour l'étude de la morphologie de la structure des réseaux fluviaux. Le principe consiste à associer un nombre à chaque sommet de l'arbre binaire, donnant ainsi une information quantitative sur la complexité de chacun des sous-arbres de l'arbre de départ [130]. Ershov [49] a prouvé que si nous considérons une expression arithmétique et son arbre d'évaluation T, le nombre de Strahler de l'arbre T, augmenté de 1 est le nombre minimal de registres nécessaires à l'évaluation de l'expression arithmétique sur un processeur.

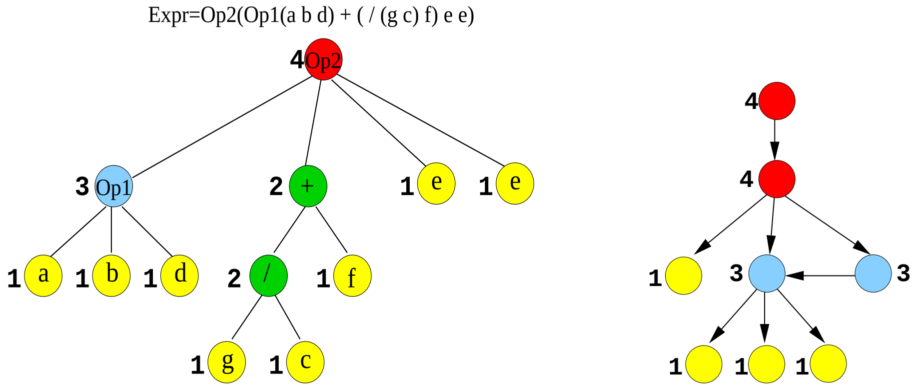
Jean-Marc Fédou a proposé une extension du nombre de Strahler aux arbres planaires [8], et aux arbres généraux [51]. L'idée est de considérer des opérateurs n-aires à la place des opérateurs binaires. Ainsi, le nombre de Strahler sur les arbres généraux est défini comme étant le nombre minimal de registres nécessaires à l'évaluation d'une expression n-aire.

Définition 29 (Nombre de Strahler)
Soit v un sommet interne d'un arbre T possédant $k+1$ successeurs $\{v_i\}_{0 \leq i \leq k}$ tels que, si $i \leq j$, alors $\sigma(v_i) \geq \sigma(v_j)$. Le nombre de Strahler $\sigma(v)$ du sommet v est défini par :

$$\sigma(v) = \begin{cases} 1 \ \textit{si } v \textit{ est une feuille de } T \\ \\ \max\limits_{0 \leq i \leq k} (\sigma(v_i) + i) \ \ \textit{si } v \textit{ possède } k+1 \ \ \textit{successeurs } v_i \end{cases} \tag{1.1}$$

Dans le cas des arbres binaires, le nombre de Strahler étendu donne les mêmes résultats que l'algorithme original sur les arbres binaires. De plus, l'extension du nombre de Strahler aux graphes orientés sans cycle ne demande aucune modification de la définition 29.

La figure 1.6.a montre un exemple du calcul du nombre de Strahler sur une expression n-aire, et 1.6.b sur un graphe orienté sans cycle.



(a) (b)

FIG. 1.6: Nombre de Strahler : (a) expression n-aire avec son arbre d'évaluation, (b) graphe orienté sans cycle.

1.2 Mise en correspondance de graphes

La comparaison des graphes est l'un des problèmes classiques lorsque l'on utilise cette structure de données comme modèle de représentation. En général, la comparaison de graphes consiste à établir une correspondance entre les sommets et les arêtes des structures que l'on cherche à comparer. Cette problématique est définie dans la littérature comme étant un problème de recherche d'*isomorphisme* (ou *quasi-isomorphisme*) de graphes. Dans cette section, nous présentons les algorithmes classiques pour la recherche d'isomorphisme et de quasi-isomorphisme dans les graphes.

1.2.1 Isomorphisme de graphes

Le problème d'*isomorphisme de graphes (IG)* revient à déterminer si la structure de deux graphes est la même si on ne tient pas compte de l'étiquetage des sommets.

Définition 30 (Isomorphisme de graphes (IG))
Deux graphes $G_1 = (V_1, E_1)$ et $G_2 = (V_2, E_2)$ sont ***isomorphes****, dénoté par $G_1 \sim G_2$, s'il existe une bijection $\phi : V_1 \to V_2$ telle que, pour chaque paire de sommets $\{v_i, v_j\} \subset V_1$, on a $(v_i, v_j) \in E_1$ si $(\phi(v_i), \phi(v_j)) \in E_2$.*

Selon les objectifs, on peut être amené à chercher si un motif précis représenté par un graphe est contenu dans un autre graphe. Dans ce cas, les graphes ne sont plus comparés dans leur intégralité mais on cherche s'il existe une occurrence du graphe représentant le motif recherché dans le graphe cible. On parle alors d'*isomorphisme de sous-graphes (ISG)*.

Définition 31 (Isomorphisme de sous-graphes (ISG))
Un graphe $G = (V, E)$ est isomorphe à un sous-graphe S' du graphe $G' = (V', E')$, s'il existe une injection $\phi : V \to V'$, telle que, pour chaque paire de sommets $\{v_i, v_j\} \subset V$, si $(v_i, v_j) \in E$ alors $(\phi(v_i), \phi(v_j)) \in E'$.

L'isomorphisme de graphes et de sous-graphes ont fait l'objet d'une recherche intensive depuis les quatre dernières décennies dans des domaines d'applications variés tels que par exemple la vision par ordinateur [81], la chimie [143], la vidéo [57] ou encore la maintenance de code informatique [73, 17].

Un nombre important de résultats issus de la théorie des graphes concerne la complexité des problématiques d'IG et d'ISG [127]. Il a été prouvé que le problème d'ISG est NP-complet, c'est à dire que tous les algorithmes qui garantissent une solution optimale sont exponentiels dans le pire des cas. La complexité du problème d'IG est encore un problème ouvert : il est établi que le problème général appartient à la classe NP. Toutefois, aucune démonstration à ce jour ne permet d'affirmer que l'isomorphisme de graphes peut être résolu de façon déterministe en temps polynomial, et à l'inverse il n'est pas non plus prouvé qu'une telle solution n'existe pas [104]. Ainsi, on ne sait pas encore si le problème général est NP-complet ou non.

En général, les algorithmes standards pour détecter l'IG ou l'ISG ont une forte complexité de calcul. Cependant, la recherche a permis d'énumérer un certain nombre de classes de graphes particuliers pour lesquelles le problème d'IG peut être résolu par un algorithme en temps polynomial par rapport à la taille de l'instance n. Ainsi, on peut citer les graphes planaires pour lesquels Hopcroft et Tarjan ont proposé un algorithme en temps linéaire ($O(n)$) [62] ; pour les graphes dont le degré maximal est borné par k, Lucks a énoncé dans [83] une résolution en temps polynomial : $O(n^{c.k\log(k)})$, avec c une constante. Une méthode de complexité $O(n^3\log(n))$ a été proposée par Lucks et Hoffman dans [84] pour les arbres triconnectés. En ce qui concerne la résolution du problème d'IG dans le cas général, l'algorithme le plus efficace à l'heure actuelle à notre connaissance a été présenté dans [9] par Babai et Lucks. Sa complexité est exponentielle par rapport à la taille de l'instance et s'élève à $O(\exp^{n^{\frac{1}{2}+c}})$, avec c une constante.

1.2.1.1 Backtracking

L'un des algorithmes les plus connus pour détecter l'isomorphisme de graphes ou de sous-graphes est décrit pour la première fois par Corneil et Gotlieb [35]. L'algorithme est basé sur une méthode de recherche permettant la remise en cause d'une décision par un retour en arrière dans l'espace de résolution du problème, appelée *backtracking*. Le principe de l'algorithme est le suivant : étant donnés un graphe $G = (V, E)$ modèle et un graphe $G' = (V', E')$ en entrée, les sommets de G sont mis en correspondance progressivement les uns après les autres vers les sommets de G'. Après chaque mise en correspondance, on vérifie que la structure de connexion par les arêtes de G est préservée dans G' par la mise en correspondance, *i.e.* si l'isomorphisme de sous-graphes est vérifié. Si tel n'est pas le cas, on annule la dernière mise en correspondance et on cherche un nouvel appariement.

Si pour le $i$ème appariement, l'ensemble des possibilités de mise en correspondance restantes (*i.e.* l'ensemble des $i+1$ème appariements) sont incompatibles avec l'isomorphisme de sous-graphes, alors on annule le $i$ème appariement pour tester une autre paire candidate. On peut ainsi remonter jusqu'au premier appariement et tester toutes les combinaisons possibles.

L'algorithme se termine lorsque tous les sommets de G ont été appareillés, ou lorsque toutes les combinaisons possibles ont été testées. Si, à la fin du processus tous les sommets de G sont mis en correspondance avec succès avec les sommets de G' et si G et G' sont de taille égale ($|V| = |V'|$), alors un isomorphisme de graphes est détecté. Si G contient moins de sommets que G', alors, on a trouvé un isomorphisme de sous-graphes de G vers G'.

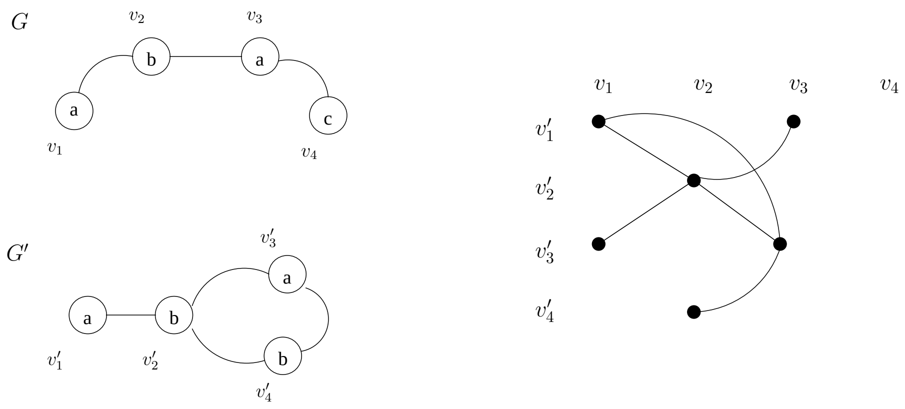
Bien que cette méthode soit performante pour les graphes de petite taille, le nombre d'étapes nécessaires explose en complexité quand les graphes deviennent plus gros. Ainsi, Ullman a proposé dans [127] de combiner le *backtracking* avec une procédure de vérification en amont (*forward-checking*) qui réduit considérablement le nombre d'étapes de remise en cause de décision. L'idée est de tester d'une part si l'ensemble des i appariements déjà effectués représente un isomorphisme de sous-graphes. D'autre part, on teste si à partir de cette configuration courante, les sommets suivants v_k de G ($k > i$) ont au moins un candidat potentiel restant dans G'. Si l'une des deux conditions n'est pas vérifiée, alors on annule le dernier appariement pour tester une nouvelle configuration.

1.2.1.2 Clique maximale

Une autre approche classique pour la détection d'IG et d'ISG est décrite pour la première fois par Borrow et Burstall dans [22] et reprise ensuite dans [63, 117]. La méthode est basée sur la construction d'un graphe appelé *graphe d'associations*. Chaque sommet dans le graphe d'associations décrit une mise en correspondance cohérente (au vu des attributs des sommets, le cas échéant). Les arêtes représentent ensuite les paires de mises en correspondance compatibles (du point de vue structurel).

On peut imaginer différentes contraintes pour la cohérence de mise en correspondance. Par exemple, on peut considérer tout simplement l'étiquette des sommets si le graphe est étiqueté (*e.g.* le type de l'atome représenté par chaque sommet dans une molécule chimique) et ne permettre une mise en correspondance qu'entre des sommets de même étiquette. On peut imaginer de calculer une distance entre chaque paire de sommets basée sur des paramètres intrinsèques et/ou extrinsèques. Dans ce cas, une mise en correspondance est considérée comme cohérente si la distance entre les sommets ne dépasse pas un certain seuil. On établit ainsi la liste des paires de sommets que l'on considère comme cohérentes, et on associe à chacune de ces paires un sommet dans le graphe d'associations.

Les mises en correspondance compatibles deux à deux sont elles, déterminées par la structure des graphes. Ainsi, un même sommet ne peut pas être mis en correspondance avec plusieurs sommets. D'autre part, deux paires de sommets $\{u, v\}$ et $\{u', v'\}$ ne sont compatibles deux à deux que s'il existe les mêmes relations d'adjacence entre les sommets u et v et entre les sommets u' et v'.



(a) (b)

Fig. 1.7: Approche par recherche de clique maximale : (a) les graphes à comparer et (b) le graphe d'associations.

Prenons l'exemple des graphes G et G' étiquetés de la figure 1.7.a. Dans cet exemple, les étiquettes représentent le type des sommets. On considère qu'une mise en correspondance d'un sommet v de G avec un sommet v' de G' est localement cohérente si v et v' ont le même type, et donc la même étiquette. Par exemple, la mise en correspondance des sommet $\{v_1, v_1'\}$ est cohérente, un sommet représentant cet appariement est donc ajouté au graphe d'associations. Par contre, la paire $\{v_1, v_2'\}$

n'est pas cohérente, on n'ajoute pas de sommets correspondant à cet appariement. On obtient alors l'ensemble des sommets du graphe d'associations de la figure 1.7.b.

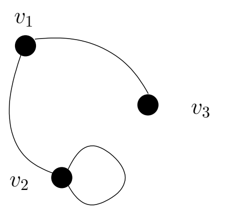
Ensuite, deux mises en correspondance sont considérées comme étant compatibles si la structure de connexions entre les sommets est conservée. Par exemple, les paires de mise en correspondance $\{v_1, v'_1\}$ et $\{v_2, v'_2\}$ sont compatibles car la structure d'arêtes est conservée, il existe donc une arête entre les sommets représentant les paires $\{v_1, v'_1\}$ et $\{v_2, v'_2\}$ dans le graphe d'associations (Fig. 1.7.b). Les paires $\{v_1, v'_1\}$ et $\{v_2, v'_4\}$ ne sont pas compatibles car il y a une arête $e = \{v_1, v_2\}$ dans G et les sommets v'_1 et v'_4 ne sont pas connectés dans G'. Ainsi, les sommets correspondants aux paires $\{v_1, v'_1\}$ et $\{v_2, v'_4\}$ ne sont pas reliés dans le graphe d'associations.

La clique (Déf. 12) maximale dans le graphe d'associations identifie le plus grand sous-ensemble de mises en correspondance qui est cohérent et par conséquent le plus grand sous-graphe commun. Bien que le nombre de sommets du graphe d'associations est réduit par rapport aux graphes d'entrée, le problème de recherche de la clique maximale est connu pour être NP-complet.

1.2.1.3 Étiquette canonique

Parmi les approches classiques, on peut également citer les approches basées sur le calcul d'une *étiquette canonique* [10]. Étant donné un ordre lexicographique total sur les sommets (en considérant par exemple leur étiquette), on cherche la permutation des sommets qui produit le plus petit mot. L'étiquette canonique correspond alors à ce plus petit mot. Si deux graphes ont la même étiquette canonique, alors ils sont isomorphes.

Prenons par exemple la matrice d'adjacence M pour représenter un graphe G (Déf. 28). À partir de M, on forme un mot composé des lignes de la matrice concaténées de haut en bas. Sur l'exemple de la figure 1.8, le mot correspondant à la matrice d'adjacence (b) est 011 110 100. L'étiquette canonique du graphe (a) correspond au plus petit mot (au sens de l'ordre lexicographique usuel) que l'on peut former à partir de permutations des sommets dans la matrice d'adjacence (l'ordre des sommets en ligne doit être identique à l'ordre des sommets en colonne). Dans cet exemple, la matrice de la figure 1.8.c produit le plus petit mot 001 011 110. L'étiquette canonique du graphe exemple est donc 001 011 110.



(a)

	v_1	v_2	v_3
v_1	0	1	1
v_2	1	1	0
v_3	1	0	0

(b)

	v_3	v_2	v_1
v_3	0	0	1
v_2	0	1	1
v_1	1	1	0

(c)

Fig. 1.8: Etiquette canonique d'un graphe.

Le nombre de permutations des sommets possibles s'élève à $n!$ et cet algorithme reste acceptable dans le cas de comparaisons de graphes de petite taille. Cependant, le problème de recherche d'isomorphisme de sous-graphes est plus complexe : il est nécessaire de trouver le plus grand sous-ensemble des sommets des deux graphes à comparer qui vérifie l'isomorphisme. Le nombre de possibilités dans ce cas est exponentiel.

Il existe d'autres approches intéressantes, définies sur les structures arborescentes, pour la mise en correspondance.

1.2.1.4 L'approche par classes d'équivalence

L'approche par classes d'équivalence a été initialement proposée par Zemlyachenko [146] pour l'isomorphisme de sous-arbres puis reprise ensuite par Dinitz *et al.* [42]. Le principe est le suivant : les sommets (correspondant à la racine d'un sous-arbre de l'arbre), sont répartis en classes d'équivalence (*i.e.* en groupes de sommets partageant les mêmes propriétés). La classification des sommets peut alors être utilisée pour détecter l'isomorphisme ou le quasi-isomorphisme de deux arbres.

La construction des classes d'équivalence est effectuée en utilisant une variante du principe d'étiquetage canonique décrit dans la section précédente. Dans cette approche, l'étiquette canonique d'un sous-arbre correspond au plus petit mot formé par les indices des classes des fils de la racine (ordre lexicographique usuel). Si deux sous-arbres ont une même étiquette canonique, alors ils appartiennent à la même classe d'équivalence.

Les affectations des indices de classes sont établies depuis les feuilles de l'arbre vers la racine. À l'initialisation, chaque feuille se voit attribuer l'indice 1, correspondant à la première classe d'équivalence (voir l'exemple de la figure 1.9.a).

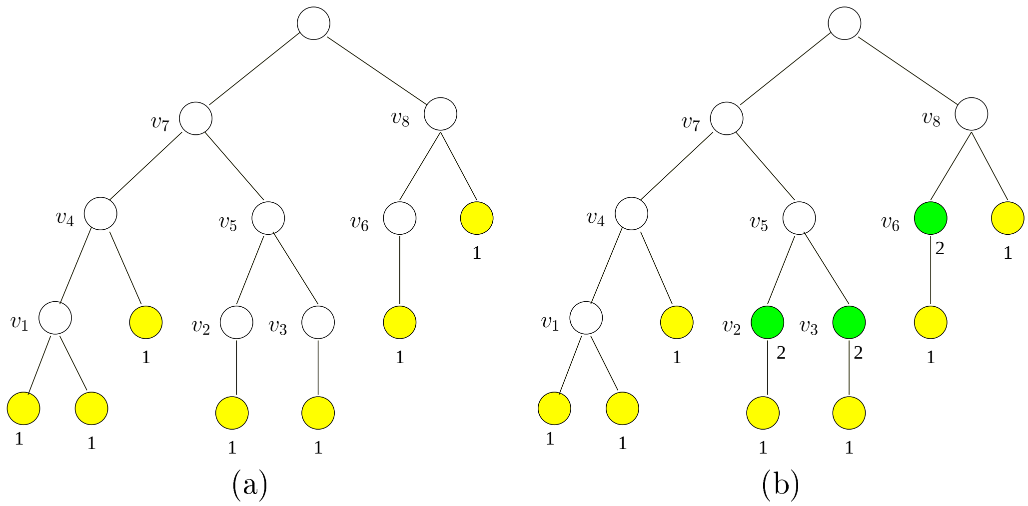


FIG. 1.9: Construction des classes d'équivalence. (a) Initialisation et (b) première itération.

Par la suite, le processus est itératif : pour chaque sommet v non encore classé, tel que l'ensemble des fils de v appartiennent à une classe d'équivalence, on calcule l'étiquette canonique $c(v)$. On note C l'ensemble des étiquettes distinctes obtenues. Sur l'exemple de la figure 1.9.a, seuls les sommets v_1, v_2, v_3 et v_6 ont l'ensemble de leurs fils déjà classés. Les étiquettes canoniques associées sont 1 pour les sommets v_2, v_3 et v_6 et 11 pour le sommet v_1. Dans ce cas, $C = \{1, 11\}$.

On construit désormais la nouvelle classe d'équivalence. Soit c_{min} la plus petite étiquette de l'ensemble C. Soit i_{max} l'indice maximal de classe d'équivalence de l'arbre. L'ensemble des sommets dont l'étiquette canonique est égale à c_{min} constitue une nouvelle classe d'équivalence dont l'indice est fixé à $i_{max} + 1$. Dans l'exemple de la figure 1.9.a, les sommets v_2, v_3 et v_6 portent la plus petite étiquette canonique $c_{min} = 1$, on leur affecte donc l'indice de classe 2 et on obtient la classification de la figure 1.9.b.

À l'itération suivante, on calcule les étiquettes canoniques : v_1 a pour étiquette 11, v_5 a pour étiquette 22 et v_8 a pour étiquette 12. Le sommet v_1 qui porte la plus petite étiquette à cette itération constitue la classe d'équivalence 3 (Fig. 1.10).

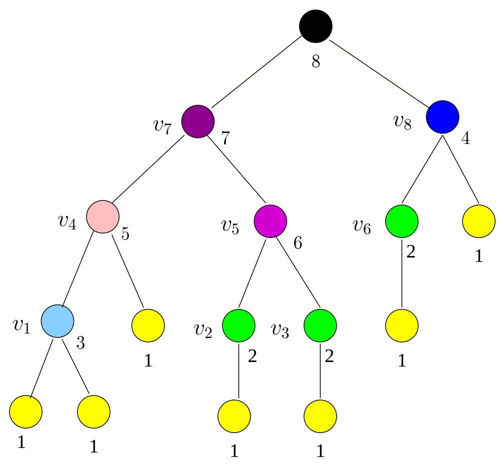


FIG. 1.10: Classes d'équivalence à la fin du processus.

Pour vérifier que l'arbre T_1 est isomorphe à l'arbre T_2, on construit un arbre T dont la racine contient exactement deux fils : la racine de T_1 et celle de T_2. Les arbres T_1 et T_2 sont isomorphes, si et seulement si leurs racines respectives appartiennent à la même classe d'équivalence.

Cette approche par classes d'équivalence a l'avantage de lister la totalité des isomorphismes de sous-arbres qu'il existe dans un arbre. Toutefois, cette méthode est définie pour les arbres et non pour les graphes.

Les techniques exhaustives décrites ci-dessus, qui cherchent à trouver la solution optimale globale au problème d'IG et d'ISG souffrent des contraintes de complexité élevée. D'autre part, ces techniques sont inadaptées à la recherche de quasi-isomorphisme, *i.e.* la mise en correspondance avec tolérance d'erreur.

1.2.2 Quasi-isomorphisme de graphes

Pour éviter le problème de complexité mentionné ci-dessus, de nombreuses méthodes d'approximation ont été explorées. Dans ce contexte, on cherche un *quasi-isomorphisme de graphes (QIG) ou de sous-graphes (QISG).*

Il existe deux approches au quasi-isomorphisme : la première concerne les algorithmes appelés algorithmes à correction d'erreur, ou à tolérance d'erreur qui sont utilisés pour trouver la solution optimale dans le cas de graphes perturbés ou qui contiennent des données imparfaites. Ces techniques qui assurent une solution optimale requièrent un temps exponentiel dans le pire des cas. La seconde approche désigne les méthodes d'approximation. Ces techniques sont également adaptées dans le cas où les graphes sont perturbés. Toutefois, elles ne garantissent pas toujours la solution optimale, mais elles ont l'avantage d'une résolution en temps polynomial.

Nous présentons dans cette section un aperçu des principales approches que l'on trouve dans la littérature pour la résolution du quasi-isomorphisme de graphes.

1.2.2.1 Les méthodes à tolérance d'erreur

Les méthodes à tolérance d'erreur sont basées sur une comparaison structurelle par le calcul d'une distance d'édition de graphes [81, 91]. L'idée est de déterminer la plus petite séquence d'opérations d'édition qui transforme un graphe en l'autre parmi la substitution, l'insertion ou la suppression par exemple. Il est classique d'associer un coût à chaque opération d'édition considérée : ainsi, il est possible de pondérer différemment chaque opération d'édition, selon l'importance qu'on lui associe. La définition de ces fonctions de coût est fortement dépendante de l'application.

Un QISG est donc défini comme la séquence d'opérations d'édition de coût minimal qui peut être appliquée à l'un des graphes pour qu'un isomorphisme de sous-graphes existe.

La plupart des algorithmes pour trouver la solution optimale sont basés sur l'algorithme de recherche A* [102]. Étant donnés un graphe $G = (V, E)$ et un graphe $G' = (V', E')$, un arbre de recherche est progressivement construit. Chaque sommet dans l'arbre correspond à une mise en correspondance partielle des sommets de G vers les sommets de G'. Au départ, le premier sommet de G est mis en correspondance avec chaque sommet de G'. Il en résulte $|V'|$ sommets issus de la racine de l'arbre de recherche et ainsi $|V'|$ chemins possibles à cette étape. La génération des étapes suivantes est ensuite guidée par le coût des opérations d'édition. C'est à dire que la mise en correspondance avec le coût le plus faible est ajoutée. C'est en quelque sorte un algorithme de recherche du plus court chemin. La première solution trouvée telle que tous les sommets de G sont mis en correspondance représente une solution optimale par rapport au coût d'édition, c'est à dire l'une des meilleures mise en correspondance entre les deux graphes.

Les performances d'un tel algorithme dépendent fortement du nombre d'états explorés lors de la construction de l'arbre de recherche qui peut être exponentiel dans le pire des cas. Des heuristiques permettant d'estimer la distance qui sépare un sommet de l'arbre de recherche à une feuille ont été proposées dans le but de réduire la taille de l'arbre de recherche [116].

1.2.2.2 La relaxation probabiliste

L'une des approches d'approximation les plus communes concerne la relaxation probabiliste [111, 68, 140]. Le principe de ces méthodes est le suivant : à chaque appariement possible est associée une probabilité que l'appariement soit cohérent pour la mise en correspondance. Cette notion de probabilité d'appariement peut être vue comme une mesure de similarité entre les sommets : plus les sommets sont similaires (au vu des caractéristiques définies pour l'application), plus la probabilité qu'ils soient mis en correspondance est élevée. Ainsi, à l'initialisation, on peut utiliser une mesure de similarité entre les sommets basée sur des attributs extrinsèques et/ou intrinsèques aux sommets. Le processus est itératif. À chaque itération, on réévalue la probabilité que deux sommets soient mis en correspondance en fonction de la cohérence de leur voisinage local. On utilise ainsi l'information sur le contexte local pour faire croître ou décroître la mesure de similarité entre les deux sommets. Il est clair que si l'ensemble des voisins d'un sommet u ont été mis en correspondance avec l'ensemble des voisins d'un sommet v, la probabilité que u corresponde à v est élevée puisque leur contexte local est cohérent. Ainsi, une paire de sommets pour laquelle la décision d'appariement n'était pas triviale peut voir sa probabilité d'appariement augmenter ou diminuer au fur et à mesure des itérations, levant ainsi l'ambiguïté.

Une technique d'accélération du processus itératif a été proposée en [57]. Une mise en correspondance définitive est effectuée dès lors qu'un appariement de sommets est jugé suffisamment cohérent (si la mesure de similarité dépasse un certain seuil et qu'elle est maximale pour les deux sommets). Ainsi, lors d'un appariement définitif, la probabilité pour que les sommets appareillés correspondent aux autres sommets des graphes est fixée à zéro.

Nous avons proposé une méthode basée sur les techniques de relaxation, pour la reconnaissance d'objets dans les vidéos à faible résolution [31]. L'algorithme complet est décrit dans le chapitre 4.

1.2.2.3 L'approche par hachage

Nous trouvons également dans la littérature des méthodes d'approximation basées sur un principe de hachage des sous-structures similaires [73, 17] pour la détection de code similaire dans un programme. Ces méthodes sont appliquées sur des arbres. L'idée est de regrouper dans une même catégorie les sous-arbres similaires via une fonction de hachage. Le principe d'une fonction de hachage est d'attribuer, à chaque élément e, un code de hachage $h(e)$. Ce code est tel que si deux éléments a et b vérifient $h(a) \neq h(b)$, alors $a \neq b$. Si par contre $h(a) = h(b)$ alors a et b sont des candidats potentiels pour la mise en correspondance.

Pour déterminer si deux sous-arbres appartiennent à la même catégorie, la fonction de hachage utilise des vecteurs caractéristiques des sous-arbres. Les caractéristiques utilisées peuvent concerner, par exemple, les occurrences d'un certain type de sommets [73], leur étiquette, des informations structurelles (degrés, taille du sous-arbre, ...) ou alors réutiliser les codes de hachage déjà calculés sur les sommets contenus dans le sous-arbre comme dans [17].

Dans [73], l'objectif est comparable à l'approche par classes d'équivalence (voir section 1.2.1.4), à savoir, deux sous-arbres de même code de hachage sont considérés comme similaires. Baxter [17] utilise une approche plus complète : la fonction de hachage est utilisé dans le but de limiter les comparaisons entre les sous-arbres. Le hachage des données est utilisée comme un filtre pour éviter de comparer des arbres qui ne peuvent pas se correspondre.

Jiang *et al.* [73] adoptent une technique à l'image de l'approche par classes d'équivalence décrite dans la section 1.2.1.4, à savoir, deux sous-arbres portant la même étiquette sont considérés comme similaires. Pour Baxter *et al.*, cet étiquetage est utilisé comme pré-traitement à l'étape de comparaison des sous-arbres. Dans leur approche, on effectue une phase de raffinement en comparant deux à deux les sous-arbres contenus dans une même famille de hachage. Pour ce faire, ils calculent une distance entre les deux sous-arbres qui prend en compte l'étiquetage de hachage. La distance $d(T, T')$, entre deux arbres T et T' correspond au rapport entre les sommets communs à T et T' et le nombre total de sommets de T et T'. On a :

$$d(T, T') = \frac{|T \cap T'|}{|T \cup T'|}$$

où $T \cap T'$ désigne l'ensemble des sommets communs à T et T' en terme d'étiquette de hachage et $T \cup T'$ désigne l'ensemble des sommets de T et de T'.

Deux sous-arbres sont alors considérés comme similaires si la distance d est inférieure à un certain seuil.

Nous avons présenté dans cette section un ensemble de méthodes pour la résolution de l'isomorphisme et du quasi-isomorphisme dont la plupart sont généralisées aux graphes et aux sous-graphes. Nous avons aussi mentionné quelques méthodes intéressantes restreintes aux structures arborescentes.
Nous proposons dans ce manuscrit, une nouvelle méthode pour la résolution du quasi-isomorphisme de graphes et de sous-graphes. La méthode heuristique proposée est décrite dans le chapitre 2. La méthode reprend le principe des deux approches appliquées aux arbres (Sec. 1.2.1.4 et 1.2.2.3) étendu aux graphes.

1.3 Génération aléatoire de graphes

La génération aléatoire de structures de graphes pour la modélisation des réseaux et structures arborescentes réels est depuis longtemps explorée. Le premier modèle de génération aléatoire de graphes a été proposé par Erdös et Rényi en 1959 [46] pour modéliser les réseaux complexes. Dans leurs travaux, Erdös et Rényi admettent que les réseaux réels qu'ils modélisent sont des réseaux complètement aléatoires.

En 1988, Reffye *et al.* [39] ont proposé un modèle de génération aléatoire générique paramétrable permettant, en modifiant les paramètres, de construire des structures aux propriétés différentes selon le paramétrage. Le modèle est évolutif (ajouts successifs de sommets) et intègre les connaissances en botanique sur l'architecture des arbres (croissance, occupation de l'espace, emplacement des feuilles, fleurs et fruits, *etc.*). En fonction de ces propriétés, à une étape de la génération, chaque sommet terminal de la structure d'arbre est considéré comme un "bourgeon" de la plante modélisée qui peut évoluer de différentes façons : devenir une fleur et mourir, se ramifier en sous-branches générant ainsi la création de nouveaux "bourgeons", rester dans l'état actuel ou encore mourir.

Viennot *et al.* ont proposé en 1989 une approche combinatoire à la génération aléatoire de telles structures [131]. L'idée est d'utiliser des mesures combinatoires telles que le nombre de Strahler (Déf. 29, section 1.1) pour "évaluer" la forme de l'arbre, ou d'un motif ramifié. À partir de ces informations combinatoires, une matrice dite *matrice de ramification* est construite pour définir les caractéristiques de chacun des modèles. La génération aléatoire consiste alors à construire des arbres combinatoires binaires utilisant la matrice de ramification pour prendre une décision sur l'évolution de chaque sommet terminal (*i.e.* un "bourgeon" de la plante).

En 1999, Kleinberg *et al.* [76] d'une part, et Barabási et Albert [12] d'autre part, étudient en parallèle la structure d'un échantillon du graphe du Web. Les deux groupes de chercheurs observent, contre toute attente, que le graphe échantillon ne correspond pas à un graphe aléatoire tel que l'ont défini Erdös et Rényi 40 ans plus tôt. En effet, une grande majorité des sommets dans le graphe ont très peu de liens, et une minorité de sommets ont quant à eux un très grand nombre de liens. On appelle ces sommets très fortement connectés des *hubs*. Dans de tels graphes, baptisés *graphes sans échelle* par Barabási et Albert dans [12], la distribution des degrés suit une loi en puissance (ou loi de Zipf [148]). Le modèle aléatoire de Erdös et Rényi, quant à lui, ne reproduit pas cette propriété : les sommets ont approximativement le même nombre de liens, la répartition des dégrés suit un loi de Poisson.

Définition 32 (Graphe sans échelle [12])
On appelle graphe sans échelle un graphe dont la distribution des degrés suit une loi en puissance.

Par la suite, de nombreuses publications ont montré que cette propriété non triviale de répartition spécifique des degrés était vérifiée pour de nombreux réseaux réels. On peut citer notamment les travaux de Newman [98, 99] dans lesquels l'auteur propose une étude approfondie des propriétés des réseaux de collaboration scientifiques. Dans [100], outre l'étude de la distribution des degrés sur de nombreux réseaux réels, Newman propose une analyse du coefficient de clustering de ces réseaux particuliers. D'autre part, il introduit la notion de *motifs mixtes* (mixing patterns) qui détermine la tendance des sommets dans un réseaux à se connecter aux sommets existants qui leur sont similaires (ou au contraire différents) [101] (par exemple, les scientifiques d'un même domaine se citent entre eux).

L'ubiquité apparente d'une telle distribution des degrés dans la plupart des réseaux naturels ou systèmes créés par l'homme a naturellement imposé les graphes *sans échelle* comme modèle représentatif des systèmes complexes dans de nombreux domaines. Ce tournant dans l'étude interdisciplinaire des réseaux complexes : la découverte, l'étude, l'application des graphes sans échelle est qualifiée de "nouvelle science des réseaux" [11].

Cette nouvelle approche de la science des réseaux qui s'est développée dans les dernières années se distingue des travaux précédents en trois points importants :

- elle se base sur une étude statistique des propriétés des réseaux réels,
- elle considère que les réseaux ne sont pas statiques, mais qu'ils évoluent dans le temps selon différentes règles dynamiques,
- elle a pour objectif de comprendre les réseaux au-delà de la simple considération d'objets topologiques : on cherche à comprendre le procédé par lequel ces systèmes distribués dynamiques sont construits.

Pour développer de nouveaux modèles théoriques de graphes qui prennent en considération les caractéristiques structurelles des réseaux réels, il est nécessaire d'être capable de définir les propriétés structurelles spécifiques à ces graphes réels. Par conséquent, des données empiriques sont indispensables pour définir ces modèles théoriques. Ainsi, à l'image de la science traditionnelle, où les expériences servent la théorie, les nouvelles approches dans l'étude des réseaux sont basées à la fois sur les observations empiriques et la modélisation de celles-ci [76, 12].

Dans le cadre de cette thèse, notre étude porte sur la structure arborescente induite par l'organisation du système de fichiers utilisateur. Nous avons suivi la même démarche, maintenant générique, pour étudier et modéliser ces structures : d'abord les observations de données empiriques puis la construction d'un modèle théorique basé sur ces observations. L'étude statistique des caractéristiques, détaillée dans le chapitre 3, suggère que de telles arborescences vérifient entre autres, les propriétés de graphe sans échelle (Déf. 32).

Dans cette section, nous décrivons les principaux modèles de génération aléatoire de graphes sans échelle existants. L'étude portant sur les systèmes de fichiers est présentée dans le chapitre 3, dans lequel nous proposons un modèle pour la génération aléatoire de structures arborescentes.

1.4 Modèles de génération aléatoire

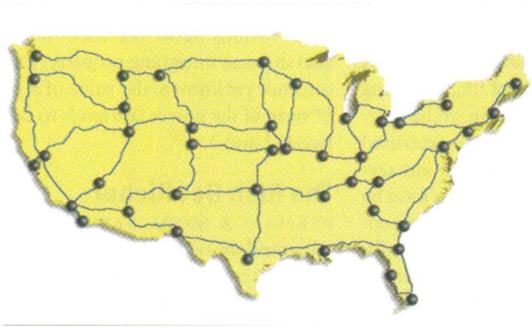
La nécessité de comprendre et de travailler sur les réseaux réels a motivé la communauté scientifique à proposer des modèles de génération aléatoire permettant de simuler la construction de telles structures. Le lecteur pourra se référer aux travaux de Virtanen [132] qui proposent une revue des principaux modèles de génération aléatoire destinés à produire des graphes qui ressemblent aux réseaux naturels.

Dans cette section, nous présentons quelques uns des plus importants modèles de génération aléatoire de structures de graphe. Dans un premier temps, nous décrivons le modèle aléatoire de Erdös et Rényi [46] qui a longtemps été admis comme modèle de référence pour générer des réseaux complexes. Nous nous intéressons par la suite aux modèles de génération de graphes plus spécifiques : les structures sans échelle, dont les arborescences de fichiers font partie.

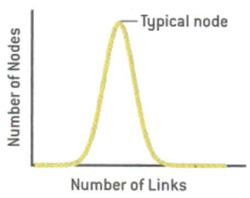
1.4.1 Modèle de Erdös-Rényi : graphe aléatoire

Pendant plus de 40 années, la science a considéré les réseaux complexes comme étant des réseaux complètement aléatoires. L'étude des réseaux aléatoires a été très largement dominée par le modèle proposé par Paul Erdös et Alfréd Rényi [46] qui considèrent les réseaux comme un ensemble de sommets reliés deux à deux à probabilité égale. Le principe du modèle de Erdös-Rényi peut être abordé de deux façons différentes. Soit on considère dans le graphe l'ensemble des arêtes possibles sur lesquelles on définit un ordre aléatoire et on tire les m premières arêtes, c'est le modèle $G_{n,m}$. Soit on considère que chaque arête possible dans le graphe est présente avec une probabilité p, indépendamment des autres. On parlera alors dans ce cas du modèle $G_{n,p}$. Dans les deux cas, le graphe obtenu possède les mêmes propriétés de répartition des degrés, à savoir une répartition équitable caractérisée par une loi de Poisson. Ainsi, malgré le placement aléatoire des liens, le graphe résultant est relativement uniforme : la majorité des sommets ont approximativement le même nombre de liens.

Un graphe aléatoire tel que défini par Erdös et Rényi a une structure qui ressemble au réseau auto-routier simplifié des États-Unis (Fig. 1.11.a), comme l'illustre Barabási dans un article de vulgarisation paru dans *Scientific American*. En faisant abstraction de l'aspect sémantique géographique des sommets qui représentent les grandes villes des États-Unis, on remarque que le graphe est particulièrement bien équilibré, avec une répartition des liens entre les sommets quasi-uniforme.



(a)



(b)

Fig. 1.11: Exemple d'un réseau uniforme : (a) le système auto-routier simplifié des États-Unis, (b) la distribution des degrés (loi de Poisson). © A.-L. Barabási [15]

Dans un tel système, la distribution des degrés des sommets suit une loi de Poisson (Fig. 1.11.b) avec une grande majorité de sommets qui ont approximativement le même nombre de liens et il existe une très faible minorité de sommets qui ont un degré très inférieur ou très supérieur à la moyenne. Ces graphes sont également appelés graphes exponentiels, parce que la probabilité qu'un sommet soit connecté à k autres sommets décroît exponentiellement pour k grand.

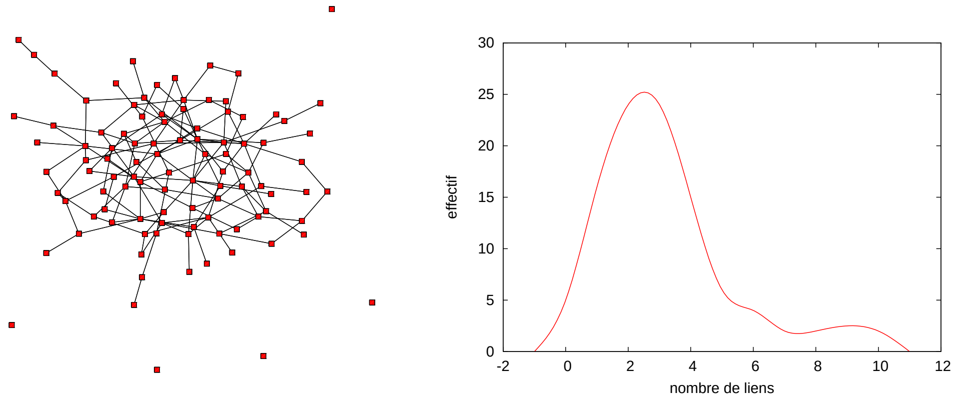
Dans les deux paragraphes qui suivent, on définit formellement les modèles de génération $G_{n,m}$ et $G_{n,p}$, qui sont les deux approches essentielles à la construction d'un graphe aléatoire Erdös-Rényi.

Le modèle $G_{n,m}$

Le principe de construction d'un graphe aléatoire de Erdös-Rényi suivant le modèle $G_{n,m}$ consiste à définir dans un premier temps le graphe G à n sommets dénué d'arêtes. On établit ensuite les connexions entre les sommets. Le paramètre m fixe le nombre de liens qui seront présents dans le graphe G : on tire aléatoirement uniformément m arêtes parmi l'ensemble des arêtes possibles sur le graphe G. On considère que le graphe G est un graphe simple (Déf. 5), et par conséquent, ni les boucles, ni les multi-arêtes ne sont considérées comme étant des arêtes possibles.

Par ce procédé, le choix des arêtes est équiprobable et par conséquent, le nombre de liens issus de chacun des sommets du graphe est proche de la moyenne $\frac{2m}{n}$. On note que le graphe obtenu n'est pas nécessairement connexe.

La figure 1.12.a montre un exemple de génération aléatoire par le procédé $G_{n,m}$ tel que $n = 100$ et $m = 150$. La distribution des degrés dans le graphe obtenu est donné par la courbe de la figure 1.12.b qui a une "allure" de courbe de Gauss.



(a) (b)

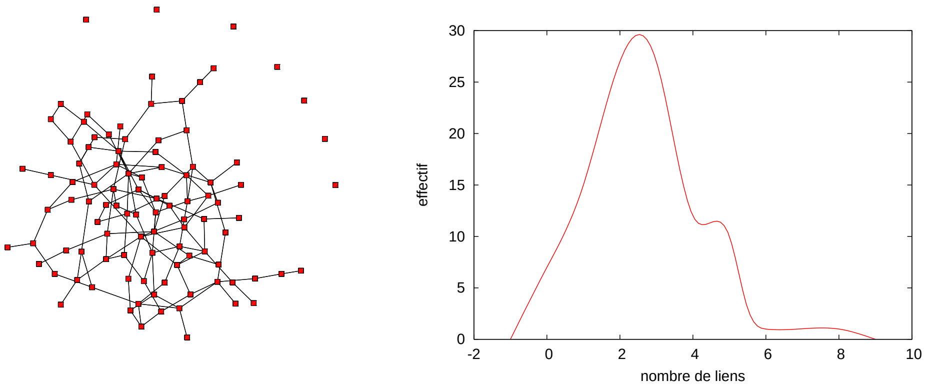
FIG. 1.12: Modèle $G_{n,m}$: (a) graphe aléatoire avec $n = 100$ et $m = 150$, (b) la distribution des degrés (courbe de Gauss).

Le modèle $G_{n,p}$

La seconde technique pour générer aléatoirement des graphes correspondant au modèle de Erdös-Rényi consiste à utiliser le processus de génération $G_{n,p}$. Le nombre de sommets du graphe G obtenu est fixé à n. Le principe diffère du générateur $G_{n,m}$ dans le sens où le nombre de liens n'est pas fixé *a priori*. Ici, la présence de chaque arête est testée selon le paramètre p qui désigne la probabilité qu'une arête e soit présente dans le graphe résultat.

Le choix de la présence de chacune des arêtes possibles du graphe G est équiprobable. Le degré des sommets du graphe est alors en moyenne $\frac{n(n-1)}{2}p$. On note que les deux modèles $G_{n,m}$ et $G_{n,p}$ produisent des graphes similaires pour $p = \frac{4m}{n^2(n-1)}$.

La figure 1.13.a montre un exemple de génération aléatoire par le procédé $G_{n,p}$ tel que $n = 100$ et $p = 0.03$. La distribution des degrés dans le graphe obtenu est ici encore une Gaussienne (Fig. 1.13.b).



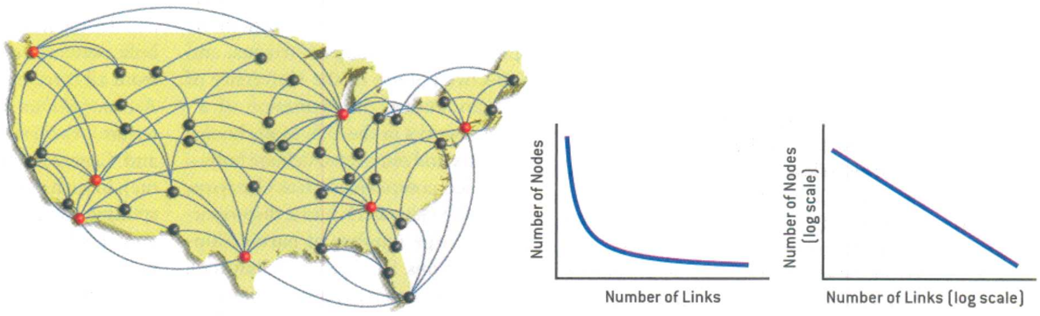
(a) (b)

FIG. 1.13: Modèle $G_{n,p}$: (a) graphe aléatoire avec $n = 100$ et $p = 0.03$, (b) la distribution des degrés (courbe de Gauss).

1.4.2 Modèle de Barabási-Albert : graphe sans échelle

Pendant de nombreuses années, les réseaux complexes ont été décrits par la théorie de Erdös et Rényi [46]. En l'absence de données sur les grands réseaux, les prédictions de Erdös et Rényi étaient alors difficilement vérifiables pour les structures issues des données réelles. En 1999, les travaux de Kleinberg [76] et de Barabási et Albert [12] ont révélé l'existence d'un degré élevé d'organisation des individus caractérisant les réseaux complexes à grande échelle. L'exploration de larges bases de données telles que le graphe du Web, ou le graphe des citations d'articles scientifiques ont permis de constater que, indépendemment du système et de l'identité de ses composants, la distribution des degrés suit une loi en puissance : la très grande majorité des sommets sont de faible degré et il existe quelques sommets de degré très élevé : les *hubs*. Les graphes qui respectent une telle distribution sont dits *sans échelle*.

La figure 1.14.a montre un exemple de réseau sans échelle : le réseau aérien simplifié des États-Unis. Dans ce réseau, on remarque que certaines très grandes villes telles que New York, Chicago, Washington, Houston, *etc.* sont de réels pivots dans le réseau avec un très grand nombre de lignes aériennes qui transitent par leur aéroport. La plupart des autres villes sont assez peu desservies avec environ 2 à 3 destinations : la distribution des degrés de ce réseau suit une loi en puissance (Fig. 1.14.b).



(a) (b)

Fig. 1.14: Exemple d'un réseau sans échelle : (a) le système aérien simplifié des États-Unis, (b) la distribution des degrés (loi en puissance). © A.-L. Barabási [15]

1.4.2.1 L'abondance de graphes sans échelle

La découverte de la propriété sans échelle des graphes ne cesse d'être vérifiée depuis, et ce, pour un très large éventail de réseaux d'interactions. Les travaux de Sydney Redner ont permis d'établir que le graphe des citations d'articles scientifiques de la revue *Physical Review* est un graphe sans échelle. Dans d'autres domaines, on peut citer le graphe de partenariat industriel des industries de biotechnologie aux États-Unis [108], le réseau cellulaire métabolique [14] qui vérifient eux aussi les mêmes propriétés de distribution des degrés. Autant d'exemples dans des domaines aussi variés que la génétique, l'informatique, la sociologie, qui poussent à s'interroger : comment ces systèmes *a priori* si différents les uns des autres ont une même architecture et obéissent aux mêmes lois ?

1.4.2.2 Un processus évolutif : “Les riches de plus en plus riches” [15]

On pourrait s'interroger sur les raisons de l'incapacité du modèle de Erdös et Rényi à reproduire le phénomène de *hubs* (sommets de très fort degré) dans les graphes générés. Un élément de réponse est le suivant : dans leur modèle, Erdös et Rényi considèrent un graphe dont l'inventaire des sommets est connu et fixé sans qu'aucun lien entre eux ne soit défini. Les arêtes sont par la suite ajoutées aléatoirement. Dans la réalité, les structures ne sont pas statiques : les réseaux ont plutôt tendance à s'étendre, des sommets nouveaux venant se greffer à la structure existante avec un certain nombre de liens. Les graphes ont un nombre de sommets *croissant*, qui augmente dans le temps. On parle alors de *processus évolutif*.

Lors de leur étude du graphe du Web, Barabási, Albert et Jeong s'attendaient à ce que le réseau capturé ressemble à un graphe aléatoire tel que l'ont décrit Erdös et Rényi [13]. En effet, leur supposition était la suivante : lorsqu'une personne décide quels sites seront pointés par son document web, elle respecte ses propres centres d'intérêts. Étant donné la diversité des sujets d'intérêts et le nombre de pages web traitant d'un même sujet, on peut supposer que le résultats des choix de connections vers les autres pages web soit assez aléatoire. Cependant, les observations sur les données réelles ont prouvé que tel n'était pas le cas. Ainsi, plus de 80% des pages web ont moins de quatre liens, et une très faible minorité ont plus d'un milliers de liens. En effet, les pages web ne sont pas équivalentes. Lorsque l'on doit choisir vers quelles pages web diriger un lien, on a le choix entre près de quelques milliards de pages. Cependant, la plupart d'entre nous sont familiers avec seulement une très mince portion de la totalité de la toile, et ce sous-ensemble tend à inclure les sites les plus connectés car ce sont les sites les plus faciles à trouver via un moteur de recherche. En ajoutant un nouveau lien vers ces sites populaires, on ne fait que renforcer la popularité de ces derniers au détriment de la grande majorité des sites très peu cités.

Ce processus d'*attachement préférentiel* s'observe dans la plupart des réseaux complexes. Prenons par exemple le graphe *IMDB* (Internet Movie Database) [125] pour lequel les acteurs sont représentés par les sommets, et un lien entre deux acteurs signifie qu'ils ont joué dans un même film. Les acteurs ayant le plus grand nombre de collaborations ont le plus de chance d'être sélectionnés pour de nouveaux rôles. Dans le réseau des industries américaines de bio-technologie, les entreprises de renommée qui ont déjà établi un grand nombre de partenariats sont plus favorables aux propositions de nouvelles collaborations, et augmentent encore de ce fait leur attractabilité pour les éventuels futurs partenaires. Dans la littérature scientifique, plus un article est cité, plus on peut supposer que ce dernier a un impact important dans la société scientifique. Cette popularité incite les chercheurs à lire et à citer à leur tour cet article.

Ces deux mécanismes (*croissance* et *attachement préférentiel*) nous aident à expliquer la présence de *hubs* dans les structures sans échelle. Les nouveaux sommets ont tendance à se rattacher aux sommets déjà fortement connectés et ce, au détriment des autres sommets. Avec un tel scénario, les sommets les plus anciennement apparus dans le graphe sont favorisés et ont le plus de chance de devenir des *hubs* dans le réseau. Ce phénomène rappelle la citation “les riches de plus en plus riches” parfois utilisée pour caractériser les graphes sans échelle.

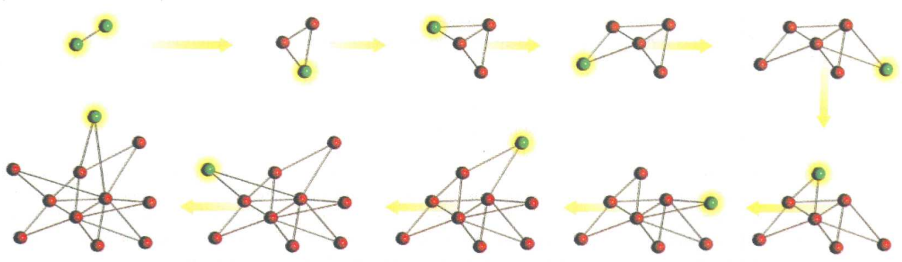


Fig. 1.15: Exemple d'évolution d'un réseau sans échelle. © A.-L. Barabási [15]

La figure 1.15 montre un exemple du processus de croissance avec attachement préférentiel. Le réseau croît incrémentalement de deux à onze sommets : à chaque itération, un sommet est créé. Ce nouveau sommet (symbolisé par la couleur verte dans la figure), lorsqu'il décide d'établir un nouveau lien, préfère se rattacher à un sommet existant (colorié en rouge) dont le nombre de connections est déjà important.

1.4.2.3 Le modèle de génération aléatoire de graphes sans échelle de Barabási-Albert

Nous avons vu que les graphes des réseaux réels obéissent à une même loi en puissance de distribution des degrés. Plus formellement, la propriété observée dans ces structures est la suivante :

– la probabilité $P(k)$ qu'un sommet dans le réseau interagisse avec k autres sommets, suit une loi en puissance du type

$$P(k) \sim k^{-\gamma}, \gamma > 0 \tag{1.2}$$

Il a été observé que généralement, $\gamma \in [2, 3]$ pour les réseaux d'interactions réels [12].

Le modèle de génération aléatoire de Barabási et Albert combine les deux mécanismes décrits dans le paragraphe précédent. Le processus démarre avec un graphe qui contient un petit nombre de sommets n_0. À chaque itération, un nouveau sommet est ajouté avec m arêtes ($m \leq n_0$) qui relient ce sommet à m différents sommets déjà présents dans la structure. Le paramètre m est fixé *a priori*. Ce procédé traduit le caractère d'évolution *croissante* du réseau. Pour introduire la propriété d'*attachement préférentiel*, on suppose que la probabilité Π qu'un nouveau sommet soit connecté à un sommet u du réseau dépend de la connectivité $d(u)$ (Déf. 14) de ce sommet. On a ainsi

$$\Pi(u) = \frac{d(u)}{\sum_v d(v)} \tag{1.3}$$

Après t étapes, le modèle produit un graphe à $n_0 + t$ sommets et mt arêtes.

Le graphe résultat évolue vers un graphe sans échelle tel que la probabilité qu'un sommet possède k liens, suit une loi en puissance avec $\gamma = 2.9 \pm 0.1$ (voir figure 1.16). La loi en puissance est observée pour les systèmes naturels, à plusieurs instants de leur vie (taille différente à des instants différents). Ainsi, cette propriété de loi en puissance est invariante avec le temps, et un bon modèle doit produire une distribution dont les caractéristiques ne varient pas avec le temps. La figure 1.16 montre que la distribution $P(k)$ est indépendante du nombre d'itérations t.

Le modèle de Barabábasi-Albert indique que la *croissance* et l'*attachement préférentiel* jouent un rôle important dans le développement des graphes sans échelle. Dans [12], ils montrent que si l'un des deux mécanismes est omis du modèle, alors les propriétés sans échelle ne sont plus vérifiées. Dans un premier exemple, l'attachement préférentiel est enlevé du modèle, un nouveau sommet est connecté avec une égale probabilité à n'importe quel sommet u de la structure existante (on a $\Pi(u) = const$). Ils prouvent qu'un tel modèle mène à une distribution exponentielle. Dans le second exemple, le graphe considéré a un nombre de sommets n fixé. À chaque étape, un sommet est sélectionné aléatoirement et se connecte aux autres sommets en respectant l'attachement préférentiel. Dans un tel modèle, même si la loi en puissance est observée au début du processus, $P(k)$ n'est pas stationnaire : comme n est constant et que le nombre d'arêtes croît avec le temps, il vient un moment (quand $t \simeq n^2$, où le graphe atteint le statut de graphe complet, dans lequel tous les sommets sont reliés à tous les autres, (Déf. 11)). Cette expérience indique

que l'*évolution* et l'*attachement préférentiel* sont tous deux nécessaires au développement d'une distribution des degrés suivant une loi en puissance stationnaire comme on peut l'observer sur la figure 1.16.

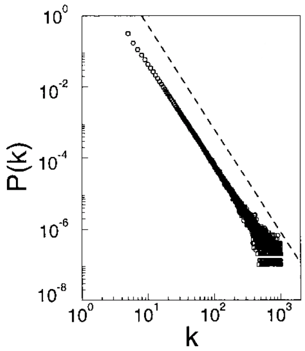


FIG. 1.16: Distribution des degrés du modèle Barabási-Albert à $t = 150000$ ($\circ$) et $t = 200000$ ($\square$), avec $m = n_0 = 5$. © A.-L. Barabási [12]

1.4.2.4 Modèle étendu de Barábasi-Albert

Dans des travaux récents, Albert et Barabási ont proposé un modèle étendu du modèle originel [4] : un processus de redirection des arêtes est intégré. Le modèle devient alors le suivant : partant d'un graphe à n_0 sommets, à chaque étape, on procède comme suit :

- Avec une probabilité p, on ajoute m nouvelles arêtes depuis un sommet u choisi aléatoirement uniformément. Les sommets destination sont sélectionnés avec une probabilité $\Pi(u)$ (voir formule (1.3)).
- Avec une probabilité q, on redirige m arêtes : on sélectionne un sommet u et une arête e issue de ce sommet aléatoirement uniformément, on supprime e et on la remplace par une arête e' reliant le sommet u et un sommet v selon la loi de distribution $\Pi(v)$.
- Avec une probabilité $1 - p - q$, on ajoute un nouveau sommet avec m arêtes vers un sommet u choisi avec la probabilité $\Pi(u)$.

Ce nouveau modèle permet plus de liberté quant au graphe obtenu. On note que pour les valeurs $p = q = 0$, le modèle correspond au modèle originel restrictif de Barabási-Albert.

Si l'on cherche à simuler le graphe de collaboration des acteurs d'Hollywood par exemple, on fixera $q = 0$ car une fois que deux acteurs ont joué dans un même film, le lien est établit, et il n'y a pas de redirection. Dans le graphe du Web en revanche, un lien peut être supprimé puis remplacé par un lien vers une autre page. Dans ce cas, on fixera q non nul.

1.4.2.5 Adaptation du modèle de Barábasi-Albert à la génération d'arbres aléatoires sans échelle

L'algorithme de Barábasi-Albert peut aisément être adapté à la génération d'un arbre aléatoire sans échelle [95, 96]. Le principe de construction de l'arbre est le suivant : l'algorithme commence avec un simple sommet racine v_1. À chaque étape, on rajoute un sommet v et un arc (u, v) d'un sommet u de l'arbre existant vers le sommet v. Le sommet u est choisi avec la probabilité $\Pi(u)$ de la formule (1.3).

La figure 1.17 montre un exemple de création d'un arbre en suivant le modèle de Barábasi-Albert

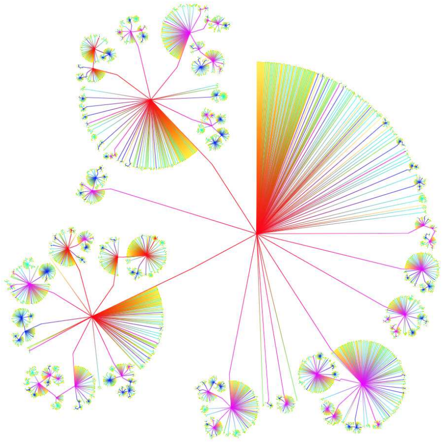


Fig. 1.17: Arbre aléatoire généré selon le modèle de Barábasi-Albert.

1.4.2.6 Autres modèles : les modèles évolutifs mixtes

On trouve dans la littérature une catégorie de modèles étendus de Barabási-Albert que l'on appelle ici les *modèles évolutifs mixtes*. Le principe est de combiner différentes stratégies pour le choix des arêtes : l'attachement préférentiel d'une part et le choix aléatoire uniforme d'autre part. Lors de l'ajout d'un nouveau sommet u, on choisit la stratégie de sélection du sommet v de graphe existant auquel sera rattaché le sommet nouvellement créé. Avec une probabilité p, le sommet v sera choisi aléatoirement uniformément parmi les sommets existants, et on choisira avec une probabilité $p - 1$ l'attachement préférentiel.

Móri a proposé une généralisation de la loi de probabilité de sélection du sommet pour la construction des arêtes (Form. (1.3)) intégrant ainsi une sélection par attachement préférentiel combinée avec une sélection uniforme [95, 96]. Le processus de construction est très proche de l'algorithme adapté de Barábasi-Albert pour la génération d'arbres aléatoires sans échelle mentionné dans la section précédente.

Le modèle de génération de Móri débute à l'itération $t = 2$ avec deux sommets v_1 et v_2, et un arc (v_1, v_2) entre ces deux sommets. A chaque étape, un nouveau sommet v est ajouté, avec un arc (u, v) reliant un sommet u de l'arbre existant et le nouveau sommet v. Le sommet u est choisi avec une probabilité

$$\Pi(u) = \frac{d^+(u) + \beta}{2(t-1) + t\beta} \tag{1.4}$$

Ici, $\beta > -1$ est un paramètre fixé. Ce paramètre permet de faire varier la distribution des stratégies d'attachement (préférentiel ou aléatoire). On rappelle que $d^+(u)$ correspond au degré sortant du sommet u (Déf. 14).

Comme nous travaillons avec un arbre connexe, on a, pour tous les sommets de l'arbre excepté la racine, la relation suivante : $d^+(u) = d(u) - 1$. Si l'on pose $p = \frac{1}{2+\beta}$, la formule (1.4) peut être ré-écrite ainsi :

$$\Pi(u) = \frac{p.d(u) + (1-p)}{t - 2p} \tag{1.5}$$

On note que pour la racine v_1, la formule est $\Pi(v_1) = (d(v_1) + 1)/(t - 2p)$. La formule (1.5) traduit le fait que pour chaque nouveau sommet, on choisit un sommet parent v dans l'arbre aléatoirement uniformément avec une probabilité p et sinon (probabilité $1 - p$) on utilisera l'attachement préférentiel.

Dans la même catégorie, on peut aussi citer le modèle de Cooper et Frieze [34] dans lequel est également ajoutée la possibilité de créer des arêtes supplémentaires entre les sommets existants.

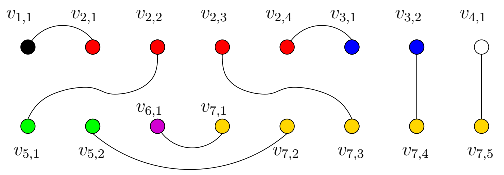
1.4.2.7 Autres modèles : les graphes aléatoires purs

Un second type de modèle de génération aléatoire de graphes sans échelle concerne les *graphes aléatoires purs*. Les graphes aléatoires sont dans ce cas générés à partir d'une distribution des degrés fixée *a priori* [2, 3, 94]. Pour générer une configuration aléatoire avec n sommets et une séquence de distribution des degrés fixée, le principe est le suivant :

- Construire un ensemble L contenant $d(v)$ copies distinctes de chaque sommet v.
- Établir un appariement aléatoire sur les éléments de L.

On peut permettre ou non l'appariement d'un sommet avec une de ses copies. On décide ainsi de la présence ou non de boucles dans le graphe résultat. De même, on peut contraindre le graphe à ne pas autoriser les multi-arêtes (on interdit la présence de deux paires $\{u_1, v_1\}$ et $\{u_2, v_2\}$, avec u_1, u_2 (resp. v_1, v_2) deux copies distinctes du sommet u (resp. v)).

On note que, dans cette approche, on ne cherche pas à reproduire les mécanismes qui sont à l'origine de la distribution selon une loi en puissance comme le font les modèles évolutifs, le but étant de produire une structure qui respecte une distribution des degrés fixée *a priori*.



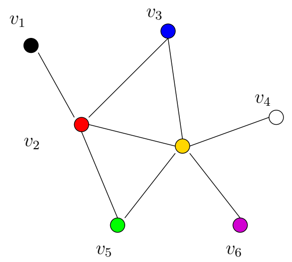


FIG. 1.18: Construction d'un graphe selon le modèle aléatoire pur. (a) Appariements des éléments de l'ensemble L, et (b) le graphe correspondant.

La figure 1.18 montre un exemple d'une telle construction pour un graphe à 11 sommets dont la séquence de distribution des degrés est $D = \{1, 4, 2, 1, 2, 1, 5\}$. On note d_i la i^{eme} valeur de la séquence D. On construit $L = \{v_{i,j}\}$ tel que $i \in \{1, \ldots, n\}, j \in \{1, \ldots, d_i\}$ et un appariement aléatoire sur les éléments de L (Fig. 1.18.a) tel que ni les boucles, ni les multi-arêtes ne sont autorisées. Le résultat obtenu est donné en figure 1.18.b.

1.4.2.8 Autres modèles : le modèle de Newman

À l'image des modèles aléatoires purs, Newman a proposé de généraliser le modèle aléatoire de Erdös et Rényi pour les graphes orientés et non-orientés de distribution des degrés arbitraire (non-Poisson) [100]. On dispose ici, comme pour le modèle précédent, d'une distribution des degrés fixée *a priori* $D = \{d_i\}_{i \in [1,n]}$. Pour générer le graphe G correspondant à la distribution D, Newman propose la méthode suivante :

- on place les n sommets tels que, pour chaque sommet v_i, on construit d_i brins (Déf. 8) (Fig. 1.19.a)
- on relie ensuite les brins aléatoirement uniformément de manière à constituer des arêtes (Fig. 1.19.b)

Lorsque tous les brins ont été connectés, le graphe résultant est l'un des éléments de l'ensemble des graphes dont la distribution des degrés correspond à la séquence D. On note que pour un sommet v_i, il existe $d_i!$ permutations possibles des brins issus de v_i. Ainsi, on a $\Pi_i(i\ d_i!)$ possibiltés de générer chaque graphe de l'ensemble. Ce facteur est constant à condition que la séquence D soit fixée. Ainsi, Newman justifie la restriction de son modèle à la considération d'une séquence de degrés fixe parce qu'elle assure une génération aléatoire uniforme des graphes de l'ensemble attendu, ce qui n'est pas le cas avec une loi de distribution.

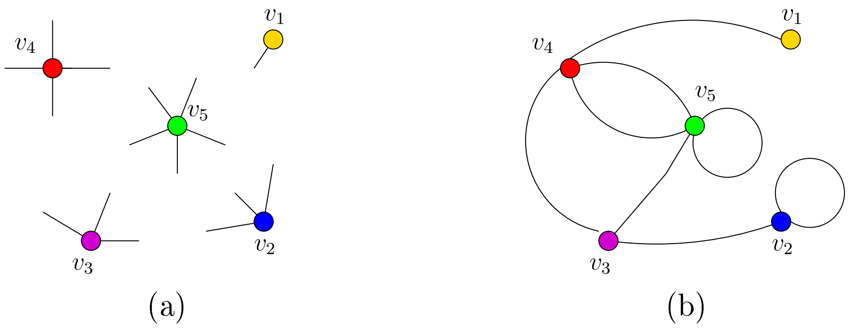


Fig. 1.19: Construction d'un graphe selon le modèle Newman. (a) Phase d'initialisation : on fait émerger d_i brins de chaque sommet v_i, et (b) le graphe correspondant à la connection des brins.

La figure 1.19 montre un exemple de génération aléatoire d'un graphe de distribution $D = \{1, 3, 3, 4, 5\}$: la phase d'initialisation (Fig. 1.19.a) qui associe d_i brins à chaque sommet v_i et le graphe généré aléatoirement par connection des brins (Fig. 1.19.b).

L'inconvénient majeur de ces deux derniers modèles est qu'ils nécessitent de fixer et le nombre de sommets que contiendra le graphe, et la distribution des degrés.

Nous avons présenté dans cette section les principaux modèles de génération aléatoire de graphes sans échelle existants. Dans le cadre de notre étude des systèmes de fichiers utilisateurs, nous verrons dans le chapitre 3, que les observations sur les données réelles ont permis d'extraire un ensemble de propriétés structurelles spécifiques au domaine d'application (outre la propriété sans échelle). Ces propriétés empiriques ne sont pas toutes reproductibles si l'on considère un modèle simple de génération aléatoire sans-échelle comme ceux présentés dans la présente section. Nous avons donc proposé un modèle plus complet qui tend à respecter l'ensemble des caractéristiques observées sur les arborescences. Le détail du modèle est présenté dans le chapitre 3.

1.5 Indexation multimédia

L'interrogation des bases de données multimédia par mots-clé est souvent considérée comme peu efficace du fait qu'elle est coûteuse en moyens humains (le plus souvent, les mots clés ou *tags* sont affectés manuellement) et que les index correspondants sont limités et subjectifs. En effet, l'interprétation de ce que l'on voit est difficile à caractériser, et d'autant plus difficile à apprendre à une machine. Ainsi, durant les dernières décennies, des tentatives ambitieuses ont été proposées pour que les ordinateurs apprennent à interpréter, indexer et annoter les documents multimédia représentant un large éventail de concepts.

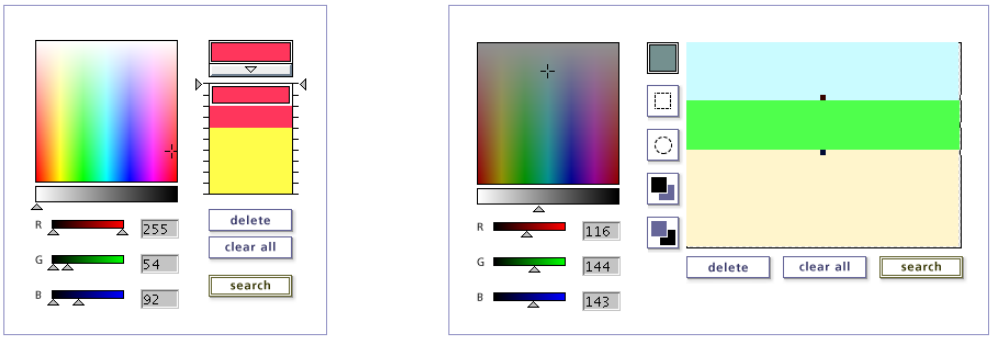
La recherche d'images basée sur le contenu (Content-based image retrieval (*CBIR*)) est l'une des techniques qui aide à l'organisation des documents multimédia digitaux par l'analyse de leur contenu visuel. Bien que la difficulté d'exprimer un contenu sémantique de haut niveau par des descripteurs visuels de bas niveau soit un problème majeur des approches basées sur le contenu, de nombreux systèmes ont prouvé leur efficacité [38, 129].

Dans cette section, nous proposons une revue des travaux et applications développées dans le contexte de l'indexation et la recherche de documents multimédia basées sur le contenu.

1.5.1 Généralités

Les interfaces de recherche dans les collections d'images basées sur une indexation par le contenu concernent essentiellement des applications professionnelles.

Les peintures du musée de l'Hermitage à Saint-Pétersbourg [70], sont indexées grâce au système QBIC [54]. Les requêtes peuvent être formulées grâce aux caractéristiques couleur des tableaux (figure 1.20.a) ou au travers d'une interface permettant de réaliser un croquis de l'œuvre recherchée (figure 1.20.b).



(a) (b)

Fig. 1.20: Interface pour la formulation de requêtes dans QBIC. (a) Spécification de la requête à l'aide de l'histogramme couleur et (b) spécification de la requête à l'aide d'un croquis couleur.

Le système de recherche SMURF [133] utilise une technique de reconnaissance de formes pour la recherche dans une base de données de hiéroglyphes (Fig. 1.21).

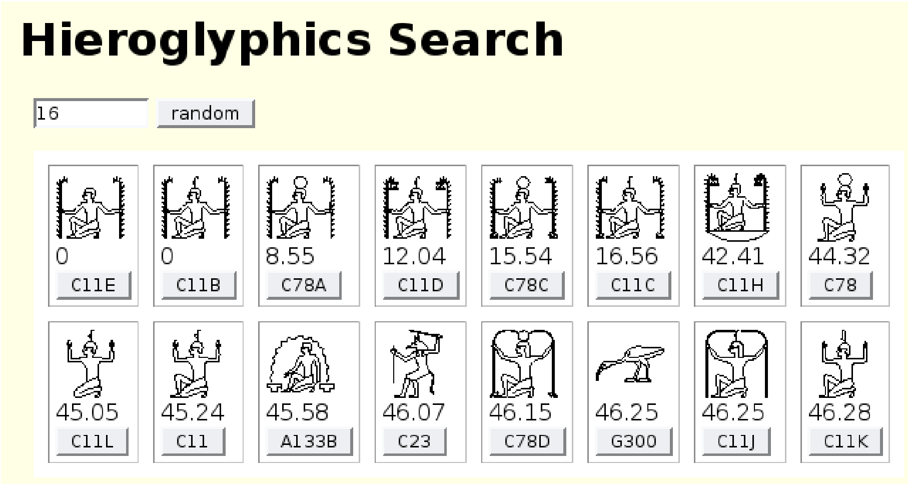


Fig. 1.21: Interface utilisateur du système de recherche de hiéroglyphes SMURF. La première image de la série, en haut à gauche, représente l'image requête.

De nombreux travaux concernent l'indexation et l'interrogation de collections d'images médicales. La recherche de caractéristiques discriminantes adaptées à la nature particulière des clichés médicaux est plus particulièrement traitée dans ce contexte (segmentation et indexation de régions d'intérêt [115, 142], détection et indexation de points d'intérêt [92]).

Dans le contexte de la protection de la propriété industrielle, des outils basés sur l'indexation par le contenu ont été proposés. Dans [67], les auteurs proposent d'indexer les schémas techniques des brevets à l'aide du graphe composé des lignes du schéma. L'indexation et la recherche de logos et de marques de fabrique dans des images est abordée dans [141].

La plupart des outils efficaces dédiés à la recherche dans les collections d'images concerne essentiellement des domaines industriels et des problèmes très contraints. Dans le contexte plus général de la recherche d'images naturelles ou de photographies personnelles, la contribution des techniques CBIR est très limitée. Cela est essentiellement dû au fossé sémantique qui empêche d'exprimer des requêtes de haut niveau sémantique à partir des caractéristiques visuelles de bas niveau des images.

Les propositions récentes concernant l'organisation et l'exploration des collections de photographies personnelles se basent essentiellement sur les annotations automatiques des collections d'images. On peut citer par exemple FlickR [55], qui permet de mettre en ligne des photographies personnelles et de permettre à un groupe d'utilisateurs d'annoter certaines zones de l'image et d'y associer des mots-clé prédéfinis (tags). La recherche d'une image se fait par mots-clé. Des systèmes similaires existent pour le partage de documents vidéo comme Google Video, Youtube et Dailymotion (voir figure 1.22).

Des techniques de classification plus sophistiquées sont utilisées pour produire des informations de plus haut niveau sémantique à partir du contenu des images.

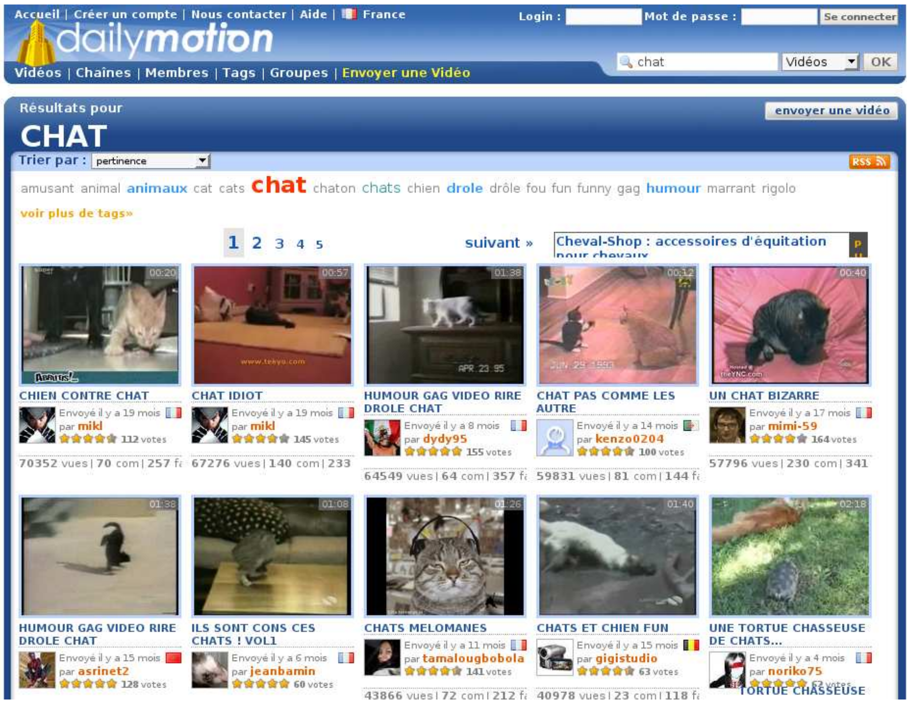


FIG. 1.22: Interface utilisateur du site Dailymotion. Résultat de la requête par le mot-clé *chat*.

La détection de visages basée sur les machines à vecteur de support (SVM) est l'exemple le plus représentatif de cette approche [103, 25]. Des systèmes de classification permettent de distinguer l'orientation des images (portrait/paysage) [128], l'environnement du contenu de l'image [85] (intérieur/extérieur), ou la nature du contenu [21] (naturel/artificiel).

Dans [93], un ensemble de catégories sémantiques est déterminé à partir d'un ensemble de descripteurs de bas niveau. Cet ensemble comporte 40 caractéristiques issues de l'analyse de l'image (nombre de régions après segmentation, fréquences spatiales, présence de lignes droites, *etc.*) et permet de définir 20 catégories sémantiques (feuillage, foules, paysages avec présence d'eau, paysages urbains, *etc.*).

1.5.2 Documents audiovisuels

Un flux vidéo est constitué d'une succession d'images, échantillonnées toute les 1/25 ou 1/30 de seconde. L'image constitue l'unité d'information vidéo. Selon les règles de production audiovisuelles [74], un document vidéo peut être décomposé en une hiérarchie de scènes et de plans. Un plan correspond à l'ensemble d'images situées entre le démarrage et la coupure de la caméra, ou entre des effets de transition. Les plans sont les composants élémentaires de la vidéo. Des plans consécutifs et dont le contenu partage les mêmes caractéristiques de lieu, de temps ou d'action peuvent être groupés en scènes [74].

La figure 1.23, illustre la décomposition d'un document vidéo en plans et scènes.

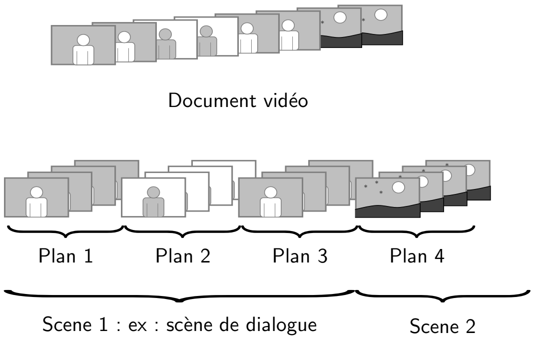


Fig. 1.23: Modèle de document vidéo.

Les contenus audiovisuels se distinguent notamment des collections d'images par l'ajout de la dimension temporelle et d'un flux audio. La recherche dans une base de données vidéo peut être considérée comme une extension au problème de CBIR puisque les vidéos sont une succession d'images fixes. Ainsi, les techniques dédiée à la recherche d'images par le contenu restent valables dans le domaine de la vidéo.

L'accès à un document vidéo repose essentiellement sur la structuration automatique du document vidéo. Les techniques utilisées consistent à détecter les frontières des plans vidéo et les frontières des scènes. Des informations supplémentaires peuvent alors être associées à ces segments (descripteurs de couleur, de mouvement, mots-clé). Un degré de raffinement supplémentaire consiste à analyser la présence d'objets dans les plans vidéo, par exemple un personnage, un véhicule, ou plus généralement une zone de l'image dont le mouvement diffère du mouvement global du plan vidéo [87]. Les objets peuvent alors être caractérisés par des descripteurs de trajectoire ou de couleur et être indexés indépendamment.

1.5.2.1 Indexation vidéo automatique

L'indexation vidéo automatique est un domaine de recherche très actif. Un projet précurseur dans ce domaine est le projet Informedia [60] qui propose la structuration de la vidéo à trois niveaux différents. D'abord, le niveau des paragraphes vidéo (équivalent des plans) est déterminé par l'analyse conjointe de la vidéo, des silences dans le flux audio et de la transcription automatique de la parole. Ensuite, les frontières de plans sont déterminées par l'analyse de l'histogramme couleur et du flot optique. Enfin, des images-clé sont choisies parmi les images statiques au sens de l'analyse du flot optique.

VideoQ [27] est un outil de recherche dans une collection de plans vidéo. Dans chaque image, un ensemble de régions est défini comme un ensemble de pixels homogènes en termes de couleur, de texture ou de forme. Un objet vidéo est défini par un ensemble de régions groupées selon certains critères au cours de plusieurs images. VideoQ caractérise les objets par leurs couleurs, forme, texture et le mouvement. Le mouvement est central dans ce système qui propose une interface utilisateur permettant de définir une requête à l'aide d'un croquis des objets (forme, couleur et texture) intégrant leur trajectoire.

Cette dernière approche constitue l'une des problématiques pour lesquelles nous avons apporté notre contribution. Deux méthodes pour la reconnaissance d'objets dans la vidéo à basse résolution sont ainsi présentées dans le chapitre 4 de ce manuscrit.

En plus de la segmentation automatique d'un flux vidéo en plans et objets, l'extraction automatique de texte peut être utilisée pour compléter l'analyse du flux vidéo. Par exemple, dans le projet Informedia [60], un système de transcription automatique de la parole (TTS) est utilisé pour produire des mots-clé associés aux paragraphes. Les mots-clé peuvent être utilisés pour accéder aux plans vidéo associés.

1.5.3 Formulation des requêtes

La formulation des requêtes dans des bases de données multimédia requiert des interfaces spécifiques permettant d'exprimer les caractéristiques des images ou des segments vidéo recherchés.

Des interfaces textuelles peuvent être proposées quand une indexation par mots-clé est disponible comme c'est le cas dans les systèmes Informedia [60] ou Dailymotion [37].

Dans WebSEEk [121], les images et les documents vidéo présents sur le Web sont organisés de manière automatique dans une hiérarchie de catégories. Cette hiérarchie fournit un point d'entrée dans la collection qui permet ensuite d'étendre la recherche au moyen d'interfaces de recherche basées sur le contenu. Il est alors possible d'effectuer une recherche en utilisant les caractéristiques couleur des images en modifiant l'histogramme des couleurs d'une image de référence.

L'interface utilisateur de VideoQ [27] permet de spécifier les caractéristiques de couleur, de forme, de texture et de trajectoire d'un objet recherché dans une séquence vidéo.

L'utilisation des propriétés globales sur les images est une approche classique dans la plupart des systèmes de CBIR. Dans un document vidéo en particulier, les objets de premier plan peuvent apparaître dans plusieurs scènes de contextes différents. La requête basée sur le contenu peut alors être formulée en termes de présence ou non d'un objet d'intérêt dans l'image ou le plan vidéo. La recherche en CBIR tend à se tourner vers des techniques orientées objet [48, 47, 71, 77, 86, 119]. Dans ce cas, la recherche par le contenu considère uniquement une région d'intérêt de l'image, appelée *objet d'intérêt*, qui porte la plus grosse partie de l'information de l'image. Deux images qui contiennent le même objet d'intérêt dans différents contextes diffèrent uniquement par leur fond : la sémantique principale des deux images est la présence de l'objet lui-même. Dans ce contexte, l'extraction manuelle ou automatique des objets de premier plan est nécessaire.

Dans le cas de l'extraction manuelle des objets d'intérêt, l'utilisateur doit sélectionner les parties de l'image qui contiennent ce qui l'intéresse. Dans [71], un processus de segmentation est effectué dans une fenêtre de sélection et l'utilisateur doit cliquer sur un ensemble de régions pour déterminer son objet d'intérêt. Dans [119] des régions appelées régions covariantes sont détectées automatiquement dans une fenêtre de sélection (une région détectée est représentée par une ellipse). Une représentation basée objet similaire (utilisée dans [77]) est appelée *blobworld* [26]. Dans cette approche, chaque objet peut être représenté par un ensemble d'ellipses ou *blobs* possédant chacune un nombre d'attributs propres.

Un autre type d'extraction concerne l'extraction automatique des objets d'intérêt [48, 86]. Le standard vidéo ISO/MPEG-4 supporte la représentation d'objets vidéo de forme arbitraire (Video Objects (VOs)). Dans ce cas, les objets sont directement disponibles parce qu'ils constituent une composante du flux vidéo. Il existe également des techniques automatiques ou semi-automatiques de détection et suivi d'objets dans la vidéo. Dans [57], une première occurrence de l'objet à suivre est segmentée de manière semi-automatique. L'utilisateur intervient en rectifiant la décomposition de l'objet en régions grâce à une hiérarchie de partitions. Par la suite, une segmentation multi-échelle de chaque image vidéo est effectuée afin d'effectuer un suivi des régions qui composent l'objet. Dans [87], une technique basée sur l'analyse du mouvement des pixels dans le flux vidéo permet d'extraire les portions de l'image dont le mouvement diffère de celui de la caméra.

Les techniques de mise en correspondance ont alors pour rôle de comparer des images dans leur intégralité ou bien des zones d'intérêt sélectionnées dans les images. Les méthodes de CBIR ne sont généralement pas restreintes à l'une ou l'autre forme de requêtes et sont donc applicables dans les deux cas, sans modifications. D'autre part, nous ne faisons pas dans ce cas la distinction entre les images issues de la vidéo et les images fixes d'une base de données quelconque. Ainsi, l'ensemble des méthodes présentées par la suite sont valables pour la recherche d'images par le contenu, quelle que soit leur origine.

1.5.4 Techniques de mise en correspondance du contenu

Deux approches sont utilisées par les systèmes CBIR pour la mise en correspondance du contenu des images. L'analyse peut être effectuée en utilisant des propriétés globales de l'image (QBIC [54], Photobook [105], VisualSeek [120]). La question que l'on peut soulever dans ce cas est à quel point les descripteurs globaux de l'image sont corrélés à la sémantique du contenu.

Une autre technique vise à décomposer les images en zones homogènes, appelées régions, par un processus de segmentation. Le contenu des images peut alors être analysé en terme d'ensemble de régions ayant chacune des propriétés locales propres. Dans cette deuxième approche, la recherche basée sur le contenu consiste à mettre en correspondance les régions des images de la base de données avec les régions de l'image ou du croquis requête. On appelle ces techniques les techniques de mise en correspondance basée région.

Dans les deux cas, les systèmes utilisent un ensemble de caractéristiques bas niveau extraites des images pour en décrire le contenu.

1.5.4.1 Description du contenu : les caractéristiques de l'image

Les systèmes de recherche d'images basée sur le contenu s'appuient sur une analyse de descripteurs bas niveau extraits de l'image : la plupart du temps, il s'agit d'extraire les propriétés de couleur, de texture et de forme pour caractériser le contenu d'une image et de comparer ensuite ces descripteurs pour évaluer la similarité qu'il existe entre les images. Un descripteur est défini pour capturer certaines propriétés visuelles d'une image, soit globalement pour décrire l'image entière, soit localement pour décrire un petit groupe de pixels. L'avantage des descripteurs globaux réside dans la rapidité de calcul et du descripteur, et de la mesure de similarité entre deux descripteurs. Cependant, les descripteurs globaux sont souvent insuffisants pour représenter une image. En particulier, ils ne prennent pas en considération la composition spatiale des images et par conséquent, sont incapables d'identifier certains éléments visuels importants. Pour améliorer la robustesse aux transformations spatiales, la seconde approche consiste à considérer les descripteurs locaux.

Nous présentons dans les paragraphes qui suivent une revue des descripteurs classiques utilisés dans le domaine du CBIR.

Les descripteurs de couleur

La très grande majorité des systèmes de CBIR utilisent la couleur comme descripteur principal du contenu. Pour cela, il est fondamental de préciser l'espace de couleurs considéré dans l'application. Il existe de nombreux espaces de couleurs (*e.g.* RVB, HSV, CIE, ...) et la perception des couleurs est très subjective. La littérature dense à ce sujet montre qu'il n'existe pas d'espace de couleur idéal pour la recherche d'image basée sur le contenu.

Le descripteur de couleur le plus simple consiste à calculer la couleur moyenne (de l'image ou de la région considérée) dans l'espace de couleurs sélectionné. La distance entre les descripteurs correspond à une distance entre les vecteurs de couleurs obtenus.

On peut citer d'autre part l'histogramme couleur [47, 54, 114, 120] qui offre une description globale de la répartition des couleurs dans l'image. Comme l'histogramme est un descripteur global, les approches basées sur l'histogramme couleur sont invariantes aux transformations usuelles telles que la rotation, la translation et le changement d'échelle. Cependant, une image dont l'histogramme couleur est proche de l'histogramme d'une autre image peuvent être très différente du point de vue du contenu. En effet, les relations spatiales entre les régions qui composent les images ne sont pas prises en compte. De ce fait, l'utilisation de l'histogramme couleur est souvent trop imprécise pour la recherche d'images par le contenu. Dans le système QBIC [54], l'espace de couleurs est quantifié en 64 couleurs afin de simplifier la représentation sous forme d'histogramme.

Il existe aussi des techniques de résumé des couleurs contenues dans une image par la construction d'une signature des couleurs [112]. Le principe est de découper l'image en blocs de pixels qui sont progressivement groupés en clusters selon une construction itérative par fusion de blocs adjacents. Chaque cluster est caractérisé par sa couleur moyenne et le nombre de pixels contenus dans le bloc. La signature de l'image correspond à la liste des clusters.

Une étude comparative des différents descripteurs colorimétrique est proposée dans [110].

Les descripteurs de texture

Les descripteurs de texture sont utilisés pour capturer la granularité et les motifs répétitifs de surfaces dans une image. Par exemple, l'herbe, un mur de briques, des nuages ou la fourrure d'un ours diffèrent du point de vue de la texture, autant au niveau de la granularité que des motifs. Leur rôle dans la recherche d'images dans un domaine spécifique, tel que l'imagerie aérienne ou l'imagerie médicale, est capital en raison de leur relation étroite avec la sémantique dans ces cas.

La technique de représentation de texture la plus communément utilisée est basée sur le calcul des moments de second degré des groupes de pixels considérés. Les premiers descripteurs de textures semblent être proposés par Tamura *et al.* [124] pour décrire six concepts liés à la perception : granularité, contraste, direction prédominante, linéarité, régularité et la rugosité (les trois premiers descripteurs sont utilisés dans le système QBIC [54]).

Les techniques basées sur la décomposition fréquentielle des images sont aussi des approches classiques. L'utilisation de filtres dans le domaine spatial (tels que les filtres de Gabor [72]) d'orientation et de taille différentes permettent d'extraire un ensemble de caractéristiques fréquentielles et directionnelles dans l'image pour définir la texture.

Il existe de nombreuses autres approches pour décrire la texture. Une comparaison des différents descripteurs est proposée dans [64].

Les descripteurs de forme

Le descripteur de forme est un attribut clé des images segmentées en régions, et sa robustesse joue un rôle important dans la reconnaissance. Nous distinguons deux catégories de descripteurs de forme : les descripteurs basés régions et les descripteurs basés frontières.

Les descripteurs basés régions font classiquement référence aux moments invariants [65] et sont utilisés pour caractériser l'intégralité de la forme d'une région. Ces attributs sont robustes aux transformations géométriques comme la translation, la rotation et le changement d'échelle.

La seconde approche, le plus souvent basée sur les descripteurs de Fourier [113], porte sur une caractérisation des contours de la forme étudiée.

Un fois les descripteurs calculés, le système de recherche d'images par le contenu se base sur ces derniers pour évaluer une mesure de similarité entre l'image requête et les images de la base de données.

1.5.4.2 La mesure de similarité entre les images

La caractérisation du contenu des images est établie par un ensemble de descripteurs visuels calculés sur celles-ci. L'utilisation d'un simple descripteur global est peu efficace, ainsi, il est fréquent de constater un grand nombre de données numériques pour caractériser une seule et même image. Le problème désormais, est d'exploiter ces données en vue de calculer une mesure de similarité entre l'image requête soumise au système et l'ensemble des images contenues dans la base de données.

La gestion des données par un système de gestion de bases de données (SGDB) est inefficace. En effet, les données se présentent sous forme de vecteurs numériques de grande dimension, et toutes les dimensions doivent être considérées simultanément, ce que les SGDB ont l'incapacité de réaliser. D'autre part, les requêtes sont telles que l'on cherche des images aux descripteurs voisins d'un descripteur requête, et non pas strictement égaux. Ce qui implique le plus souvent une exploration exhaustive des éléments de la base de données.

Il existe des techniques de filtrage pour confiner la recherche : on cherche à réduire le plus vite possible l'ensemble des données que l'on va nécessairement passer en revue. Ces techniques impliquent d'accepter que le résultat de la requête ne sera pas nécessairement optimal.

Une autre approche consiste à représenter les données sous forme de graphes d'adjacence des régions pour conserver l'information sur la composition des images. Comme nous l'avons mentionné dans la section 1.2, comparer des graphes est une opération compliquée, et l'on ne sait pas indexer dans ce cas : tous les éléments doivent être analysés. Il faut alors utiliser les spécificités des graphes afin d'accélérer la recherche (en écartant éventuellement des candidats).

Il existe de nombreuses approches pour évaluer la similarité entre les images basée sur la combinaison de descripteurs visuels. Devant l'abondance des méthodes, nous ne pouvons établir ici un état de l'art exhaustif. Toutefois, une revue détaillée des méthodes est proposée dans [38].

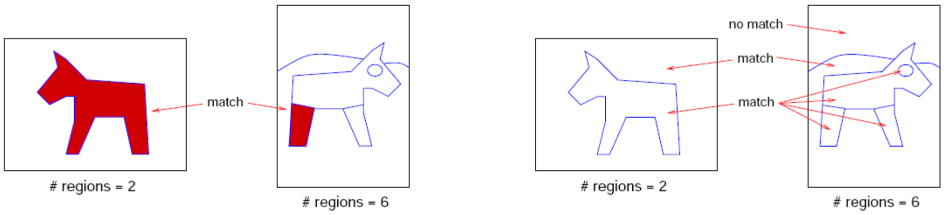
La restriction de l'espace de recherche

Les techniques visant à restreindre l'espace de recherche sont basées sur un préfiltrage utilisant le plus souvent des descripteurs globaux de l'image. L'ensemble des images à étudier peut être réduit par une étape de filtrage de la base de données : par exemple, si la couleur dominante de l'image requête est trop éloignée, ou si la distance entre les histogrammes couleurs est trop élevée, alors il est possible d'écarter certains éléments de la base de données, pour ne conserver que les candidats potentiels. Dans le système SIMPLICity [138], les auteurs utilisent une classification de la base de données d'images (intérieur/extérieur, texturée/non texturée, ville/campagne, etc.) pour accélérer la recherche.

Les approches basées sur la mise en correspondance de graphes

Les techniques basées sur la mise en correspondance des graphes d'adjacence des régions (RAG) ont fait l'objet de nombreuses recherches [66, 81, 57, 40]. L'ensemble de ces méthodes rejoint le problème de quasi-isomorphisme de graphes détaillé dans la section 1.2 de ce chapitre.

Une approche que nous n'avons toutefois pas encore abordé concerne la mise en correspondance des régions *many-to-many* [138, 75]. Dans ce contexte, la restriction de mise en correspondance des régions une à une est levée pour compenser les erreurs de segmentation et les déformations.



(a) (b)

Fig. 1.24: Exemple de mise en correspondance basée région : (a) la mise en correspondance un à un et et (b) la mise en correspondance *many-to-many*.

Dans le système SIMPLICity [138], la mise en correspondance des régions est effectuée itérativement, en appareillant à chaque nouvelle étape la paire de régions de plus forte similarité. Les approches classiques d'appariement des régions une à une écartent toute région déjà mise en correspondance pour la suite du processus, ce qui peut entraîner une sous détection dans le cas de sous-segmentation (Fig. 1.24.a). Dans les approches dites *many-to-many* une seule et même région peut être associée à plusieurs régions dans l'autre image (Fig. 1.24.b) permettant ainsi de corriger les problèmes liés à la segmentation.

En outre, pour éviter les erreurs de mise en correspondance, la technique IRM (Integrated Region Matching) utilisée dans SIMPLICity considère les relations d'adjacence des régions pour la mise en correspondance de plusieurs régions : ainsi, deux régions d'une image peuvent être associées à une seule et même région dans l'autre image uniquement si ces deux régions sont adjacentes.

L'abondance de descripteurs et l'abondance de méthodes pour exploiter ces derniers ont permis de proposer un grand nombre d'approches pour la mise en correspondance par le contenu. Les systèmes proposés utilisent diverses techniques : les descripteurs choisis sont parfois définis pour un domaine particulier (images médicales, graphiques, ...) et sont inefficaces pour la recherche d'images en général. Afin de d'évaluer les performances des techniques de recherche, deux mesures sont classiquement utilisées. Nous présentons ces mesures dans la section qui suit.

1.5.5 Évaluation des méthodes d'indexation

L'évaluation des performances des méthodes d'indexation est un point délicat. D'une part, l'accès à des collections d'images et de documents vidéo est limité par les droits intellectuels liés à ces œuvres. L'accès à des corpus de grande taille peut se faire via des sociétés commerciales telles que Corel. Dans le cadre de cette thèse, la plupart des documents vidéo utilisés sont issus du corpus du centre de ressources et d'information sur les multimédias pour l'enseignement supérieur (CERIMES, ex SFRS). Le fait que les corpus ne soient pas disponibles librement rend la comparaison des méthodes d'indexation plus difficile.

D'autre part, l'évaluation repose souvent sur une "vérité terrain", c'est à dire une annotation parfaite d'un ensemble de documents de référence. Ces méta-données sont souvent produites manuellement et servent de référence pour comparer les résultats d'indexations automatiques. Par exemple, les frontières de plans d'un flux vidéo et les effets de transition peuvent être détectés manuellement et utilisés pour évaluer les détecteurs automatiques. La qualité et la publicité des annotations de références sont également indispensables pour permettre à différents auteurs de comparer leurs contributions.

Dans le contexte de la recherche d'information l'efficacité d'une procédure de recherche d'information se mesure grâce au rappel et à la précision.

Définition 33 (Rappel et précision)
Soit Q une collection de documents, soit $P \subseteq Q$ l'ensemble de documents pertinents pour une requête donnée et soit $R \subseteq Q$ l'ensemble de documents retournés par une méthode de recherche d'information. Le rappel ("Recall") est défini par :

$$Recall = \frac{|P \cap R|}{P}$$

La précision ("Precision") est définie par :

$$Precision = \frac{|P \cap R|}{R}$$

Le rappel donne la proportion des éléments détectés qui sont effectivement pertinents. Il caractérise ainsi la capacité de détection. La précision indique le pourcentage d'éléments qui ont été correctement détectés.

Nous venons de présenter un ensemble de techniques pour la recherche de documents multimédia et les méthodes pour évaluer leurs performances. Nous avons proposé dans le cadre de cette thèse, deux méthodes pour la reconnaissance des objets extraits de la vidéo à basse résolution. La technique se base sur la comparaison de graphes construits à partir d'une zone de l'image représentant l'objet de premier plan. Dans notre contexte, les images sont fortement dégradées et les descripteurs précis sont peu appropriés. Les méthodes proposées sont des méthodes adaptées aux données de faible résolution. Le détail est donné dans le chapitre 4.

1.6 Aide à la maintenance de projets logiciels

Dans un projet de développement logiciel, il est montré que la maintenance du code existant occupe plus de 80% de l'activité totale liée au projet [107].

De nombreux outils destinés aux développeurs pour aider à la conception et au développement de gros projets ont été proposés. Nous présentons dans cette section un aperçu des outils et méthodes d'aide à la maintenance de projets logiciels, domaine dans lequel s'inscrivent certaines de nos contributions.

1.6.1 Les dépôts de versions

L'un des outils les plus importants pour une bonne gestion du développement de projets logiciels concerne les systèmes de dépôt de versions (software configuration management (SCM)) tels que CVS [58], Subversion [33], ClearCase [69] et Perforce [106].

Le rôle de ces systèmes est la gestion des multiples révisions d'un même ensemble de documents. Ils sont le plus communément utilisés dans l'ingénierie et le développement logiciel pour la gestion de l'avancement du développement des documents digitaux tels que le code source de l'application sur lesquels peut travailler toute une équipe de développeurs. Les changements des documents sont identifiés par un numéro de version et associés à l'utilisateur ayant effectué les modifications.

Le développement simultané d'un même logiciel par plusieurs programmeurs est classique dans l'industrie du logiciel. Cela implique la présence de multiples versions du même programme sur de nombreux postes de travail. Il est alors important de centraliser les travaux de toute une équipe dans une même base de dépôt commune qui continent l'intégralité des mises à jour effectuées par les différents développeurs.

Les bogues et autres problèmes logiciels sont souvent présents dans certaines versions (du fait de la résolution de certains problèmes et de l'introduction d'autres problèmes au fil du développement). Par conséquent, pour la localisation et la résolution de bogues, il est très important de pouvoir retrouver et exécuter différentes versions du programme pour déterminer dans quelle(s) version(s) le problème est apparu.

1.6.2 Interfaces de visualisation

Il est apparu récemment des outils de visualisation en extension aux gestionnaires de dépôt de versions [82, 136, 134, 135, 53]. Ces outils proposent à l'utilisateur une interface de visualisation permettant d'analyser les évènements importants qui ont eu lieu au cours du développement logiciel.

Le principe est de proposer à l'utilisateur (développeur) une représentation visuelle de l'évolution du code à différents niveaux de détails (ligne de code, fichier, projet). Cette représentation doit lui permettre d'identifier instantanément les évènements majeurs dans l'évolution du code.

Au niveau de détail du fichier, les questions auxquelles l'utilisateur cherche à répondre sont du type :

- quelles lignes de code ont été supprimées, ajoutées, modifiées et quand ?

- qui a effectué ces modifications ?
- quelles parties du code sont instables ?
- quelles sont les corrélations entre les changements ?
- comment sont réparties les tâches de développement ?

L'interface de visualisation ViewCVS [53] permet de visualiser les différences existant entre deux versions d'un même fichier (Fig. 1.25). Le principe est de montrer simultanément les deux codes source en colorant selon la légende dans l'une ou l'autre version, les lignes de code supprimées, ajoutées ou modifiées.

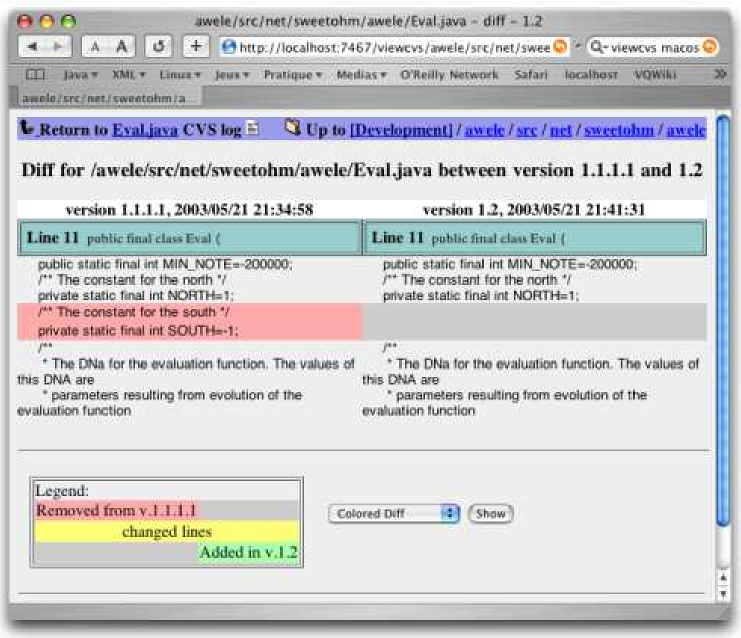


FIG. 1.25: Interface de visualisation de ViewCVS : différences entre deux versions d'un fichier.

En partant du principe que l'on ne souhaite pas nécessairement visualiser tout le texte du fichier, mais seulement le détail du texte de quelques lignes de code qui présentent un intérêt, l'outil CVSscan [136] se base sur une visualisation orientée pixel. L'ensemble des versions du fichiers sont représentées sur l'axe horizontal (de gauche à droite, des versions les plus anciennes vers les versions les plus récentes) et, une ligne de code dans le fichier source correspond à un pixel (sur l'axe vertical) dans la visualisation. La figure 1.26 montre un exemple de visualisation d'un fichier par l'outil CVSscan.

Dans la fenêtre de visualisation principale (1), chaque pixel correspond à une ligne de code du fichier à une version donnée. Les pixels gris représentent l'absence de ligne de code. Par cette visualisation, on détecte aisément les zones de code qui ont été ajoutées et supprimées (elle correspondent aux zones grises) et à quel moment. Dans cet exemple, les modifications d'une ligne de code sont représentées par une couleur correspondant à l'auteur de la modification (couleur différente du vert). Nous pouvons ainsi déterminer les changements majeurs dans le code et les auteurs des modifications. Les fenêtres (2) et (3) situées en bas de l'interface graphique sont

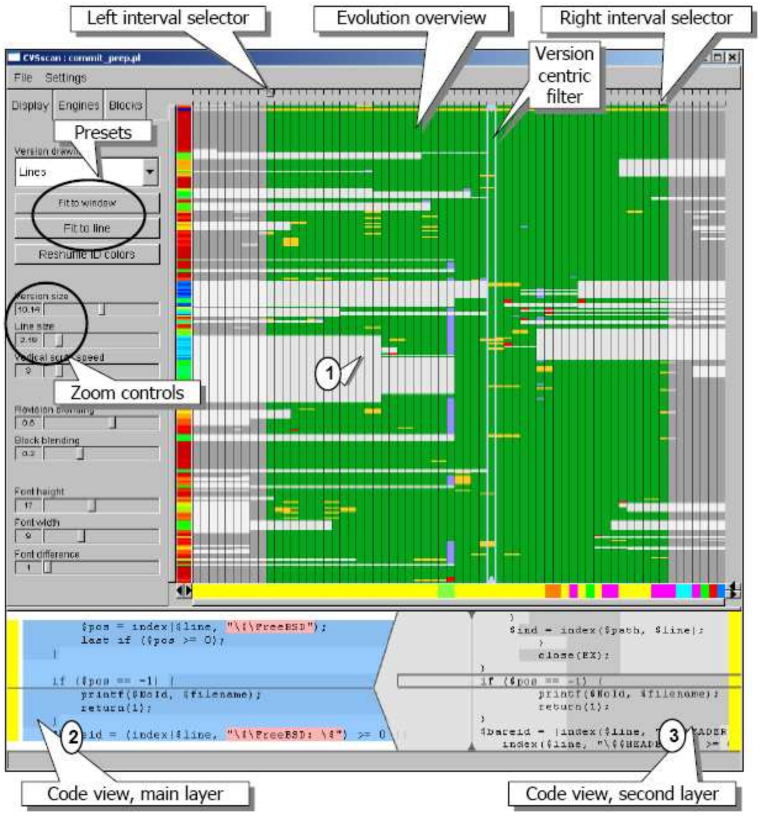


FIG. 1.26: Interface de visualisation de CVSscan.

réservées à la visualisation des lignes de code source. Dans la fenêtre de gauche (2), le code source de la version du fichier sélectionnée est représenté : c'est le code de référence. Nous pouvons visualiser, grâce à la fenêtre de droite (3), les changements par rapport à une autre version (pointée par la souris) de ce même fichier. Dans cet exemple, on peut voir que dans une version future, un bloc de code va être inséré dans le fichier à la position indiquée par la ligne horizontale grise dans la fenêtre (2).

À ce niveau de détail, les outils tels que ViewCVS [53] et CVSscan [136] sont destinés à aider les développeurs du projet. Pour la maintenance de projets entiers, il est intéressant de disposer également d'un type d'outils similaires pour la visualisation de l'évolution du projet à un niveau de détail plus élevé.

Au niveau de détail du projet, les questions posées sont du type :
- quelle sont les parties du projet les plus actives ?
- quels sont les fichiers ajoutés, supprimés, modifiés et quand ?
- qui a effectué les changements ?
- comment cette activité a-t'elle évolué au cours du projet ?
- quels sont les fichiers qui sont altérés simultanément ?

L'interface CVSgrab, proposée par Voinea et Telea [134, 135], permet de visualiser l'évolution d'un projet logiciel au niveau de détail du fichier. La figure 1.27 montre un exemple de visualisation du projet logiciel Mozilla. À la manière de

CVSscan, la visualisation est orientée pixel, avec les colonnes correspondant à une version du projet logiciel, et chaque ligne représente un fichier du projet.

Fig. 1.27: Interface de visualisation de CVSgrab.

Dans l'exemple de la figure 1.27, la fenêtre principale (D) montre les différentes versions de chacun des fichiers du projet. Les couleurs correspondent à la présence d'un bogue signalé dans le log du fichier au moment du dépôt. Les développeurs précisent lors de la signalisation d'un bogue, le niveau d'importance de ce dernier (amélioration, trivial, mineur, normal, majeur, critique et bloqueur). Les zones en rouge (selon la légende de l'attribut couleur affichée dans la fenêtre (B) de la visualisation), correspondent aux bogues bloqueurs. Nous pouvons ainsi identifier très rapidement les fichiers du projet qui posent problème, les bogues résolus...

Il existe de nombreux scénarios auxquels CVSgrab permet de répondre. L'outil intègre un système de tri des fichiers selon leur ressemblance à un fichier sélectionné du point de vue des modifications. Cela permet de visualiser les corrélations qu'il existe entre les différents fichiers. Le tri peut également considérer l'ancienneté des fichiers. Nous pouvons alors observer les phases du projet qui sont stables (pas de création de fichiers) ou au contraire, lorsque le projet est en phase de restructuration. À ces fonctionnalités est ajoutée la possibilité d'afficher un ensemble d'attributs aux fichiers (auteur du changement, présence d'un certain mot, type de fichier, ...). Il est également possible de grouper les fichiers en clusters par ressemblance selon un attribut particulier. En combinant les différentes techniques, on peut répondre rapidement à certaines questions auxquelles il est difficile de répondre sans connaissance *a priori* de l'historique du projet comme par exemple : quels ont été dans le temps les développeurs majeurs de ce projet ?

Sur l'exemple de la figure 1.28, on montre le résultat de l'affichage de deux projets différents, avec le tri des fichiers par ordre de création (en haut le fichier le plus ancien, en bas, le fichier le plus récent). Le code couleur représente les différents développeurs qui ont apporté les modifications aux fichier.

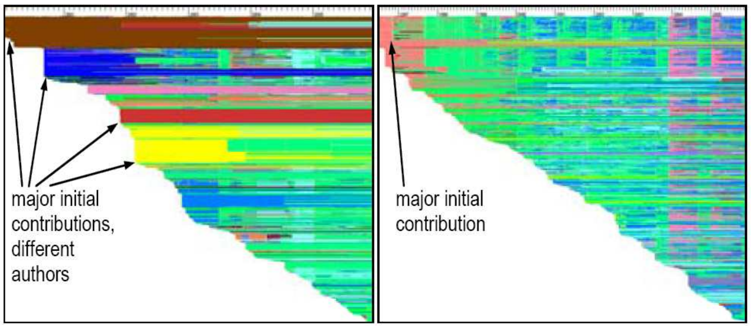


FIG. 1.28: Interface de visualisation de CVSgrab : auteurs des modifications.

On peut observer sur l'exemple de gauche, que le projet a subi d'importantes modifications (essentiellement des ajouts de fichiers) à l'arrivée de différents développeurs (bleu foncé, jaune, rouge). On peut en déduire l'ajout de nouveaux modules au projet, car d'une part, les développeurs arrivant créent beaucoup de fichiers d'un coup et d'autre part, les fichiers déjà existants continuent d'être maintenus par les développeurs initiaux (*e.g.* bande horizontale marron en haut de la visualisation) : on note dans ce cas des similarités des attributs dans le sens horizontal.

Dans l'exemple de gauche, un phénomène différent peut être observé : on remarque des similarités des attributs dans le sens vertical cette fois-ci. Ce phénomène traduit les transitions dans les équipes de développement (départ/arrivée de développeurs) car il n'y a pas (ou peu) de continuité horizontale.

L'ensemble de ces outils de visualisation ont pour objectif d'aider aux développeurs et aux chefs de projet d'identifier rapidement les phénomènes importants dans l'évolution du projet.

Les autres outils d'aide à la maintenance logicielle ont pour objectif de détecter les clones dans le code pour éviter les erreurs dues à la duplication de code engendrées par des modifications en cascade. Nous présentons dans la section qui suit un aperçu des méthodes pour la détection de duplication dans le code.

1.6.3 Détection de clones dans le code

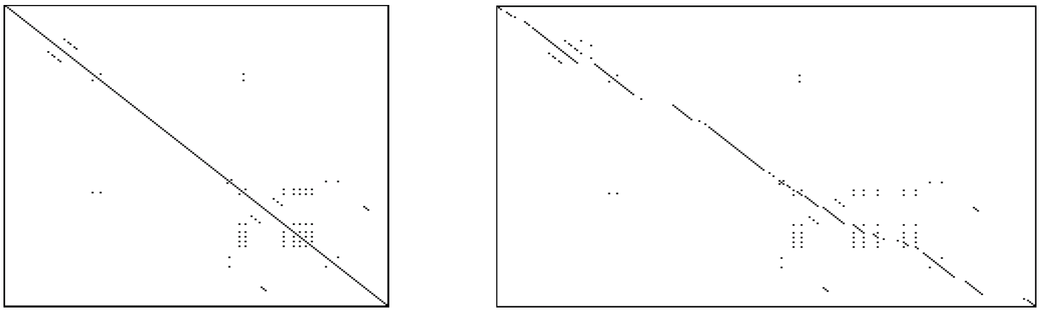
L'une des causes principales aux efforts de maintenance est la présence d'une fraction considérable de duplication de code (clones). L'existence de clones dans le code peut introduire de nombreuses instabilités dans une application logicielle. La complexité des tâches routinières de maintenance est accrue : un changement dans une méthode peut entraîner plusieurs changements dans de nombreuses autres méthodes.

Il existe plusieurs types de techniques de détection de clones dans le code. Ces techniques se distinguent par le type d'information qu'elles analysent : les méthodes basées sur le texte, et les méthodes basées sur l'arbre syntaxique.

1.6.3.1 Comparaison de texte

Parmi les méthodes basées sur la comparaison de texte, on peut citer l'approche de Ducasse *et al.* [45] qui compare toutes les lignes de code, par paires, de deux versions d'un même fichier (ou de deux fichiers différents). Pour améliorer les performances, les lignes de code sont partitionnées via une fonction de hachage pour les chaînes de caractères. Le principe est de regrouper dans un même ensemble les lignes de code proches du point de vue lexicographique. Seules les lignes de code appartenant à un même ensemble sont comparées.

Le résultat est présenté sous la forme d'une matrice de points (scatterplot) telle que les lignes du fichier i sont représentées par les lignes de la matrice, et les lignes du fichier j par les colonnes de la matrice (Fig. 1.29). Un point noir à la case (i, j) de la matrice correspond à une ligne clonée entre les deux fichiers i et j. Pour repérer les blocs de code dupliqués, on recherche les motifs apparents dans la matrice.



(a) (b)

Fig. 1.29: Comparaisons de fichiers : (a) comparaison d'une même version d'un fichier et (b) comparaison de deux versions différentes d'un même fichier.

1.6.3.2 Comparaison de métriques

Le principe des méthodes basées sur la comparaison de métriques est de rassembler différentes métriques pour des fragments de code (telles que le nombre d'affectations, le nombre de boucles `for`, le nombre de déclarations, ...) et de construire ainsi un vecteur de caractéristiques lié au bloc de code analysé. Les vecteurs sont ensuite comparés (par exemple, via une distance euclidienne) pour évaluer la similarité entre les blocs de code. Cette technique a d'abord été proposée par Merlo *et al.* [89] puis récemment réutilisée par Jiang *et al.* [73] dans le cadre de la comparaison d'arbres syntaxiques.

1.6.3.3 Comparaison des arbres syntaxiques (AST)

La représentation du code source sous forme d'arbre syntaxique est une approche classique à l'analyse du code [17, 137, 73, 50]. Le principe est de modéliser la structure hiérarchique du code définie par les éléments syntaxiques tels que les classes, les fonctions, les identifieurs, les boucles, *etc.* Il existe de nombreux avantages à une telle représentation par rapport à la version textuelle du code source. Tout d'abord, les noms donnés aux variables, fonctions ou classes constituent des informations non pertinentes à l'analyse de la structure. Ainsi, tout changement d'appellation n'aura pas d'incidence sur la représentation en AST du code. D'autre part, l'ordre des lignes de code n'est pas nécessairement déterminant. Dans la plupart des cas, on peut intervertir des lignes de code, voire des blocs de code entiers sans que cela ne modifie le résultat. Dans ce cas, il s'agit d'éléments hiérarchiques indépendants les uns des autres. Les techniques d'analyse basées sur la comparaison textuelle peuvent être perturbées par de tels changements. La représentation sous forme d'AST permet de s'affranchir de l'ordre dans lequel sont déclarés les blocs de code indépendants : la structure arborescente, si l'on ne considère pas d'ordre sur les sommets, reste stable à ces modifications.

La figure 1.30 montre un exemple de code C++ et l'arbre syntaxique annoté correspondant.

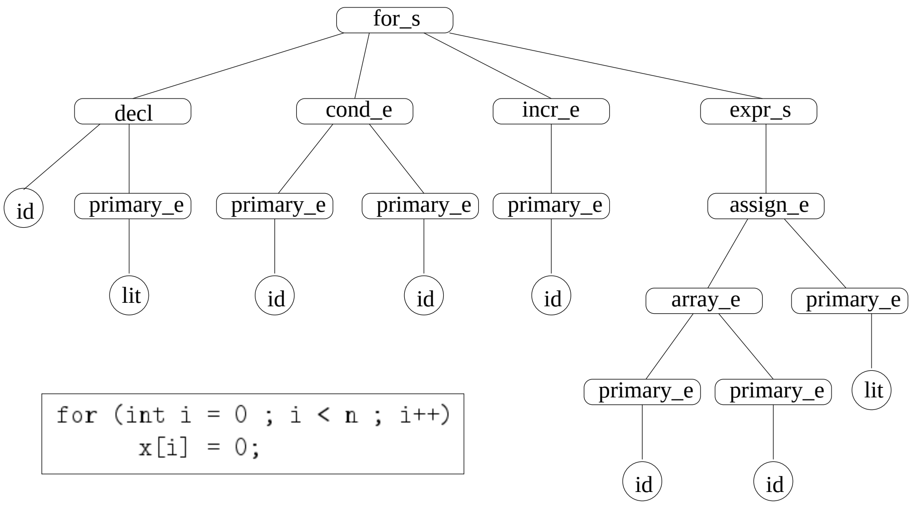


FIG. 1.30: Arbre syntaxique annoté.

Une méthode de comparaison d'arbres syntaxiques annotés classique a été proposée par Baxter *et al.*. Les techniques se basent sur une approche par hachage que nous avons exposées dans la section 1.2.2.3 de ce manuscrit dont le principe est de classifier les sommets des arbres syntaxiques par ressemblance. Deux sommets appartenant à un même groupe correspondent à deux blocs de code clonés.

Nous avons proposé une nouvelle méthode, pour une tâche non encore adressée dans le domaine de l'aide à la maintenance logicielle. La méthode s'inspire de la détection de clones dans le code. L'objectif de notre application est de visualiser l'évolution de code au fil du temps par un suivi du code stable. La méthode s'appuie sur une technique de recherche de motifs similaires entre deux versions consécutives d'un même fichier pour détecter le code stable que l'on peut ensuite visualiser dans l'interface graphique. Le détail de la méthode est présenté dans le chapitre 5.

Chapitre 2

Une heuristique pour la comparaison de graphes

Dans ce chapitre, nous proposons une méthode heuristique pour l'identification de motifs similaires entre des graphes autour de laquelle s'articule ce travail de thèse. La méthode proposée est une méthode générique qui peut être appliquée à différentes problématiques, pour lesquelles les structures à comparer spécifiques au domaine d'application ont des propriétés variables. Il s'agit de présenter visuellement, via une coloration des sommets des graphes, le résultat de la mise en correspondance de motifs similaires. En effet, les recherches de la communauté de visualisation d'information ("InfoViz") montrent clairement que l'utilisation de représentations visuelles de l'information permet un traitement plus rapide par l'être humain (dans le cerveau humain, 70% des récepteurs et 40% du cortex sont dévolus au processus de la vision [139]). Le résultat de la mise en correspondance permet par ailleurs d'évaluer la similarité entre les structures comparées. Ainsi, dans certaines applications, seule la mesure de similarité peut être exploitée sans avoir besoin d'afficher les structures similaires.

Dans ce chapitre, nous présentons le principe général de la méthode étudiée dans le cadre de cette thèse, les améliorations spécifiques à chaque domaine d'application étant définies dans les chapitres correspondants (voir chapitre 4 pour la reconnaissance d'objets dans la vidéo et chapitre 5 pour la technique de suivi de code similaire dans un projet logiciel).

2.1 Description de l'heuristique

La méthode heuristique de reconnaissance de motifs similaires dans des graphes que nous avons étudiée dans le cadre de cette thèse a été initialement proposée lors du concours Infovis 2003 [52] pour répondre à différentes tâches concernant la comparaison de structures arborescentes. Étant donnés deux arbres correspondants à deux versions d'un système de fichiers, il était question de proposer une (des) méthode(s) d'évaluation permettant entre autres, de détecter **visuellement** les éléments suivants :

- les changements entre les deux versions (en général/dans un sous-arbre en particulier)

– les suppressions/ajouts de sommets
– les déplacements de sommets ou de sous-arbres

Les tâches particulières posées pour le concours Infovis concernent la problématique de comparaison de structures arborescentes et plus particulièrement la recherche des motifs *similaires* (ou quasi-similaires) entre deux versions d'un même arbre. On dit que deux motifs sont similaires (ou quasi-similaires) s'ils sont proches en terme de structure. En quelque sorte, il existe peu de différences (sommets en plus/en moins, arête en plus/en moins) entre les deux sous-graphes représentant les motifs. Pour répondre à ces tâches spécifiques, Auber *et al.* ont proposé la solution logicielle EVAT [7], **interface de visualisation** permettant à l'utilisateur d'identifier visuellement, par la coloration des sommets et arêtes, si l'une des situations précédemment énoncées est présente entre les deux versions d'arborescences observées.

Dans ce logiciel, la méthode d'évaluation de motifs similaires entre les arbres proposée est une méthode heuristique basée sur l'étude des paramètres structurels calculés sur les sommets. L'algorithme, dans sa version initiale, se décompose en deux phases. La première étape consiste à regrouper les sommets en différentes familles en fonction de leurs caractéristiques structurelles (*e.g.* le degré, le nombre de sommets à distance n, le nombre de cliques de taille k qui contiennent le sommet, la distance à la racine, la taille du sous-arbre, *etc*). On obtient alors un étiquetage des sommets, chaque sommet portant une et une seule étiquette, deux sommets portant la même étiquette étant des sommets proches du point de vue des vecteurs de caractéristiques structurelles : on dit que les sommets sont ϵ-*similaires* (Déf. 2.2, Sec. 2.1.1.2). Dans l'interface de visualisation du logiciel EVAT, un plongement de l'intervalle des valeurs des étiquettes vers une palette de couleur est utilisé. Ainsi, à chaque valeur d'étiquette différente est associée une couleur distincte. À l'image des travaux de Baxter [17] qui utilise une méthode de hachage pour regrouper les sous-arbres potentiellement similaires, cette première phase de l'heuristique consiste à regrouper les sommets structurellement similaires pour limiter le nombre de comparaisons de sous-structures.

Dans une deuxième phase, il est question de raffiner la classification par familles de sommets similaires en identifiant des paires de motifs similaires. On procède par construction descendante progressive : étant donnés deux sommets appartenant à la même famille, on identifie un début de motif commun si les ensembles des successeurs de chacun des deux sommets sont proches en terme d'étiquetage : on parle ici de τ-similarité (Déf. 38, section 2.1.2.1). Si c'est le cas, nous venons d'identifier un début de motif similaire et nous essayons de continuer la reconnaissance par propagation du motif récursivement sur les enfants des sommets initiaux jusqu'à atteindre les feuilles ou jusqu'à ce que le processus de reconnaissance échoue. La détection de motifs engendre une propagation de l'étiquette (et donc de la couleur) des sommets parents vers les enfants. On peut alors visualiser les sous-structures communes aux deux arborescences : deux motifs similaires sont étiquetés par une seule et même étiquette et sont donc colorés par une même teinte.

La figure 2.1 montre un exemple de l'application de la méthode heuristique sur deux versions d'une arborescence de fichiers. Une même couleur suggère des motifs similaires. Dans cet exemple, on remarque que l'arborescence n'a pas subi de modification majeure car on ne discerne aucune différence visuelle.

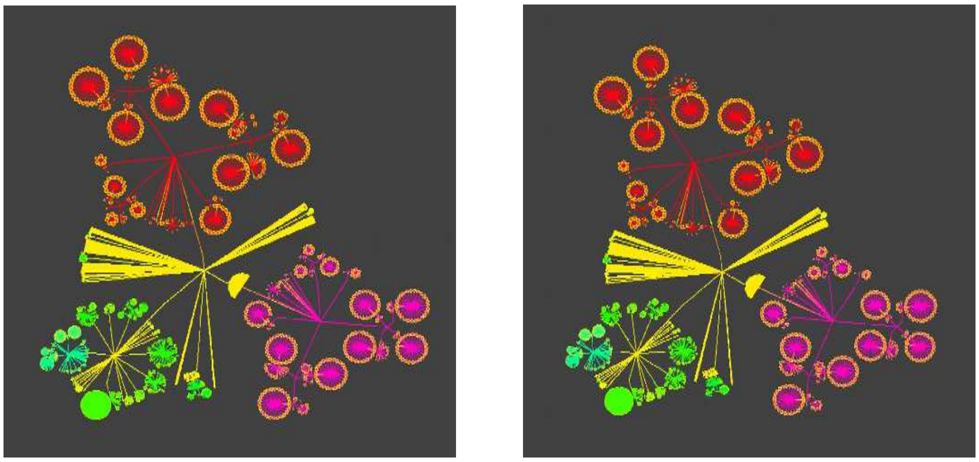


Fig. 2.1: EVAT : Visualisation de motifs communs entre deux versions d'un système de fichiers.

En filtrant les sommets appartenant à des motifs identifiés comme motifs similaires, on peut visualiser les endroits dans l'arbre ou des modifications ont été apportées. La figure 2.2 montre un exemple de détection de modifications dans une portion de l'arbre qui a été zoomée. Un zoom sur le contenu de l'un des répertoires de la portion de l'arborescence affichée sur la figure 2.2 est montré sur la figure 2.3. On remarque que l'utilisateur propriétaire de l'arborescence a ajouté plusieurs éléments dans ce répertoire.

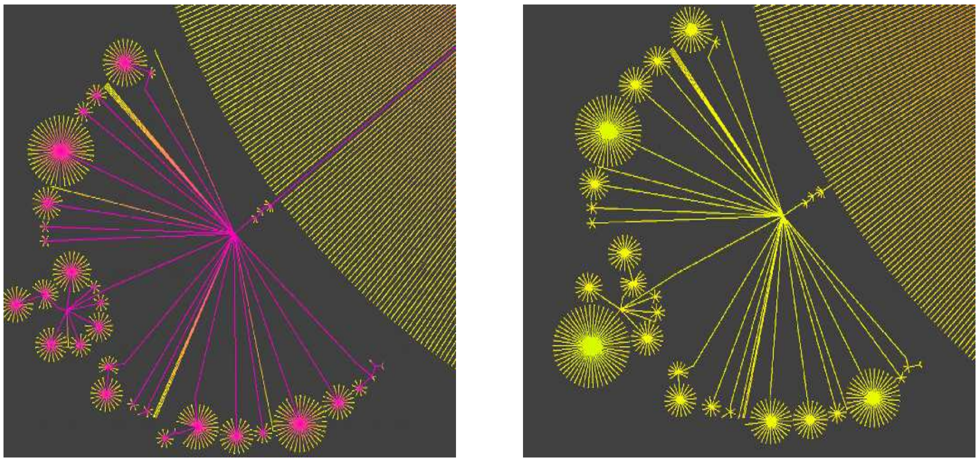


Fig. 2.2: EVAT : Visualisation des différences entre les deux versions d'un système de fichiers de la figure 2.1.

Il est important de remarquer que l'heuristique n'est pas nécessairement destinée à visualiser le résultat de la reconnaissance de motifs. En effet, la technique d'identification des motifs similaires entre deux structures permet d'évaluer à quel point les structures sont proches. L'application à la reconnaissance d'objets dans la vidéo (voir chapitre 4) ne nécessite pas la visualisation des parties similaires. Nous

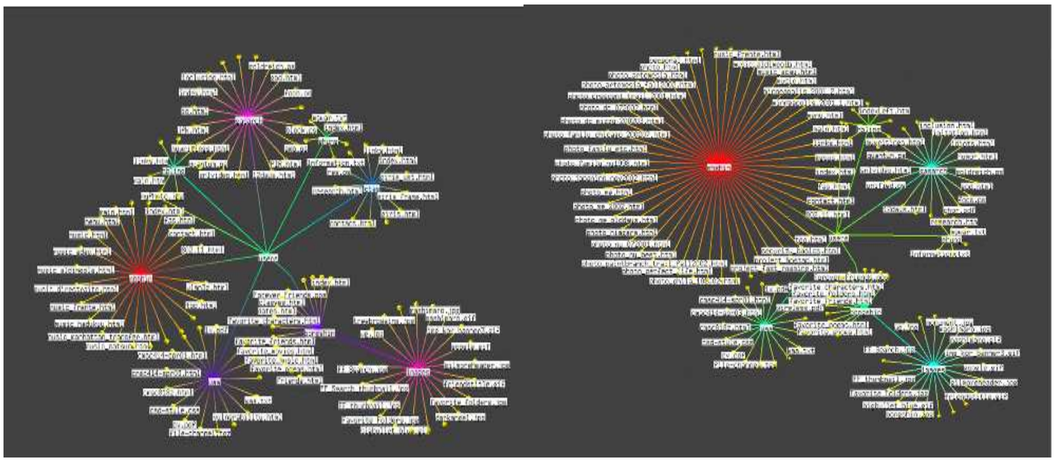


FIG. 2.3: EVAT : Zoom sur un répertoire de la figure 2.2.

utilisons simplement une évaluation de la similarité entre les objets basée sur la proportion des éléments des structures qui ont été reconnus comme appartenant à un motif similaire.

Nous venons de décrire brièvement le fonctionnement de l'heuristique telle qu'elle a été proposée pour la première fois. La suite de cette section est consacrée au détail de cette heuristique dans un cadre plus général. Bien que la méthode décrite ici a été initialement développée pour la comparaison d'arbres, elle est généralisable aux graphes. Dans la suite de ce chapitre, nous considérons que les structures à comparer sont des graphes généraux notés $G(V, E)$ et $G'(V', E')$.

Dans la section 2.1.1, nous détaillons la première phase de l'heuristique qui consiste à effectuer une classification des sommets en vue de regrouper les sommets structurellement similaires dans une même famille. La seconde phase, décrite dans la section 2.1.2 est dédiée à la reconnaissance des motifs structurels, basée sur l'étiquetage des sommets de la première phase.

2.1.1 Phase 1 : la classification des sommets

La première étape de la méthode est consacrée à la classification des sommets des graphes selon une mesure de similarité basée sur les propriétés structurelles des sommets. L'objectif est de diviser, à la manière de la phase de hachage décrite en section 1.2.2.3, l'ensemble des sommets $V \cup V'$ des graphes G et G' en des familles distinctes. Chaque famille comprend des sommets aux caractéristiques structurelles proches. Les caractéristiques structurelles considérées sont liées au type d'application. Cette classification est une première étape à la comparaison des sous-structures des graphes : deux sous-graphes dont les sommets racines ont des caractéristiques structurelles (locales et/ou globales) proches sont intéressants à comparer.

La classification des sommets en familles distinctes s'appuie sur un ensemble de caractéristiques associées à chaque sommet.

2.1.1.1 Calcul des métriques normalisées

Les métriques utilisées pour évaluer la similarité entre deux sommets peuvent correspondre à des métriques intrinsèques, calculées sur les sommets d'un graphe, indépendamment des informations propres aux données étudiées. Il est aussi envisageable d'exploiter les caractéristiques extrinsèques propres aux données.

Dans nos travaux, nous accordons une grande importance à l'aspect structurel pour la comparaison de graphes. En effet, quelle que soit l'application visée, la structure représente une caractéristique pertinente qu'il est important de considérer pour la comparaison. Parmi les caractéristiques structurelles, nous pouvons citer par exemple, pour un sommet u, le degré, le nombre de sommets à distance n de u, le nombre de cliques de taille k qui contiennent u ; si l'on considère des arbres, la distance à la racine, la taille du sous-arbre enraciné en u, *etc.*

Pour une métrique μ, la valuation associée à un sommet $u \in V \cup V'$, correspond à la valuation normalisée $\tilde{\mu}(u)$ de la métrique μ selon l'équation :

$$\tilde{\mu}(u) = \frac{\mu(u) - \mu_{min}}{\mu_{max} - \mu_{min}} \tag{2.1}$$

avec $\mu_{min} = \min_{v \in V \cup V'} \mu(v)$ et $\mu_{max} = \max_{v \in V \cup V'} \mu(v)$.

Ainsi, pour un ensemble de n métriques $\{\mu_i\}_{i \in [1,n]}$, nous calculons pour chaque sommet $u \in V \cup V'$, le vecteur caractéristique associé $(\tilde{\mu}_1(u), \tilde{\mu}_2(u), \cdots, \tilde{\mu}_n(u))$.

2.1.1.2 Construction des familles de sommets ϵ-similaires

À cette étape, nous construisons les différentes familles $\mathcal{F}$ contenant des sommets aux propriétés similaires. Nous définissons un étiquetage λ des sommets tel que si deux sommets u et v appartiennent à une même famille $\mathcal{F}$, alors $\lambda(u) = \lambda(v)$.

Définition 34 (Sommets ϵ-similaires)
Soient μ_i, $i \in [1, \cdots, n]$ l'ensemble des métriques considérées et soit le seuil de similarité $\epsilon \in [0, 1]$. On dit que deux sommets u et v sont ϵ-similaires s'ils vérifient :

$$dist(u, v) < \epsilon \tag{2.2}$$

La notion de distance $dist(u, v)$ est définie entre les vecteurs caractéristiques de u et v. Dans ce manuscrit, $dist(u, v)$ correspond à la distance euclidienne entre les vecteurs.

La construction des familles de sommets ϵ-similaires se base donc sur une comparaison deux à deux des sommets des graphes. Une approche basique serait de choisir le premier sommet u non encore classé comme le représentant d'une nouvelle classe $\mathcal{F}$ de sommets $\epsilon - similaires$. Les éléments de la famille étant sélectionnés comme suit : chaque sommet v non encore classé, tel que le représentant u de la famille $\mathcal{F}$ et le sommet v sont ϵ-similaires, appartient à la famille $\mathcal{F}$. Pour cette approche basique, dans le pire des cas, le nombre de comparaisons s'élève à $\frac{n(n-1)}{2}$ (soit un élément par famille).

Nous avons choisi d'intégrer une stratégie de partitionnement plus efficace pour la construction des familles. L'algorithme est basé sur une variante de la méthode d'*arbre de couverture* décrite dans [18].

L'arbre de couverture

Pour définir l'arbre de couverture, il est nécessaire d'introduire dans un premier temps les deux définitions de *séparabilité* et de *couverture* suivantes :

Définition 35 (Séparabilité)
Soit un ensemble S d'éléments définis dans un espace métrique. Soit dist *une distance sur cet espace et soit θ un réel.*

On dit que l'ensemble S vérifie la condition de séparabilité de seuil θ si, pour tout $u, v \in S$, on a $dist(u, v) > \theta$.

Définition 36 (Couverture)
Soient un ensemble S d'éléments définis dans un espace métrique et S' un sous-ensemble de S. Soit dist *une distance sur cet espace et soit θ un réel.*

On dit que l'ensemble S' vérifie la condition de couverture (de seuil θ) de l'ensemble S si, pour tout $u \in S$, il existe $v \in S'$ tel que $dist(u, v) \leq \theta$.

Un arbre de couverture T défini pour un ensemble d'éléments S est un arbre multi-niveaux. Le niveau 0 correspond aux feuilles de l'arbre, et le niveau le plus élevé (dénoté i_{max}) correspond à la racine. À chaque niveau i, est associé un réel θ_i tel que $\theta_i < \theta_{i+1}$. Soit S_i l'ensemble des éléments au niveau i.

Définition 37 (Arbre de couverture)
L'arbre de couverture T défini pour l'ensemble d'éléments S vérifie les propriétés suivantes pour tout i :
- $S_i \subset S_{i-1}$
- *S_i est une couverture de l'ensemble S_{i-1}*
- *S_i vérifie la propriété de séparabilté*

La définition précédente induit que si un élément u est présent à un niveau i, alors il est aussi présent dans tous les niveaux $j < i$.

Prenons pour illustration l'exemple de la figure 2.4.a sur un ensemble S de 5 points dans le plan. Les cercles autour des éléments représentent les zones du plan à distance θ_0 (trait plein), θ_1 (larges pointillés) et θ_2 (pointillés resserrés). Ainsi, le sommet t est à une distance de s comprise entre θ_1 et θ_2. Un arbre de couverture possible est représenté en figure 2.4.b.

L'arbre de couverture n'est pas nécessairement unique. Dans cet exemple, le sommet s du niveau S_1 couvre les sommets s et t du niveau S_0. Une autre solution aurait pu considérer le sommet t au niveau S_1 comme représentant des sommets s et t. Les propriétés de séparabilité et de couverture restant inchangées.

En pratique, l'arbre de couverture est construit par insertions successives des éléments. Un nouveau sommet n est inséré au niveau S_i de seuil θ_i, si :
- pour tout $j > i$, il existe un sommet $s \in S_j$ qui couvre le sommet n, alors s est un ancêtre de n dans l'arbre de couverture,

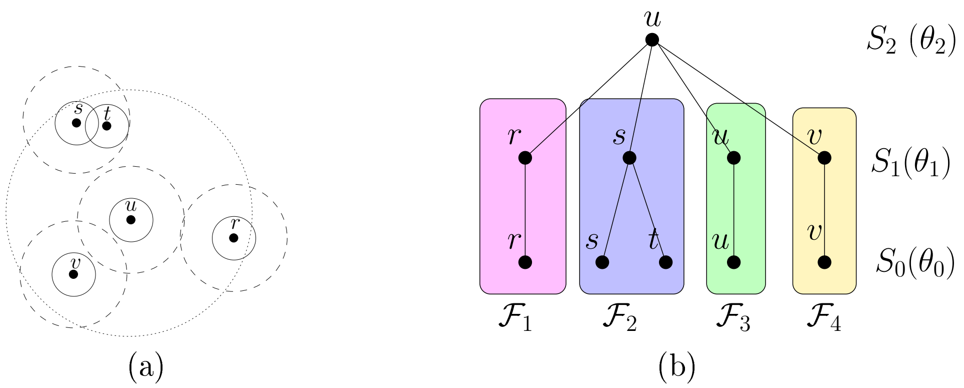


FIG. 2.4: Exemple d'arbre de couverture T. (a) Ensemble d'éléments S dans le plan et (b) un arbre de couverture possible pour S avec 3 niveaux. Les différentes familles ϵ-similaires pour $\epsilon = \theta_1$ sont désignées par les rectangles colorés.

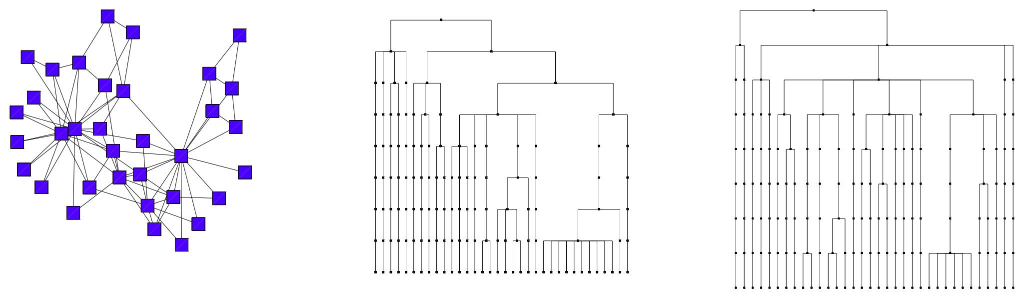
– pour tout $s \in S_i$, $dist(s, n) > \theta_i$

Dans le cas où plusieurs sommets d'un même niveau peuvent couvrir un sommet nouvellement inséré n, on sélectionne comme ancêtre de n le candidat qui minimise la distance avec n.

La composition des familles de sommets ϵ-similaires, avec $\epsilon = \theta_i$, se déduit à partir de l'arbre de couverture. Soit $R = \{u_0, u_1, \ldots, u_k\}$ l'ensemble des sommets du niveau S_i. Alors, les éléments de R correspondent aux représentants respectifs des différentes familles $\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_k$. Les éléments d'une famille $\mathcal{F}_l$ de représentant u_l sont l'ensemble des sommets distincts présents dans le sous-arbre T_{u_l} enraciné en u_l. Dans l'exemple de la figure 2.4, en fixant $\epsilon = \theta_1$, on obtient 4 familles : $\mathcal{F}_1 = \{r\}$ de représentant r, $\mathcal{F}_2 = \{s, t\}$ de représentant s, $\mathcal{F}_3 = \{u\}$ de représentant u et $\mathcal{F}_4 = \{v\}$ de représentant v.

En considérant les remarques précédentes, il est clair que le contenu des familles peut varier en fonction de l'ordre dans lequel les sommets sont traités car la construction de l'arbre de couverture dépend de l'ordre d'insertion. La figure 2.5.a montre un graphe représentant les relations amicales existant entre les membres d'un groupe de karaté [144] et les arbres de couvertures correspondants à un traitement des noeuds selon le degré décroissant (Fig. 2.5.b) et selon le nombre de Strahler décroissant (Fig. 2.5.c).

Les arbres sont différents selon l'ordre dans lequel les sommets sont traités, et par conséquent les familles de sommets ϵ-similaires diffèrent. Dans [5], Auber propose un algorithme de dessin de graphes incrémental pour la visualisation progressive de la structure à afficher. Le principe est de calculer un ordre sur les sommets afin d'afficher progressivement (en proposant à l'utilisateur le rendu à différentes étapes de calcul du dessin) la structure du graphe. Dans ce contexte, il est important de faire apparaître dès les premiers calculs la forme générale du graphe afin d'offrir à l'utilisateur un vision globale de la structure qu'il souhaite visualiser. Pour ce faire, Auber utilise un ordre sur les sommets : en considérant les sommets par ordre inverse



(a) (b) (c)

FIG. 2.5: Exemple d'arbres de couverture T. (a) le graphe et (b) l'arbre de couverture correspondant au traitement des sommets triés par degré décroissant et (c) par nombre de Strahler décroissant.

de leur nombre de Strahler, il semble que la vue schématique de l'image finale (moins de 10% des éléments affichés) donne une bonne représentation de l'allure générale du graphe.

Il semble donc que le nombre de Strahler traduit l'importance d'un sommet du point de vue de la structure globale. Nous avons choisi, pour construire l'arbre de couverture d'un graphe, de traiter les sommets dans l'ordre inverse de leur nombre de Strahler, positionnant ainsi les sommets les plus importants comme représentants des familles de sommets.

La procédure d'étiquetage des sommets est décrite par l'algorithme 2.1. On rappelle que T_u désigne le sous-arbre enraciné en u, $prof_T(u)$ désigne la distance du sommet u à la racine de l'arbre T (Déf. 25), et $\sigma(u)$ correspond au nombre de Strahler du sommet u (Déf. 29).

Données: Le niveau hiérarchique i considéré pour la construction
des familles (correspondant au seuil de tolérance ϵ_i fixé)

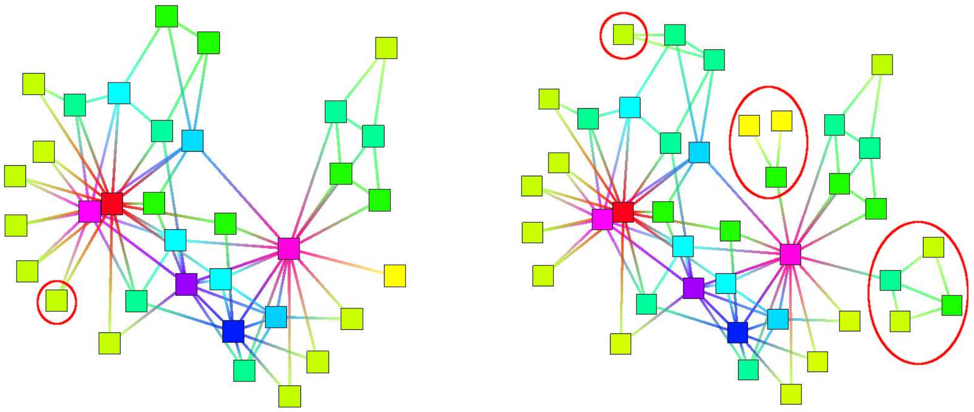
$S := V \cup V'$
trier S par σ décroissant;
$T :=$ arbre de couverture de l'ensemble trié S
$R := \{u \in T \mid prof_T(u) = i\}$
Entier $L := 1$;
Pour $u \in R$ **Faire**
    $U := v \in T_u$
    Pour $v \in U$ **Faire**
        $\lambda(v) := L$
    FinPour
    $L := L + 1$;
FinPour

Algorithme 2.1: Procédure d'étiquetage des familles

À l'issue de l'algorithme, on obtient un étiquetage λ des sommets de G et G' en L familles de sommets similaires. On note que la numérotation des familles respecte l'ordre d'insertion sur les sommets, par conséquent, l'ensemble R des représentants des familles est trié par ordre de Strahler décroissant. Ainsi, les familles correspondant aux sommets de plus fort Strahler sont les familles de plus faible valeur d'étiquette λ. Par la suite, on note $\mathcal{F}_l$ la famille de sommets u d'étiquette $\lambda(u) = l$.

Complexité Il a été prouvé par Don [43] que l'algorithme de construction de l'arbre de couverture d'un ensemble de points S défini sur un espace métrique E a pour complexité $O(2^{2dd} n \log A)$, avec dd la dimension doublante de E associée à l'arbre, et $A = \frac{d_{max}}{d_{min}}$ l'*aspect ratio* de l'ensemble S (d_{min} et d_{max} étant les distances minimale et maximale respectivement entre les paires d'éléments distincts de S).

La figure 2.6 montre un exemple de calcul de familles de sommets ϵ-similaires sur l'ensemble des sommets de deux versions du graphe de la figure 2.5.a. Les couleurs représentent les différentes valeurs d'étiquettes, une même couleur correspondant à une même étiquette. Le premier (Fig. 2.6.a) correspond aux données originales, que nous avons légèrement modifié manuellement en ajoutant et supprimant des sommets et des arêtes (entourés en rouge dans la figure 2.6) pour obtenir un second graphe (Fig. 2.6.b). Nous pouvons d'ores et déjà remarquer des ressemblances au niveau colorimétrique.



(a) (b)

FIG. 2.6: Exemple de classification en familles de sommets ϵ-similaires. (a) le graphe original du club de karaté et (b) le même graphe après modifications.

L'objectif est maintenant d'utiliser cette classification en vue d'identifier dans les deux graphes les sous-structures qui se correspondent. On cherche à reconnaître des motifs similaires par un processus de propagation de l'étiquette (donc de la coloration) des sommets vers les voisins si les motifs sont identifiés comme similaires. La propagation des étiquettes en cas de reconnaissance de motifs similaires engendre une coloration des graphes telle que les motifs de même couleur représentent les motifs similaires. Un exemple d'identification de motifs à partir des deux graphes de la figure 2.6 est montré sur la figure 2.7. On note dans cet exemple que les structures ne sont pas identiques (certains sommets ne sont pas inclus dans les motifs reconnus). On note cependant deux importants motifs stables sur les deux versions.

Le détail de la procédure de reconnaissance de motifs est décrit dans la section qui suit.

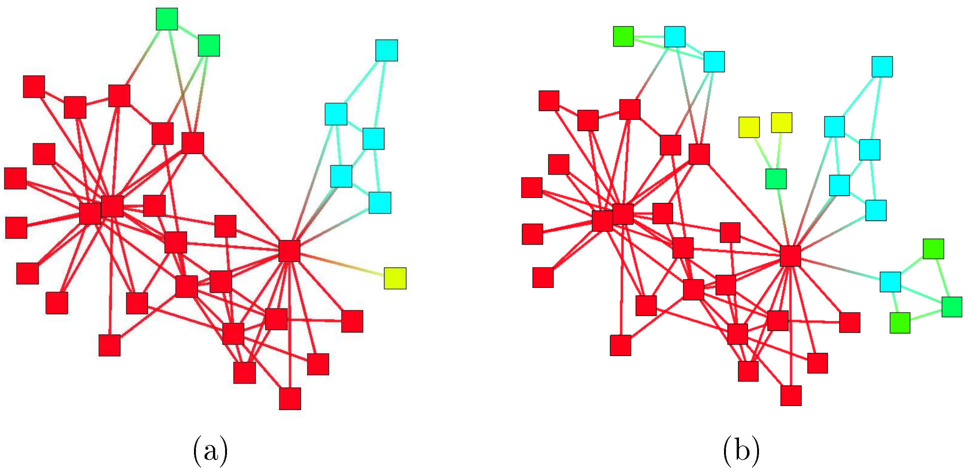


Fig. 2.7: Exemple de reconnaissance des motifs correspondant à la classification de la figure 2.6

2.1.2 Phase 2 : la construction des motifs similaires

L'objectif de cette seconde étape est l'identification de motifs dits *similaires* entre les deux structures à comparer G et G'. D'après la procédure de construction de familles de sommets ϵ-similaires (phase 1), on émet l'hypothèse que deux sommets appartenant à une même famille ont de fortes chances d'être deux sommets racines de sous-graphes similaires. Si l'on considère $u \in G$ et $u' \in G'$ tels que $\lambda(u) = \lambda(u')$, alors les caractéristiques structurelles de u sont proches de celles de u' car les sommets sont dits ϵ-similaires (voir section 2.1.1.2) et il est probable que les sous-graphes G_u issu de u et $G_{u'}$ issu de u' soient similaires.

La procédure de reconnaissance de motifs considère donc des paires de sommets $\{u, u'\}$ telles que $u \in G$, $u' \in G'$ et $\lambda(u) = \lambda(u')$. La construction du motif similaire se fait par propagation : l'algorithme traite les sommets depuis les sommets considérés u et u' vers les voisins identifiés comme appartenant au motif, et récursivement sur les voisins des voisins. Les sommets considérés au départ sont les sommets appartenant à la famille dont le représentant a la plus forte valeur de Strahler (par construction, l'ordre des familles correspond à une valeur de Strahler décroissante).

Dans les cas particuliers des DAGs ou des arbres, la procédure est descendante : on examine récursivement les sommets depuis les parents vers leurs successeurs. Dans le cas des graphes généraux, on considère les voisins des sommets comparés.

L'idée est donc de confirmer la similarité entre deux sommets u et u' établie par la construction de familles de sommets ϵ-similaires, en comparant le voisinage de ces sommets pour identifier des structures similaires.

On note que toute décision est définitive : si un ensemble de sommets est considéré comme appartenant à un motif, alors on ne reviendra pas sur cette décision.

Nous proposons trois versions de l'heuristique pour l'identification de motifs par propagation, correspondant à trois différentes stratégies de comparaison sur la similarité des sous-structures.

Dans la première version, nous cherchons à mettre en correspondance les portions des voisinages directs par famille. Dans l'exemple de la figure 2.8.a, on considère qu'il existe un trop grand écart de cardinalité entre les ensembles de voisins de u et de v étiquetés par un rond bleu. Ainsi, ces sommets voisins ne sont pas inclus dans le motif représenté par la zone rouge de la figure 2.8.b. Par contre, les voisinages étiquetés d'un triangle vert sont assez proches en terme de cardinalité pour être inclus dans le motif.

Dans la deuxième version, on examine les ensembles des sommets voisins dans leur intégralité : il est nécessaire que les voisinages complets soient proches. Dans cette version, la construction des motifs similaires se fait “par niveaux” : on propage toujours sur l'ensemble des voisinages si ces derniers se correspondent. Ainsi, pour une paire de sommets, soit tous les voisins sont inclus dans le motif, soit aucun. Dans l'exemple de la figure 2.8.a, les ensembles des voisins de u et de v sont trop différents pour être considérés comme appartenant à des motifs similaires. Par contre, les compositions des voisinages de u_3 et u'_5 sont suffisamment proches pour constituer des motifs similaires.

Enfin, dans la troisième version, on considère non pas le voisinage direct des sommets, mais les sous-graphes entiers issus des sommets en respectant l'orientation des arêtes. Ainsi, cette dernière version n'est pertinente que pour les graphes orientés, et plus particulièrement les DAGs et les arbres, car elle exploite la notion de hiérarchie entre les sommets traduite par l'orientation des arêtes. Dans cette version, si un sommet appartient à un motif, alors l'ensemble des sommets du sous-DAG (ou sous-arbre) enraciné en ce sommet appartiennent aussi au motif. Dans l'exemple de la figure 2.8.a, les sous-arbres issus de v_3 et v_5 sont considérés comme proches et constituent des motifs similaires (Fig. 2.8.d).

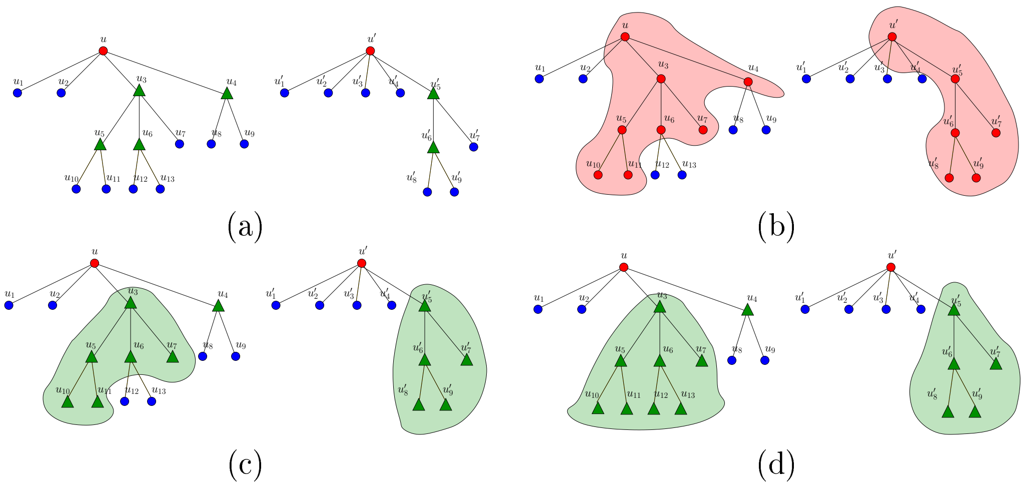


Fig. 2.8: Les différentes versions de propagation d'étiquette. (a) Deux graphes étiquetés en familles de sommets ϵ-similaires et (b) Version 1 : propagation sur les voisins par classe, (c) Version 2 : propagation sur l'ensemble des voisins et (d) Version 3 : propagation sur le sous-DAG (ou sous-arbre).

2.1.2.1 Évaluation de la similarité de voisinage

Le principe général de l'heuristique (dans ses différentes versions) consiste, pour chaque paire de sommets ϵ-similaires, à évaluer la *similarité* de leur voisinage. On utilise pour cela une mesure de dissimilarité $D(C, C')$ entre deux ensembles de sommets C et C' (définie, pour chacune des trois variantes, dans les paragraphes 2.1.2.2, 2.1.2.3 et 2.1.2.4). Cette mesure se base sur la distribution des étiquettes des sommets de C et C'. Intuitivement, elle peut être assimilée à une distance d'édition entre les étiquetages des sommets de C et C' en considérant les opération d'ajout et de suppression uniquement.

Si $D(C, C')$ est inférieure à un seuil de similarité τ fixé, alors on considère que les deux étiquetages sont cohérents et les ensembles de sommets sont dits τ-similaires : C et C' correspondent à un même motif. On énonce ainsi la définition de motifs τ-similaires comme suit :

Définition 38 (Motifs τ-similaires)
On dit que deux ensembles de sommets étiquetés C et C' sont τ-similaires si la mesure de dissimilarité d'étiquetage $D(C, C')$ vérifie :

$$D(C, C') \leq \tau$$

La procédure de reconnaissance est appliquée récursivement sur les sommets nouvellement inclus dans le motif.

2.1.2.2 Version 1 : propagation sur les voisins par classe

Dans cette première version de la méthode, on cherche à mettre en correspondance les portions des voisinages par étiquette. La mesure de dissimilarité d'édition D_1 est calculée pour chaque valeur d'étiquette $l \in [1, \cdots, L]$ selon la définition 39. On note $C(u)$ l'ensemble des voisins de u et $C_l(u)$ désigne l'ensemble des sommets $v \in C(u)$ tels que $\lambda(v) = l$.

Définition 39 (Mesure de dissimilarité d'étiquetage, version 1 : D_1)
Considérons $C_l(u)$ et $C_l(u')$ les ensembles des voisins de u et u' respectivement d'étiquette l. La distance d'édition d'étiquetage D_1 se calcule comme suit :

$$D_1(C_l(u), C_l(u')) = abs(|C_l(u)| - |C_l(u')|)$$

La mesure D_1 est une distance et correspond à la différence de cardinalité entre deux ensembles de sommets de même étiquette.

Si, pour une valeur d'étiquette l, la distance $D_1(C_l(u), C_l(u'))$ est inférieure au seuil τ_1 fixé, alors on a identifié une portion de motif τ-similaire (Déf. 38).

L'algorithme 2.2 décrit la procédure récursive de reconnaissance de motifs par propagation. On construit un étiquetage α sur les sommets permettant l'identification des différents motifs. Une valeur $\alpha(u) = 0$ signifie que le sommet u n'appartient à aucun motif.

À l'initialisation, l'ensemble des valeurs $\alpha(u)$ sont fixées à 0. La procédure de l'algorithme principal est décrite dans l'algorithme 2.3.

```
Données: Les ensembles de sommets S et S', la valeur du
motif courant α.

Pour (s, s') ∈ S × S' Faire
    Pour l Allant_de L A 1 Par_pas_de −1 Faire
        Si D_1(C_l(s), C_l(s')) < τ_1 Faire
            Si α(s) = 0 Faire
                // On a identifié un début de nouveau motif
                α := α + 1
                α(s) := α
                α(s') := α
            FinSi

            Pour u ∈ C_l(s) ∪ C_l(s') Faire
                α(u) = α
            FinPour

            propagation(C_l(s), C_l(s'), α)
        FinSi
    FinPour

    Si α ≠ 0 Faire
        S_l := S_l{u ∈ S_l | α(u) = α}
        S'_l := S'_l{u ∈ S'_l | α(u) = α}
    FinSi
FinPour
```

Algorithme 2.2: Procédure récursive de reconnaissance de motifs par propagation

```
α := 0
Pour u ∈ V ∪ V' Faire
    α(u) := 0
FinPour

Pour l Allant_de L A 1 Par_pas_de −1 Faire
    S_l := {u ∈ V ∩ F_l | α(u) ≠ 0}
    S'_l := {u ∈ V' ∩ F_l | α(u) ≠ 0}

    propagation(S_l, S'_l)
FinPour
```

Algorithme 2.3: Procédure principale de reconnaissance de motifs

Soit l'exemple de la figure 2.9. Les sommets u et u' sont ϵ-similaires ; le sommet u possède six voisins répartis en trois familles : la famille $\mathcal{F}_a$ (cercle bleu) comprend 1 sommet, la famille $\mathcal{F}_b$ (triangles verts) comprend 2 sommets et la famille $\mathcal{F}_c$ représentée par les carrés jaunes, qui est composée de 3 sommets. Les voisins de u' sont répartis ainsi : 1 sommet dans la famille $\mathcal{F}_a$, 3 sommets dans la famille $\mathcal{F}_b$ et 1 sommet dans la famille $\mathcal{F}_c$.

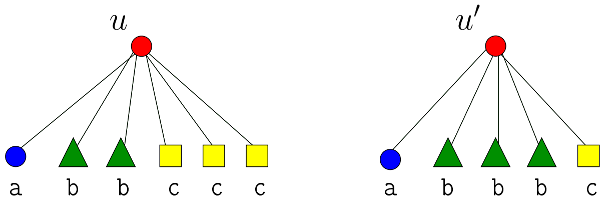


FIG. 2.9: Exemple d'arbres étiquetés en famille de sommets ϵ-similaires.

Si l'on fixe le seuil τ_1 à la valeur 1. On a

- $|C_a(u)| = 1$ et $|C_a(u')| = 1$: la distance D_1 est nulle, donc les sommets de la famille $\mathcal{F}_a$ appartiennent à un même motif τ-similaire.
- $|C_b(u)| = 2$ et $|C_b(u')| = 3$: la distance D_1 est égale à 1. Comme $1 \leq \tau_1$, les éléments de la famille $\mathcal{F}_b$ sont considérés comme τ-similaires et appartiennent au même motif.
- $|C_c(u)| = 3$ et $|C_c(u')| = 1$: la distance D_1 s'élève à 2. On a $2 > \tau_1$, alors il y a une trop grande différence de cardinalité entre les deux ensembles, et les éléments de la famille $\mathcal{F}_c$ ne sont pas inclus dans le motif τ-similaire.

La figure 2.10 montre le résultat de la propagation de motif τ-similaire sur l'exemple de la figure 2.9.

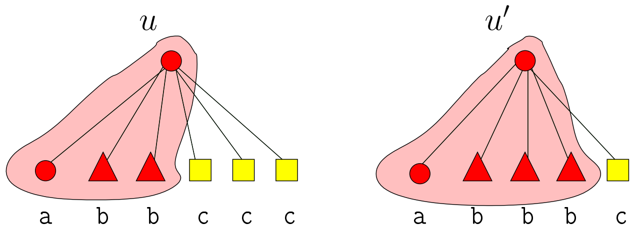


FIG. 2.10: Exemple de reconnaissance d'un motif τ-similaire (version 1), $\tau = 1$.

On note que dans les deux procédures, les familles de sommets sont traitées par ordre de leur étiquette λ. Nous avons mentionné, dans la section 2.1.1.2, que les familles de plus faible étiquette correspondent aux familles regroupant les sommets de fort Strahler, et que les sommets dont le Strahler est élevé semblent correspondre aux sommets les plus importants du point de vue structurel. Le fait de considérer d'abord les sommets qui correspondent au squelette de la structure permet une construction depuis les sommets centraux vers les feuilles, assurant ainsi une reconnaissance de grands motifs avant d'identifier de multiples petites structures similaires.

2.1.2.3 Version 2 : propagation sur l'ensemble des voisins

Dans cette seconde version, il est nécessaire, pour la reconnaissance d'un motif τ-similaire, que les ensembles des sommets voisins de deux sommets ϵ-similaires u et u', dans leur intégralité, soient proches en terme d'étiquetage. Dans ce cas, la mesure de dissimilarité d'étiquetage D_2 considère tout le voisinage des sommets u et u'.

Définition 40 (Mesure de dissimilarité d'étiquetage, version 2 : D_2)
Soient $C(u)$ et $C(u')$ les ensembles des voisins de u et u' respectivement, quelle que soit leur étiquette. La mesure de dissimilarité d'étiquetage D_2 est définie par

$$D_2(C(u), C(u')) = \sum_{l \in [1, \cdots, L]} D_1(C_l(u), C_l(u'))$$

D_2 vérifie les propriétés d'une distance.

L'algorithme 2.4 décrit la procédure récursive de reconnaissance de motifs par propagation. Seules les lignes 6,14 et 18-20 diffèrent de l'algorithme 2.2 de la version 1 de l'heuristique. La procédure principale reste inchangée (voir l'algorithme 2.3).

```
1  Données: Les ensembles de sommets S et S', la valeur du
2  motif courant α.
3
4    Pour (s, s') ∈ S × S' Faire
5        Pour l Allant_de L A 1 Par_pas_de −1 Faire
6            Si D2(C(s), C(s')) < τ2 Faire
7                Si α(s) = 0 Faire
8                    // On a identifié un début de nouveau motif
9                    α := α + 1
10                   α(s) := α
11                   α(s') := α
12               FinSi
13
14               Pour u ∈ C(s) ∪ C(s') Faire
15                   α(u) = α
16               FinPour
17
18               Pour l' Allant_de L A 1 Par_pas_de −1 Faire
19                   propagation(Cl(s), Cl(s'), α)
20               FinPour
21           FinSi
22       FinPour
23
24       Si α ≠ 0 Faire
25           Sl := Sl{u ∈ Sl | α(u) = α}
26           S'l := S'l{u ∈ S'l | α(u) = α}
27       FinSi
28   FinPour
```

Algorithme 2.4: Procédure récursive de reconnaissance de motifs par propagation (version 2)

Prenons l'exemple de la figure 2.11 avec une valeur τ_2 fixée à 1. Les sommets représentés par un cercle bleu appartiennent à la famille $\mathcal{F}_a$, et les carrés verts correspondent aux sommets de la famille $\mathcal{F}_b$. Si l'on considère dans un premier temps la paire de sommets ϵ-similaires $\{u, u'\}$, on a $D_2(C(u), C(u')) = 1$. Les ensembles $C(u)$ et $C(u')$ appartiennent donc au motif τ-similaire.

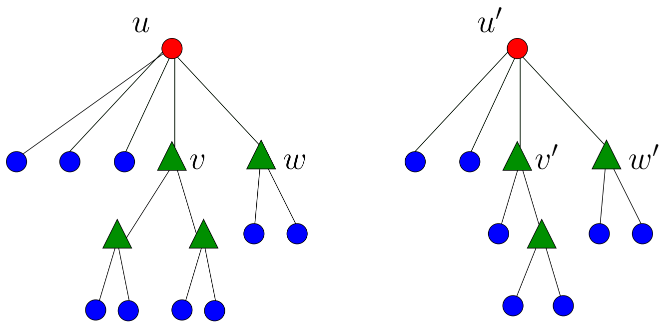


FIG. 2.11: Exemple d'arbres étiquetés en familles de sommets ϵ-similaires.

La procédure de reconnaissance de motifs est récursive sur les éléments nouvellement identifiés comme appartenant au motif τ-similaire (voir ligne 19 de l'algorithme

2.4). Ainsi, nous procédons à la comparaison des sommets par paires (et appartenant à une même famille) des ensembles $C(u)$ et $C(u')$. Les sommets de $C(u)$ et $C(u')$ de la famille $\mathcal{F}_a$ correspondent à des sommets dont l'ensemble des voisins sont déjà marqués comme appartenant à un motif. Il n'y a donc pas de comparaisons à effectuer. Nous comparons alors les sommets de $C(u)$ et $C(u')$ de la famille $\mathcal{F}_b$. On a $C_b(u) = \{v, w\}$ et $C_b(u') = \{v', w'\}$. Si l'on considère d'abord la paire $\{v, v'\}$, on a $D_2(C(v), C(v')) = 2$, donc les voisinages ne se correspondent pas au vu du seuil de tolérance τ_2 fixé. On considère alors la paire $\{v, w'\}$: $D_2(C(v), C(w')) = 4$, les ensembles de voisins ne se correspondent pas. Il reste désormais la paire $\{w, w'\}$: $D_2(C(w), C(w')) = 0$, alors on a identifié un prolongement du motif. Il reste enfin la paire $\{w, v'\}$. On a $D_2(C(w), C(v')) = 2$, donc les voisinages ne se correspondent pas et la procédure de reconnaissance récursive des motifs issus de u et u' est terminée. Le résultat de la reconnaissance de motifs est montré sur la figure 2.12.

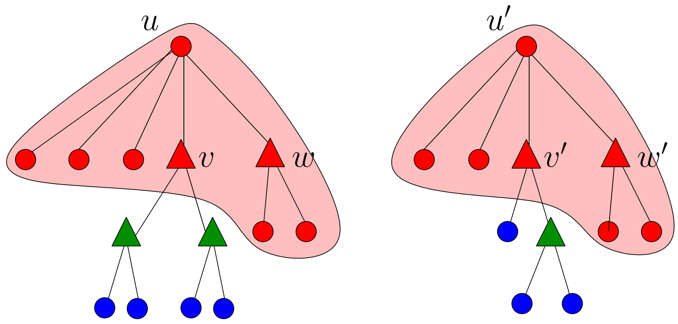


FIG. 2.12: Exemple de reconnaissance d'un motif τ-similaire (version 2), $\tau = 1$.

2.1.2.4 Version 3 : propagation sur le sous-DAG (ou sous-arbre)

La troisième variante de l'heuristique s'applique aux arbres et aux DAGs. Dans cette version, un motif identifié prenant racine sur le sommet u, correspond nécessairement à tout le sous-arbre (ou sous-DAG) T_u enraciné en u. On cherche à identifier des structures qui se correspondent dans leur intégralité (*i.e.* l'ensemble des sommets qui constituent le sous-arbre ou le sous-DAG), et non par niveau sur le voisinage comme dans les deux versions précédentes.

Pour déterminer si deux sommets u et u' (et donc les deux sous-arbres ou sous-DAG T_u et $T_{u'}$) sont τ-similaires, nous vérifions dans un premier temps que les deux sommets sont localement cohérents : deux sommets u et u' τ-similaires (version 3) sont nécessairement τ-similaires (version 2, voir section 2.1.2.3). Si effectivement u et u' sont τ-similaires (vers.2), alors, pour déterminer leur τ-similarité (vers.3), on se base sur le rapport entre le nombre de sommets des deux sous-structures T_u et $T_{u'}$ qui ont été identifiés comme τ-similaires (vers.3) sur le nombre total de sommets de T_u et $T_{u'}$. La mesure de dissimilarité d'étiquetage pour une paire de sommets $D_3(u, u')$ se construit, dans cette version de l'heuristique, à partir des sous-DAGs (ou sous-arbres) T_u et T'_u.

La mesure de dissimilarité d'étiquetage D_3 se calcule sur les ensembles T_u et $T_{u'}$ des sous-DAGs ou sous-arbres enracinées en u et u' respectivement, quelle que soit leur étiquette.

Définition 41 (Mesure de dissimilarité d'étiquetage, version 3 : D_3)
Soient T_u et $T_{u'}$ les sous-arbres (ou sous-DAGs) enracinés en u et u' respectivement. Soient $\kappa(u)$ et $\kappa(u')$ le nombre de sommets de T_u et $T_{u'}$ respectivement ayant été mutuellement identifiés comme τ-similaires. La valeur D_3 est définie par :

$$D_3(u, u') = 1 - \frac{\kappa(u) + \kappa(u')}{|T_u| + |T_{u'}|}$$

avec $|T_u|$ (resp. $|T_{u'}|$) le nombre de sommets de T_u (resp. $T_{u'}$).
Par convention, deux feuilles sont considérées comme τ-similaires.

L'algorithme 2.5 décrit la procédure de construction de motifs τ-similaires.

```
1   α := 0;
2
3   Pour u ∈ V ∪ V' Faire
4       α(u) := 0
5   FinPour
6
7   Pour l Allant_de L A 1 Par_pas_de −1 Faire
8       S_l := {u ∈ V ∩ F_l}
9       S'_l := {u ∈ V' ∩ F_l}
10
11      Pour (s, s') ∈ S_l × S'_l Faire
12          Pour u ∈ T_s ∪ T_s' Faire
13              κ(u) := 0;
14          Si calcule_D3(s, s') < τ3 Faire
15              // On a identifié un nouveau motif
16              α := α + 1
17              Pour u ∈ T_s ∪ T_s' Faire
18                  α(u) = α
19              FinPour
20              S_l := S_l {u ∈ S_l | α(u) = α}
21              S'_l := S'_l {u ∈ S'_l | α(u) = α}
22          FinSi
23      FinPour
24  FinPour
```

Algorithme 2.5: Procédure de construction de motifs similaires (version 3)

La fonction d'identification de motifs `calcule_`D_3 appelée à la ligne 14 de l'algorithme 2.5 retourne un booléen (vrai/faux) selon que la mesure D_3 est supérieure ou non au seuil de tolérance τ_3. Le détail de cette fonction est donné par l'algorithme 2.6.

Soit l'exemple de la figure 2.13. Les sommets sont étiquetés en 3 familles distinctes : la famille $\mathcal{F}_1$ est représentée par les cercles bleus, la famille $\mathcal{F}_2$ correspond aux carrés vert, et les cercles rouges représentent la famille $\mathcal{F}_3$. La reconnaissance de motifs s'effectue sur la paire de sommets u et u'.

Nous fixons les seuils de tolérance $\tau_2 = 2$ et $\tau_3 = 0.4$. Dans un premier temps, on teste si les ensembles de successeurs $C(u)$ de u et $C(u')$ de u' sont τ-similaires, vers.2 (ligne 15 de l'algorithme 2.6). On a $D_2(C(u), C(u')) = 2$, donc les sommets u et u' sont une paire de candidats potentiels à la τ-similarité vers.3.

```
1   Données: deux sommets $u$ et $u'$
2   Retour: un booléen indiquant si les sous−structures $T_u$
3             et $T_{u'}$ enracinées en $u$ et $u'$ respectivement
4             sont $\tau$−similaires
5
6   Si $u$ et $u'$ sont des feuilles Faire
7       $\kappa(u) := 1$
8       $\kappa(u') := 1$
9       Retourner vrai
10  FinSi
11
12  $C_(u)$ := les successeurs directs de $u$
13  $C_(u')$ := les successeurs directs de $u'$
14
15  Si $D_2(C_u, C_{u'}) > \tau_2$ Faire
16      Retourner faux
17  FinSi
18
19  Pour $l$ Allant_de $L$ A $1$ Par_pas_de $-1$ Faire
20      $C_l(u) := \{v \in C(u) \cap \mathcal{F}_l\}$
21      $C_l(u') := \{v \in C(u') \cap \mathcal{F}_l\}$
22
23      Pour $(s, s') \in C_l(u) \times C_l(u')$ Faire
24          Si calcule_$D_3(s, s') < \tau_3$ Faire
25              $C_l(u) := C_l(u)\ \{s\}$
26              $C_l(u') := C_l(u')\ \{s'\}$
27          FinSi
28      FinPour
29  FinPour
30
31  $\kappa(u) := 1 + \sum_{v \in C_l(u)} \kappa(v)$
32  $\kappa(u') := 1 + \sum_{v \in C_l(u')} \kappa(v)$
33
34  Retourner $\frac{\kappa(u)+\kappa(u')}{|T_u|+|T_{u'}|} > \tau_3$
```

Algorithme 2.6: Procédure d'identification de motifs similaires (version 3)

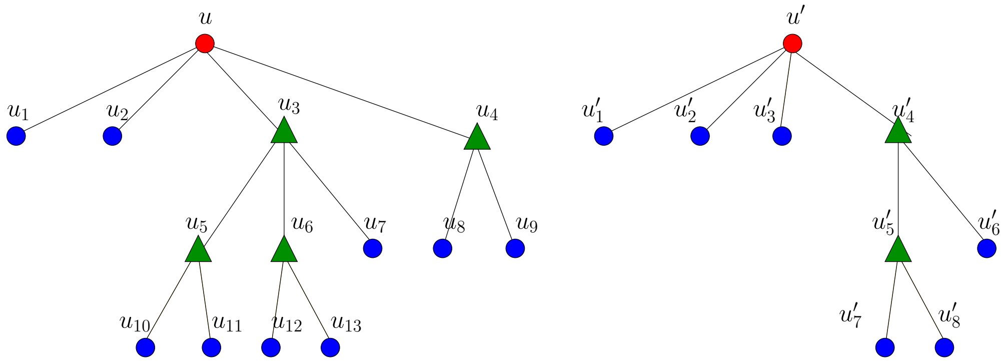


FIG. 2.13: Exemple d'arbres étiquetés en familles de sommets ϵ-similaires.

La procédure est alors récursive sur les successeurs directs de u et u'. Ainsi, pour chaque famille de sommets ϵ-similaires, on cherche une correspondance entre les successeurs de u et les successeurs de u'. Prenons par exemple la famille $\mathcal{F}_1$ (cercles bleus). On a $C_1(u) = \{u_1, u_2\}$ et $C_1(u') = \{u'_1, u'_2, u'_3\}$. On effectue un appel récursif de la fonction `calcule_`D_3 sur la paire de sommets $\{u_1, u'_1\}$ (voir ligne 24 de l'algorithme 2.6). Les deux sommets étant des feuilles, ils sont identifiés comme τ-similaires vers.3 et la valeur de κ pour chacun des deux sommets est fixée à 1, car dans les sous-arbres T_{u_1} enraciné en u_1 et T_{u_2} enraciné en u_2, il y a exactement 1 sommet identifié comme τ-similaire vers.2. Il en est de même pour la paire de sommets $\{u_2, u'_2\}$. Cependant, comme les sommets u_1 et u_2 ont déjà été mis en correspondance, on ne trouve pas de sommet associé à u'_3, ainsi, la valeur $\kappa(u_3)$ reste nulle.

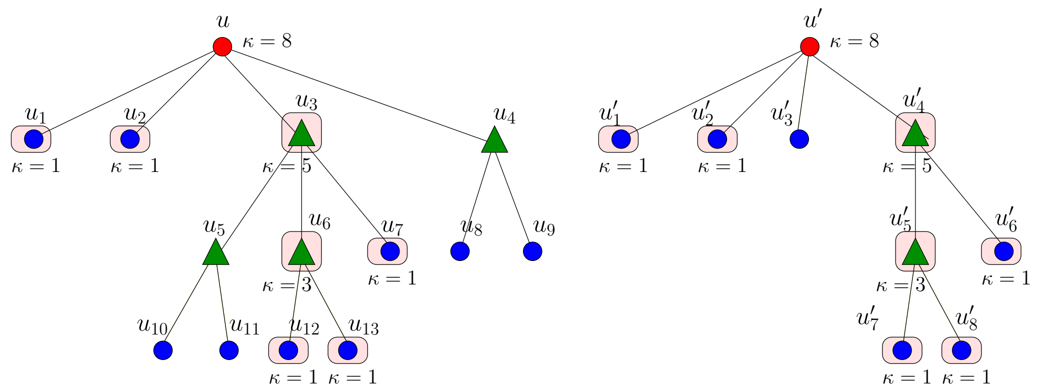


FIG. 2.14: Exemple de reconnaissance d'un motif τ-similaire (version 2).

Si l'on considère désormais les éléments de la famille $\mathcal{F}_2$ (triangles verts), on a $C_2(u) = \{u_3, u_4\}$ et $C_2(u') = \{u'_4\}$. On effectue l'appel `calcule_`D_3 sur la paire de sommets $\{u_3, u'_4\}$. On a $D_2(C(u_3), C(u'_4)) = 1$, donc on peut procéder à l'analyse du sous-arbre des sommets u_3 et u'_4 : on étudie les paires $\{u_5, u'_5\}$ qui ne se correspondent pas, puis $\{u_6, u'_5\}$ qui peuvent se correspondre.

Une fois encore, on effectue l'appel récursif sur les successeurs de u_6 et u'_5. Chacun des deux sommets possède exactement deux feuilles. L'algorithme identifie donc deux sommets τ-similaires vers.2 dans chaque arbre (les sommets u_{12}, u_{13}, u'_7 et u'_8), chacun d'entre eux a sa valeur κ fixée à 1 (voir lignes 7-8 de l'algorithme 2.6).

Nous pouvons désormais calculer la valeur κ des sommets u_6 et u'_5 : $\kappa(u_6) = 1 + \kappa(u_{12}) + \kappa(u_{13}) = 3$, $\kappa(u'_5) = 3$. La comparaison des sommets u_5 et u_6 s'achève par le calcul du rapport $D_3(u_5, u_6) = 1 - \frac{3+3}{3+3} = 0$ (voir ligne 34 de l'algorithme 2.6). Dans ce cas, les arbres se correspondent parfaitement : on a $D_3 = 0$. Donc les sommets u_5 et u_6 sont τ-similaires vers.3.

La figure 2.14 montre le résultat de l'algorithme sur l'exemple de la figure 2.13. Les sommets identifiés comme τ-similaires vers.3 sont encadrés d'un rectangle rose. Les valeurs de κ sont affichées en-dessous de chaque sommet. En l'absence de notification de κ, on a $\kappa = 0$.

Pour déterminer si les sommets u et u' sont τ-similaires vers.3, on calcule $D_3(u, u') = 1 - \frac{8+8}{14+9} = 0.31$. Comme τ_3 est fixé à 0.4, alors la proportion de sommets correspondants entre les deux sous-arbres T_u et $T_{u'}$ est suffisante pour la reconnaissance de motifs similaires.

Complexité Nous n'avons pas prouvé la complexité globale de l'étape de propagation des sommets. Nous donnons cependant ici quelques indications de complexité pour chacune des trois versions de la méthode.

Soient $G = (V, E)$ et $G' = (V', E')$ deux graphes et soit L le nombre de familles de sommets ϵ-similaires définies sur G et G'. Le nombre de comparaisons de sommets pour la phase de propagation est au pire $\sum_{i \in [1,L]} |\mathcal{F}_i \cap V||\mathcal{F}_i \cap V'|$. Dans ce cas, toutes les paires de sommets sont testées, ce qui implique que la propagation échoue systématiquement. Dans la pratique, ce cas extrême est très peu probable, ainsi, dès qu'une étape de propagation est effectuée, l'ensemble des sommets appartenant au motif nouvellement identifié sont exclus du processus de propagation par la suite. Ce qui a pour effet de réduire le nombre de comparaisons.

Pour une étape de propagation de motifs issus des sommets u et u', dans les versions 1 et 2, on effectue au pire L comparaisons (on compare le nombre de successeurs de u et de v appartenant à chacune des classes). Si la propagation est validée, alors le processus est récursif sur les sommets nouvellement aggrégés au motif, mais ces sommets ne seront plus considérés pour la propagation par la suite.

Concernant la version 3 de la méthode, pour une comparaison des sommets u et u', l'algorithme nécessite la comparaison deux à deux des sous-DAGs G_{u_i} et $G'_{u'_j}$ issus des successeurs u_i de u et u'_j de u' respectivement. Soit $\mathcal{N}^+(u)$ (resp. $\mathcal{N}^+(u')$) les successeurs de u (resp. u'). Pour déterminer si deux sommets u et u' sont τ-similaires, on compare au pire $|\mathcal{N}^+(u)||\mathcal{N}^+(u')|$ sous-DAGs. Et ce, récursivement sur les sous-DAGs issus des successeurs de successeurs de u et u'.

Dans la section qui suit, nous présentons quelques applications de la méthode que nous venons de présenter.

2.2 Exemples d'application

L'heuristique de reconnaissance de motifs similaires présentée dans ce chapitre est générique et peut être adaptée à de nombreux domaines d'applications. Dans cette section, nous présentons des applications utilisant l'une des variantes de l'heuristique ayant donné lieu à une publication.

2.2.1 Recherche d'objets similaires dans la vidéo

Nous avons proposé d'adapter la méthode pour la reconnaissance d'objets extraits de flux vidéo à très basse résolution [30]. Ces travaux ont été effectués dans le cadre de la thèse et font l'objet d'un chapitre complet (voir chapitre 4).

FIG. 2.15: Exemple d'une requête d'objet dans une base de données vidéo.

Le principe de l'application est de retourner à l'utilisateur les objets de la base de données qui correspondent le mieux à un objet requête. Un exemple de l'application sur deux requêtes (vignettes détourées de rouge) est montré sur la figure 2.15. Dans cet exemple, les 5 objets les plus similaires sont proposés à l'utilisateur.

Dans cette application, on construit des DAGs à partir des objets de la vidéo et on applique la première variante de l'heuristique pour déterminer les portions communes entre deux objets.

2.2.2 Évolution de code

Plus récemment, nous avons utilisé la troisième variante de l'heuristique pour la visualisation de l'évolution de code C++ au fil des différentes versions d'un projet logiciel [28]. Le principe est de proposer au responsable de projet un outil de visualisation rapide des modifications de code source pour la maintenance du projet afin de repérer les structures stables et les éléments du projet qui ont fait l'objet de modifications. Le code source de chacune des versions est représenté par un arbre syntaxique. Nous illustrons les structures stables entre les versions par des tubes colorés.

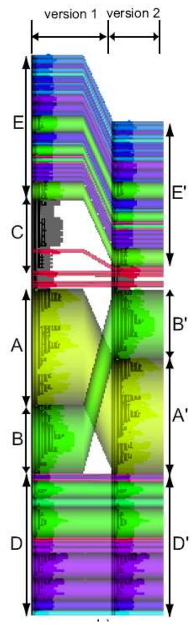


FIG. 2.16: Deux versions d'un fichier source de 850 lignes de code environ.

Le chapitre 5 est consacré au détail de cette application.

La figure 2.16 montre un exemple de visualisation sur deux versions de code source d'environ 850 lignes. On note sur cet exemple une suppression de code (bloc C), une inversion de blocs (blocs A et B) et du code stable (bloc D).

2.2.3 Évolution du trafic aérien

Dans ce troisième exemple d'applications, nous avons testé la deuxième version de l'heuristique sur des graphes non orientés, sans exploiter les informations extrinsèques aux données. Ainsi, la reconnaissance de motifs quasi-similaires n'exploite que l'information structurelle des graphes considérés.

Dans cet exemple, nous avons cherché à mettre en évidence les motifs quasi-similaires entre les réseaux représentant le trafic aérien en 2000 et en 2004 du projet de recherche SPANGEO (Spatial Networks in Geography). Nous avons filtré les réseaux afin de ne conserver que les lignes aériennes ayant brassé plus de 800 000 passagers au cours de l'année.

La figure 2.17 à la fin de ce chapitre montre le résultat de la reconnaissance de motifs par propagation entre les réseaux à l'an 2000 (Fig. 2.17.a) et à l'an 2004 (Fig. 2.17.a).

Sur la figure 2.18, nous avons mis en évidence l'ensemble des sommets ayant été agrégés à un motif quasi-similaire (en vert). Nous pouvons observer sur cette figure que le "coeur" du réseau (les villes américaines) a été reconnu comme motif quasi-similaire. À l'inverse, nous pouvons remarquer que le trafic aérien issu de Londres a évolué entre les années 2000 et 2004. En effet, le nombre de lignes aériennes de plus de 800 000 passagers transitant par Londres a légèrement évolué (41 en 2000 à 47 en 2004) engendrant une évolution du réseau de proximité de la ville de Londres. Nous pouvons notamment mentionner la ville de New-York et les villes desservies ainsi que le réseau San-Jose, Sacramento, Seattle.

Sur la figure 2.19, nous avons isolé le motif quasi-similaire le plus important identifié dans les deux réseaux. Les sommets coloriés en vert sont les sommets représentant des villes communes entre les deux motifs. Dans ce motif, 52 % des sommets sont communs aux deux réseaux.

2.2.4 Évolution du réseau de migration alternantes

Dans cette dernière application, nous avons comparé des graphes du projet SPANGEO représentant les migrations domicile-travail en 1975 et en 1982 dans la région Provence Alpes Côte d'Azur. Les graphes sont orientés et les arêtes sont dirigées de la commune de résidence vers la commune de travail. Nous n'avons conservé que les arêtes du graphe telles que l'effectif dépasse 8 personnes.

La figure 2.20 montre le résultat de la reconnaissance de motifs quasi-similaires entre les années 1975 (Fig. 2.20.a) et 1982 (Fig. 2.20.b).

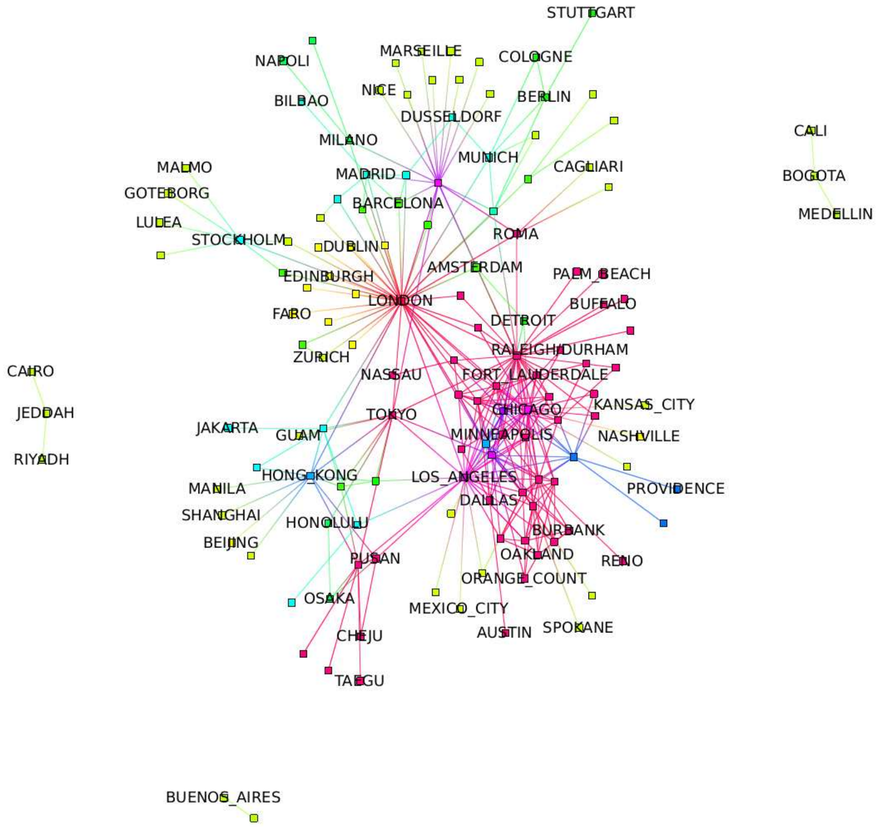
On peut déjà remarquer la présence d'un large motif commun (en violet) entre les deux réseaux. Sur la figure 2.21, les sommets appartenant à un motifs quasi-similaire ont été coloriés en vert. À l'image de la ville de Londres dans les réseaux aériens, le réseau autour de la ville de Nice a évolué entre les années 1975 et 1982. Le nombre de résidents de la ville travaillant à l'extérieur a peu évolué (70 en 1975 et 79 en 1982) alors que le nombre de travailleurs extérieur venant exercer à Nice a fortement augmenté (de 92 à 130).

Sur la figure 2.22, nous avons isolé le motif quasi-similaire le plus important. Dans cet exemple, 50 % des sommets sont communs aux deux motifs, avec notamment des villes pivots comme Le Cannet, Saint-Laurent-du-Var, Carros et Villeneuve-Loubet. Ainsi, les migrations professionnelles sur ce réseau restreint sont restées stables.

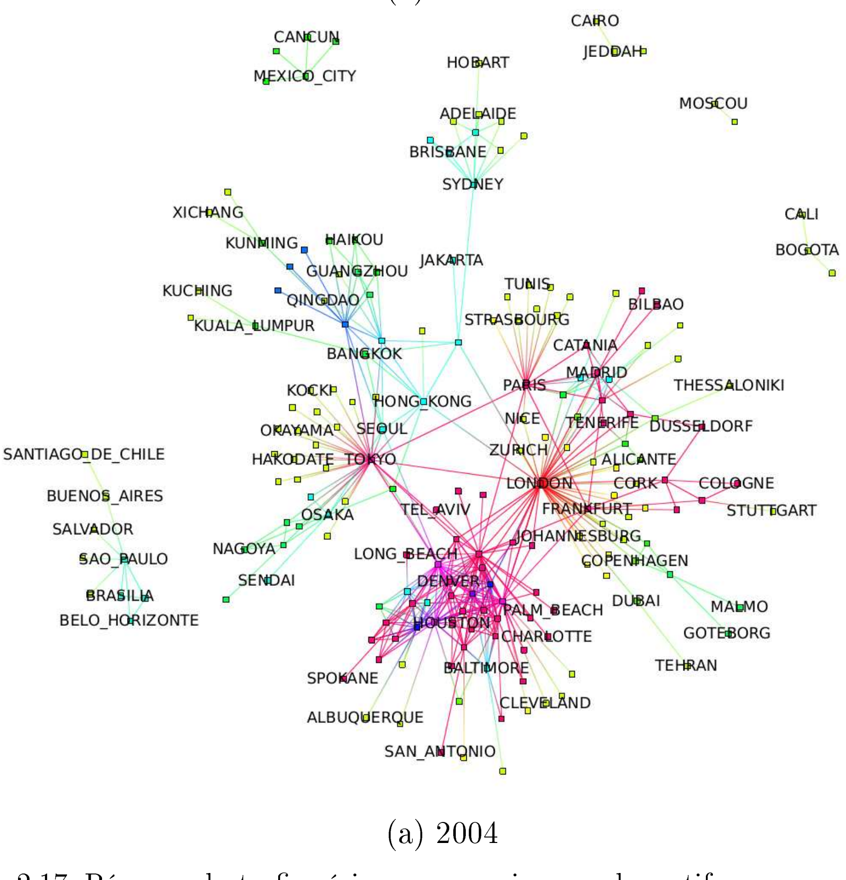
2.3 Conclusion

Nous avons présenté dans ce chapitre une méthode pour la reconnaissance de motifs similaires dans des graphes. En proposant trois variantes correspondant à différentes stratégies de tolérance sur la notion de similarité de structure, nous avons défini une méthode générique qui peut être adaptée à de nombreuses applications, dans divers domaines tels que la biologie, le multimédia, la maintenance logicielle, comme nous l'avons illustré dans la dernière section de ce chapitre.

Les applications ayant donné lieu à des publications présentées dans ce chapitre basées sur l'heuristique de reconnaissance de motifs concernent dans tous les cas des structures orientées sans cycles (DAG ou arbres) dans lesquelles un parcours hiérarchique des sommets peut être effectué. Nous envisageons, en perspective de ces travaux, d'appronfondir la méthode pour des applications nécessitant la comparaison de graphes non orientés tels que les réseaux géographiques, ou les réseaux sociaux.



(a) 2000



(a) 2004

Fig. 2.17: Réseaux du trafic aérien : reconnaissance de motifs par propagation d'étiquette.

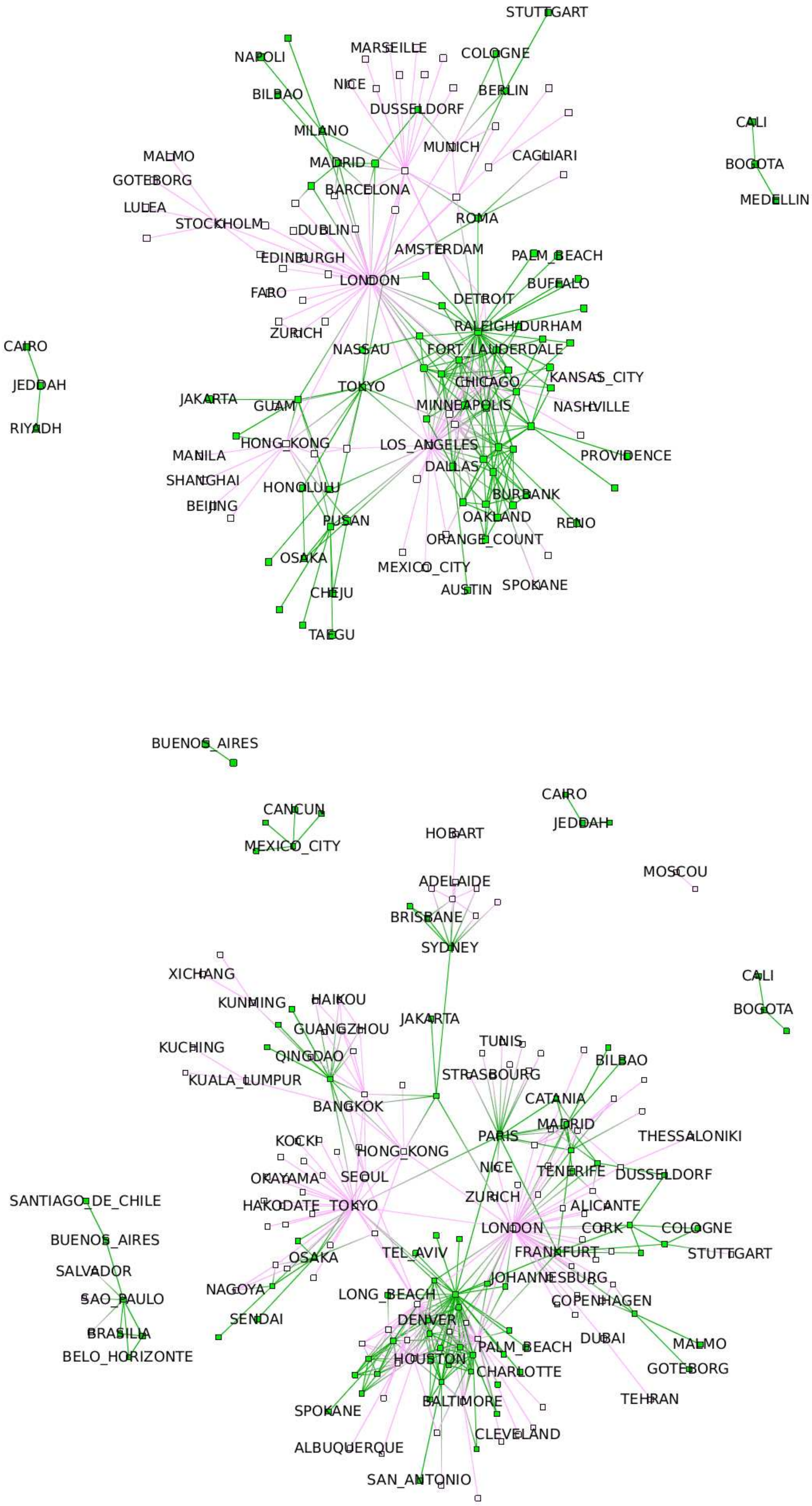


FIG. 2.18: Réseaux de trafic aérien : motifs quasi-similaires identifiés (en vert).

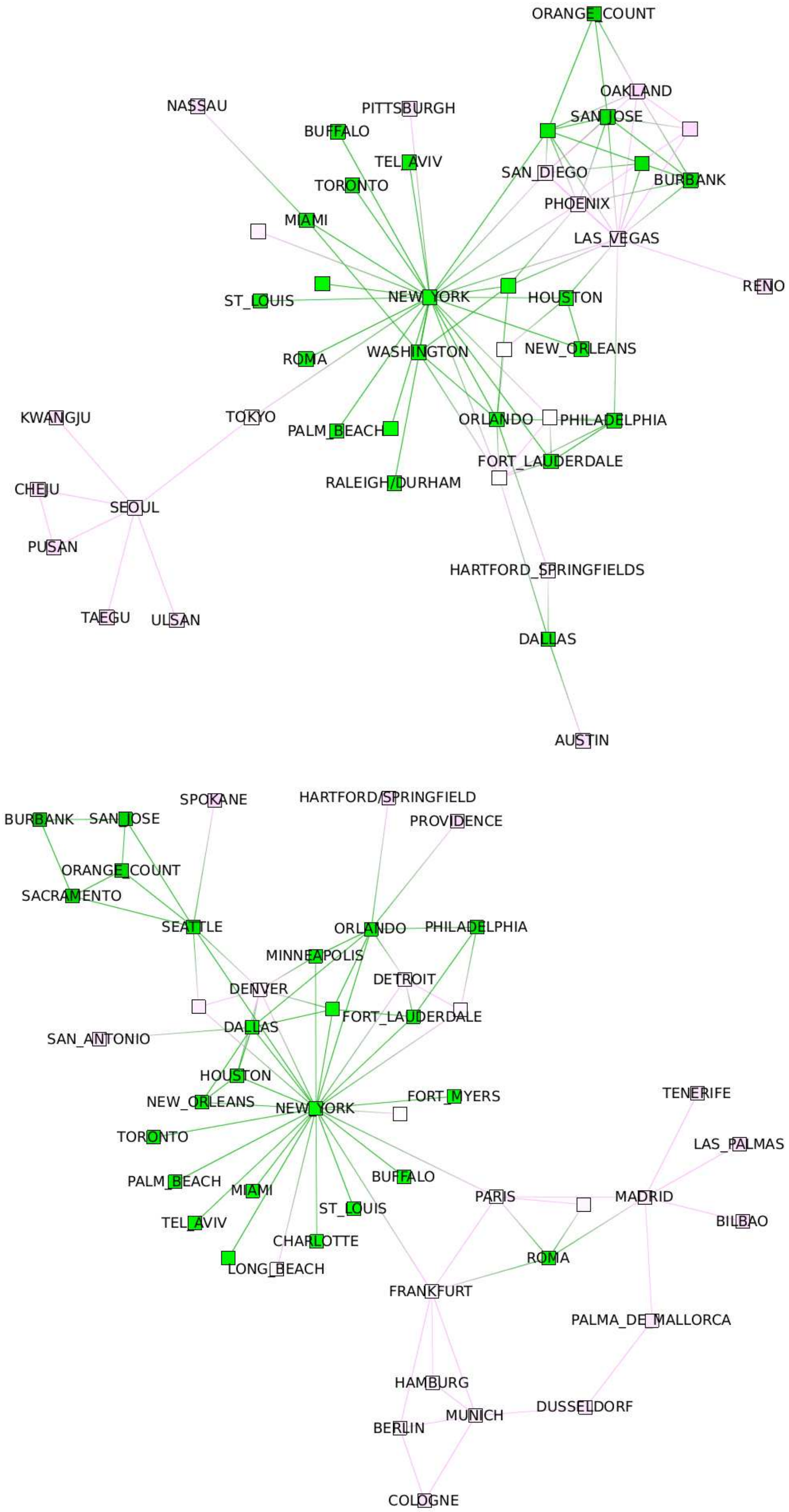
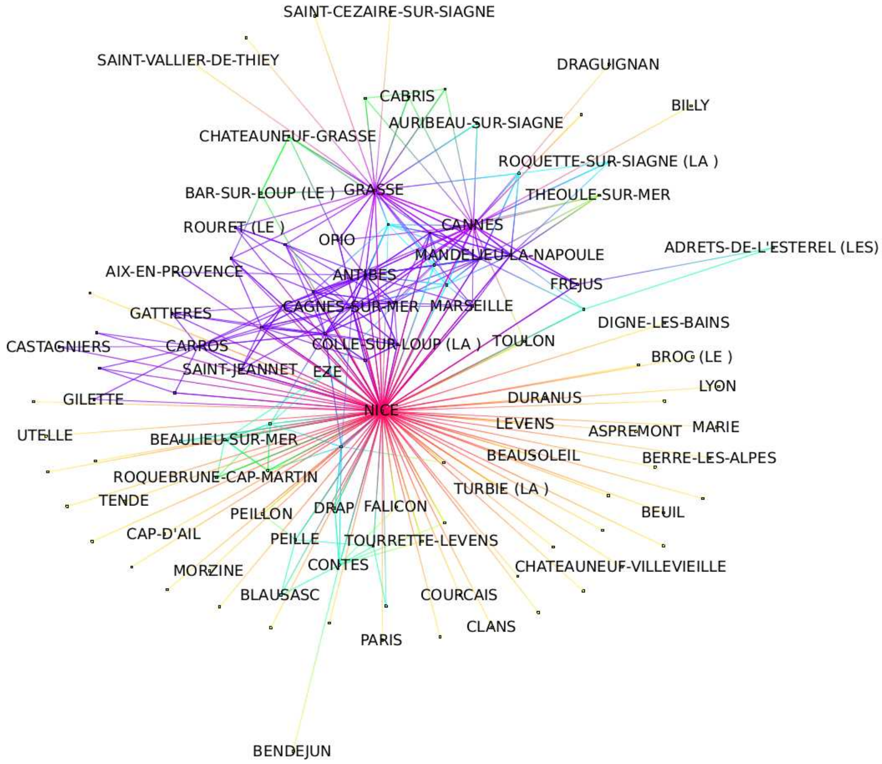
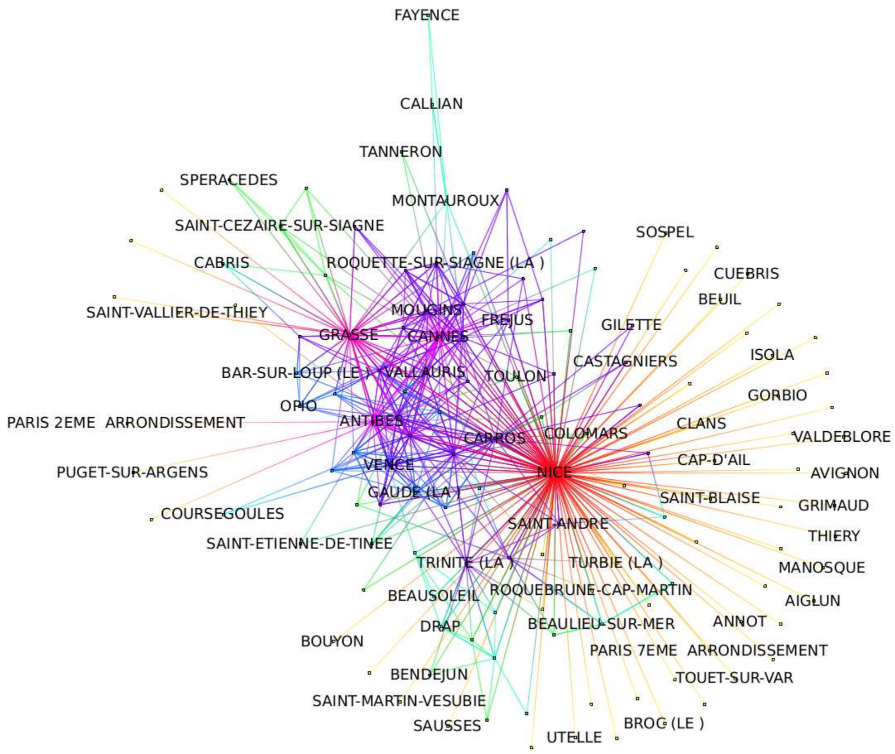


Fig. 2.19: Réseaux de trafic aérien : motif quasi-similaires. Les sommets communs aux deux motifs sont coloriés en vert.

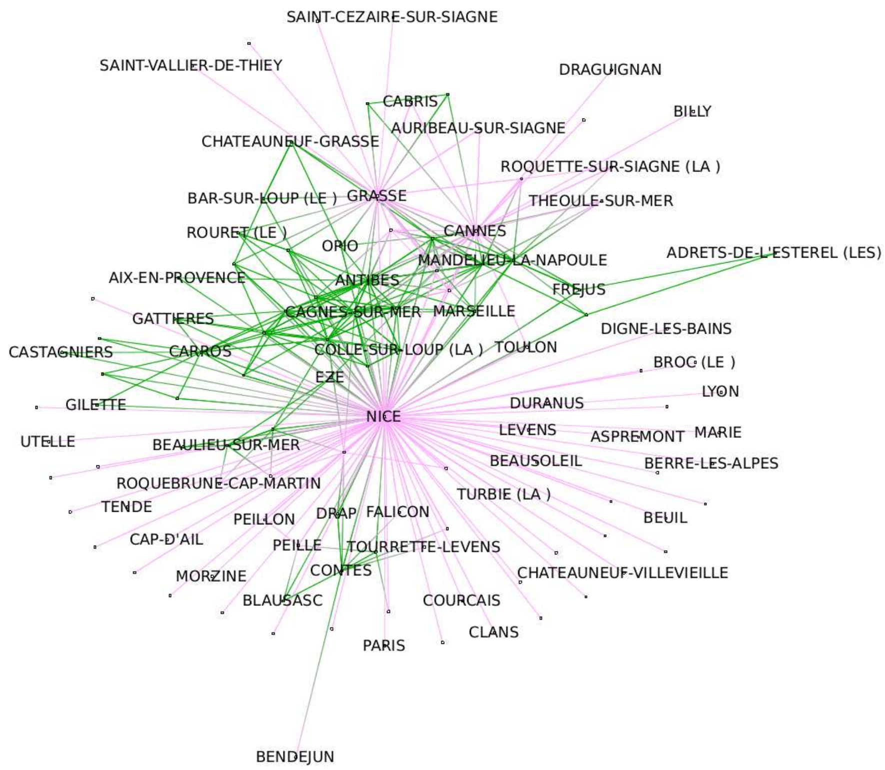


(a) 1975

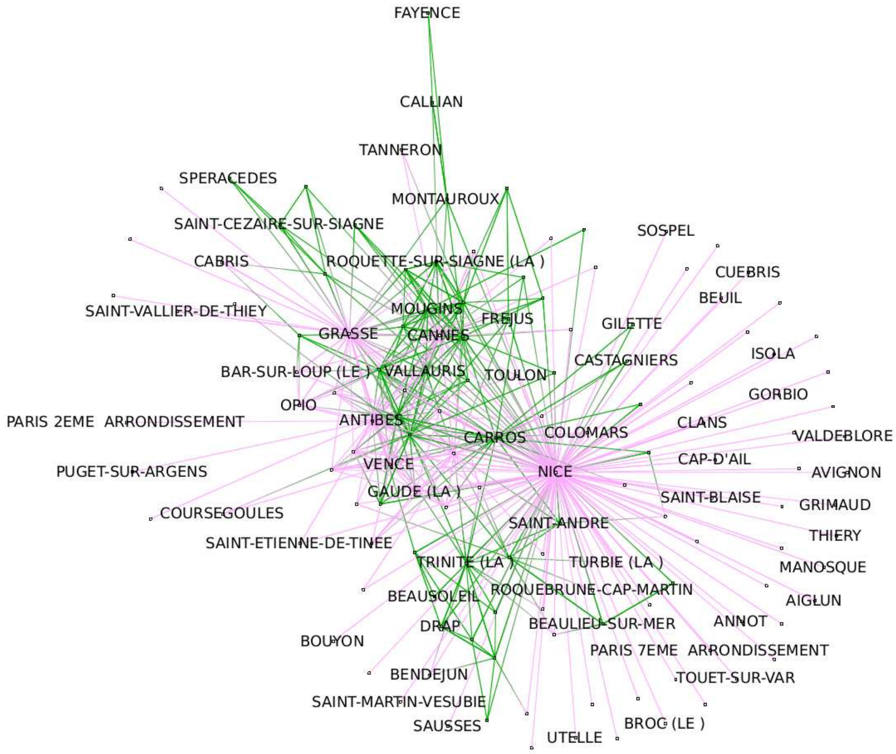


(a) 1982

FIG. 2.20: Réseaux de migrations professionnelles : reconnaissance de motifs par propagation d'étiquette.

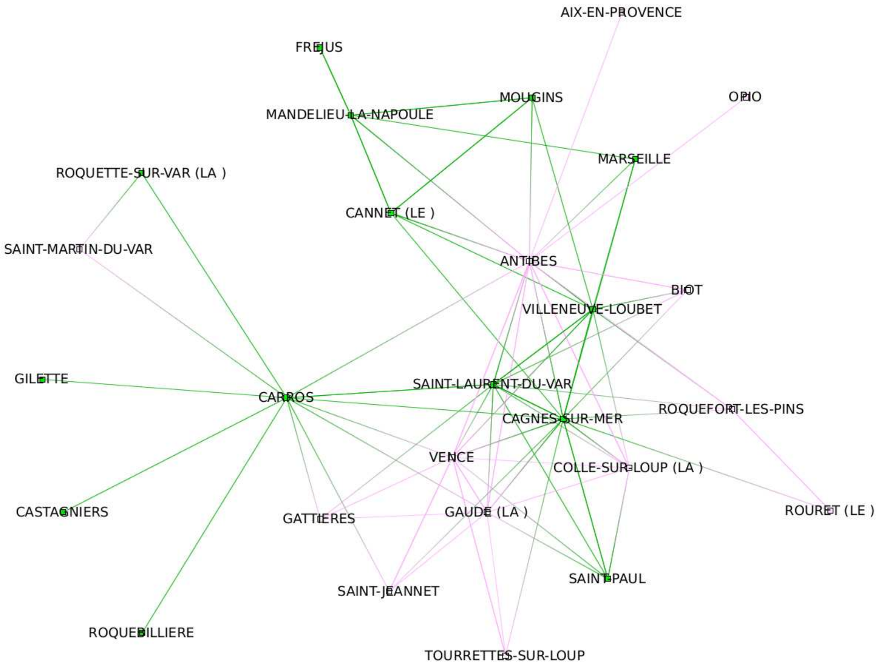


(a) 1975

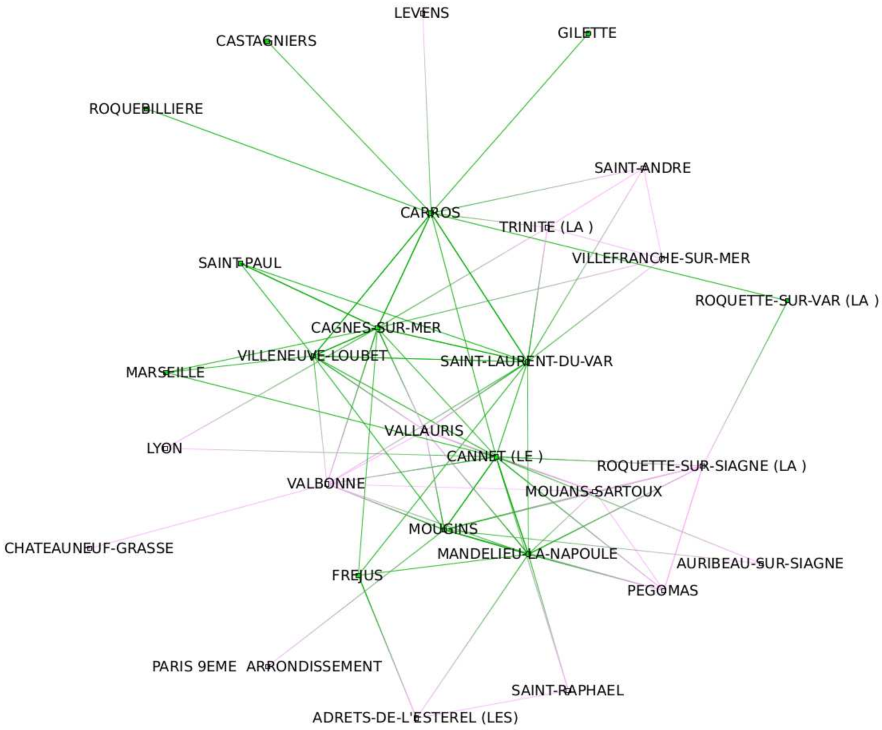


(a) 1982

FIG. 2.21: Réseaux de migrations professionnelles : motifs quasi-similaires identifiés (en vert).



(a) 1975



(a) 1982

Fig. 2.22: Réseaux de migrations professionnelles : un motif quasi-similaire. Les sommets communs aux deux graphes sont représentés en vert.

Chapitre 3

Modèle de génération aléatoire d'arborescence de fichiers

Nous venons de décrire, dans le chapitre précédent, une heuristique permettant de comparer des structures de graphes afin d'en identifier les motifs communs. Cette heuristique a été initialement motivée par l'étude de systèmes de fichiers [7] et testée sur la paire d'arborescences proposée pour le concours InfoVis [52]. Dans le but de valider cette heuristique sur un jeu de données plus large, nous avons commencé par récolter des arborescences de fichiers réelles. L'étude de ces structures a motivé le développement d'un modèle de génération aléatoire basé sur les observations, permettant de produire des arbres proches des systèmes de fichiers.

La génération de graphes est communément utilisée pour l'étude de la validité d'algorithmes approximant la solution optimale. Si des graphes aléatoires peuvent être générés, ils constituent une possibilité pour tester les algorithmes cherchant à résoudre un problème difficile dont on ne peut pas pour l'instant calculer la solution optimale. En utilisant de tels graphes, les simulations peuvent être utilisées pour déterminer, en moyenne ou au pire, les performances d'un algorithme.

Dans ce chapitre, nous définissons un modèle aléatoire permettant de générer des arbres proches des arborescences de fichiers utilisateurs. Le chapitre est structuré comme suit : dans une première section, nous présentons les résultats de l'étude statistique des données réelles qui a permis d'établir le modèle de génération. La deuxième section présente le nouveau modèle de génération d'arborescence de fichiers.

3.1 Étude statistique

Dans cette section, nous présentons les résultats de l'étude statistique du jeu de données réelles étudié dans le cadre de cette thèse. La base comprend un ensemble de structures arborescentes correspondant à des systèmes de fichiers utilisateurs. Après avoir décrit en détails la nature des données que nous avons utilisées pour ce travail, nous établissons une liste de propriétés observées sur ces structures d'arbres particuliers.

3.1.1 Description du jeu de données

Le jeu de données étudié dans le cadre de ces travaux est le résultat de l'aspiration des structures arborescentes constituant les systèmes de fichiers utilisateurs du Laboratoire Bordelais de Recherche en Informatique (LaBRI). Chacun des arbres obtenus représente une capture de l'arborescence de fichiers d'un compte utilisateur (Unix ou Windows^{TM}) à un instant donné.

La base d'étude comprend les arbres de 83 utilisateurs, capturés à environ deux mois d'intervalle. À noter que certains comptes utilisateur (2 comptes) ont été créés après la première capture, et qu'aucune suppression de compte n'a été effectuée entre les deux acquisitions de structures arborescentes. Toutefois, le caractère dynamique des données dont nous disposons n'a pas encore fait l'objet d'une étude approfondie et constitue l'une des perspective de ce travail de thèse. Nous avons dans un premier temps exploré les caractéristiques structurelles des arbres sans chercher à exploiter les relations existantes entre les arbres de la première capture et ceux de la seconde acquisition.

Le plus petit des arbres de la base de données contient 16 sommets, alors que l'arbre le plus volumineux de la base est constitué de près de 170 000 sommets. L'étude de données statistiques nécessite d'observer les propriétés sur des arbres comparables en terme de taille. En effet, le choix d'une normalisation des mesures n'est pas trivial et risquerait de biaiser l'interprétation des données étudiées. Ainsi, nous avons choisi d'explorer le jeu de données en considérant les arbres séparément selon des fourchettes de tailles définies empiriquement.

La table 3.1 donne le découpage utilisé ainsi que le nombre d'arbres contenus dans chaque tranche pour chacune des captures d'arborescences.

Nombre de sommets n	Effectif capture 1	Effectif capture 2	Effectif total
$0 \leq n < 500$	12	13	25
$500 \leq n < 1000$	4	8	12
$1000 \leq n < 2000$	4	3	7
$2000 \leq n < 5000$	8	9	17
$5000 \leq n < 10000$	10	10	20
$10000 \leq n < 15000$	11	7	18
$15000 \leq n < 20000$	6	9	15
$20000 \leq n < 30000$	6	6	12
$30000 \leq n < 50000$	9	8	17
$50000 \leq n < 90000$	5	2	7
$90000 \leq n < 160000$	4	8	12

TAB. 3.1: Découpage de la base par fourchette de taille.

Dans ce manuscrit, nous considérons que les arbres représentant les systèmes de fichiers utilisateurs sont des arbres orientés. Les arcs sont orientés des répertoires vers les éléments qu'il contiennent (fichiers, ou sous-répertoires).

3.1.2 La loi en puissance

Comme nous l'avons vu dans le chapitre d'introduction, il est fréquent d'observer une distribution des degrés qui suit une loi en puissance dans les réseaux réels. Il parait moins évident d'observer une telle propriété lorsque les graphes considérés sont des arbres.

Dans le cas des arborescences de fichiers, les feuilles représentent dans la très grande majorité des cas les fichiers (parfois ce sont des répertoires vides), les sommets internes correspondent toujours à des répertoires.

La figure 3.1 illustre la distribution des degrés des arbres de la base de données pour différentes fourchettes de tailles.

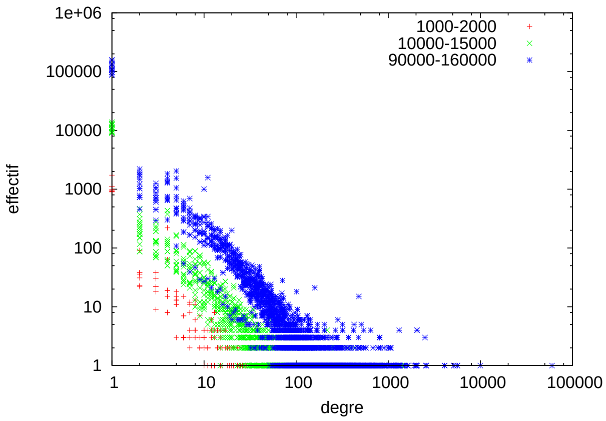


FIG. 3.1: Distribution des degrés sur le jeu de données pour différentes tranches d'arbres.

Nous retrouvons une distribution des degrés qui suit une loi en puissance. Ce phénomène correspond en quelque sorte au "taux d'attractabilité" de chaque répertoire. En effet, dans le système de fichiers d'un utilisateur dit "moyen", on trouve plusieurs centaines de répertoires. Dans la réalité, les modifications apportées à ces répertoires sont liées à leur contenu. Si l'on prend l'exemple du doctorant en phase de rédaction, le répertoire qui contient les fichiers correspondants au manuscrit de thèse va connaître une forte activité durant la phase de rédaction. Une fois la thèse soutenue, ce répertoire ne subira certainement plus que de très rares modifications. Si l'on prend le cas des répertoires qui contiennent beaucoup d'éléments, on constate que, le plus souvent, ces répertoires correspondent aux répertoires d'images, de fichiers musicaux ou bien des répertoires contenant des jeux de tests. Ces répertoires sont souvent peu nombreux et très volumineux : les données qu'ils contiennent ne peuvent pas être réparties en plusieurs sous-catégories, ou leur tri est tellement long et fastidieux que l'on se contente de laisser tous ces éléments groupés dans un seul répertoire. À l'inverse, il existe un grand nombre de répertoires qui ne contiennent que très peu d'éléments : parfois un répertoire a été créé il y a si longtemps que l'on ne se souvient plus de sa localisation ou même de son existence et l'on créé un second répertoire contenant le même type de données.

Mais il semble que la raison principale à cette tendance à multiplier les répertoires de petite taille au détriment des dossiers volumineux vient de notre capacité à ordonner les éléments pour optimiser la recherche d'un document précis. Notre aptitude naturelle à organiser les données en sous-catégories rappelle la stratégie "diviser pour régner" qui consiste à diviser un problème de grande taille en plusieurs sous-problèmes. Le problème ici est la recherche d'un document parmi l'ensemble des fichiers stockés. À moins de connaître le nom du fichier en question, le temps de recherche dans une très grande collection non organisée est proportionnel au nombre d'éléments. Lee et MacGregor [79] ont montré que le nombre optimal d'entrées dans un menu pour la navigation efficace est compris entre quatre et seize : le temps de recherche est plus rapide dans une structure hiérarchique dont le nombre d'éléments est réduit à chaque étape de décision. Ainsi, dès lors que l'on trouve le temps de décision trop long, on décide de classer, c'est-à-dire de subdiviser les données.

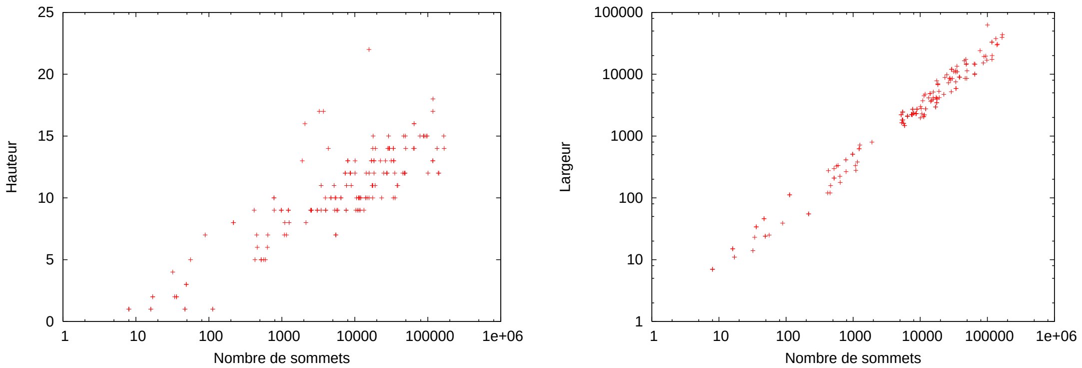
La première propriété que nous pouvons établir à partir des observations en pratique sur les arborescences de fichiers est la suivante :

Propriété 3.1 (Arbre sans échelle)
Les structures arborescentes vérifient la propriété d'arbre sans échelle (Déf. 32).

3.1.3 Forme des arborescences de fichiers

Nous avons cité en introduction les travaux de Viennot *et al.* [131] sur l'utilité de la caractérisation de la forme des plantes et arbres biologiques. Pour la génération d'arbres binaires aléatoires, Viennot fait remarquer que le modèle qui consiste à affecter à chaque feuille aléatoirement un, deux ou aucun sommet produit un arbre plutôt filiforme. Il est donc indispensable d'apporter une information supplémentaire quant à la probabilité que telle ou telle situation se produise afin d'obtenir des arbres aux caractéristiques différentes.

Les arbres correspondant aux systèmes de fichiers ont la particularité d'avoir une hauteur et une largeur qui dépendent fortement du nombre de sommets qu'ils contiennent. La figure 3.2.a montre la hauteur de chaque arbre en fonction de sa taille, et la figure 3.2.b montre la largeur en fonction du nombre d'éléments contenus dans chaque arbre.



(a) (b)

FIG. 3.2: (a) Hauteur et (b) largeur des arbres en fonction de leur taille.

On remarque que la hauteur de l'arbre croît logarithmiquement avec son nombre d'éléments. Dans une arborescence de fichiers, plus il y a de documents à stocker, plus leur hiérarchisation est nécessaire afin d'assurer une navigation efficace. Ainsi, dès lors qu'un répertoire commence à être "encombré", on va essayer de le sub-diviser en sous-catégories afin de faciliter la navigation, comme on l'a mentionné dans le paragraphe précédent. Toutefois, la hiérarchisation d'un répertoire sur de très nombreux niveaux est peu fréquente. En effet, le chemin absolu d'un document ne dépasse pas les 22 répertoires dans la base de données que nous avons étudiée (pour un arbre contenant quelques 20 000 éléments). En règle générale, un utilisateur cherche à concilier une prise de décision rapide au sein d'un répertoire (hiérarchisation thématique) et la minimisation du nombre de répertoires à parcourir pour atteindre un document (accès rapide). Par conséquent, il est rare qu'une arborescence de fichiers respecte une propriété de séparabilité sémantique au sein d'un même niveau de la hiérarchie. Il est possible que l'un des répertoires, sous-thème d'un autre répertoire du même niveau, ait été positionné volontairement plus haut dans la hiérarchie pour faciliter son accès car il est plus récent, ou plus sollicité que le répertoire qui aurait dû le contenir. On admet alors la propriété empirique :

Propriété 3.2 (Hauteur de l'arbre)
La hauteur h d'une arborescence de fichiers est logarithmiquement proportionnelle au nombre d'éléments n contenus dans l'arbre. On a la relation suivante :

$$h = a_h \log(n) + b_h$$

Où a_h et b_h sont déterminés empiriquement.

La propriété sur la largeur de l'arbre est une conséquence directe aux observations que nous venons d'établir. Les éléments de l'arborescence ont tendance à être concentrés sur un niveau intermédiaire de l'arbre. C'est le niveau qui concilie au mieux l'accès rapide aux données (relativement à la taille de l'arbre) et la recherche efficace au sein d'un répertoire.

Propriété 3.3 (Largeur de l'arbre)
Le logarithme de la largeur l d'un arbre représentant un système de fichiers est fonction du nombre d'éléments n de l'arbre. On a la relation :

$$\log(l) = a_l \log(n) + b_l$$

Où a_l et b_l sont déterminés empiriquement.

La figure 3.3 représente pour une hauteur h, la distribution moyenne des éléments dans les niveaux des arbres. Les distributions observées ont des allures gaussiennes. Le "pic" de chaque courbe correspond au niveau dans l'arbre où les éléments sont le plus nombreux, niveau de l'arbre le plus large.

Nous admettons la propriété 3.4 pour les structures arborescentes.

Propriété 3.4 (Distribution des éléments)
La répartition des éléments dans les niveaux de l'arborescence de fichiers suit une loi de Gauss.

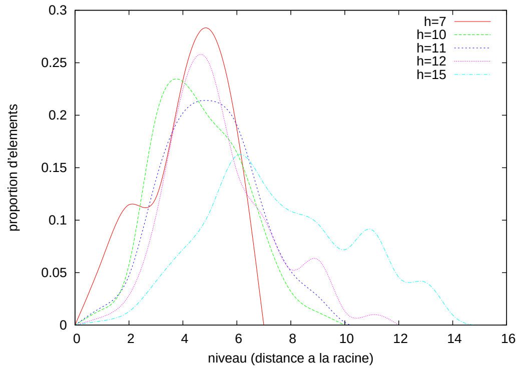
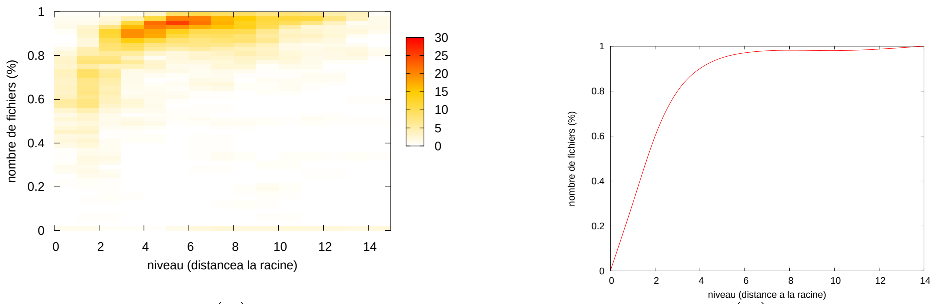


FIG. 3.3: Distribution des éléments selon le niveau dans l'arborescence de fichiers.

3.1.4 Composition des niveaux

Après avoir étudié les caractéristiques structurelles des systèmes de fichiers arborescents, nous nous sommes intéressés à leur contenu. Dans une arborescence de fichiers, nous distinguons deux types d'éléments : les fichiers et les dossiers. Nous nous sommes interrogés sur la répartition de ces éléments au sein des niveaux dans l'arbre afin d'identifier une éventuelle stratégie commune de répartition des éléments selon leur type. La figure 3.4.a suggère qu'une telle propriété existe. Sur cette figure sont représentées, pour chaque niveau dans l'arbre, les proportions de fichiers par rapport au nombre d'éléments total dans le niveau correspondant. La figure montre le résultat de la construction d'une carte d'élévation pour l'ensemble des arbres de la base de données. La carte d'élévation traduit la fréquence d'apparition (modélisée par la couleur) d'un point de coordonnées (x, y) pour l'ensemble des courbes associées aux arbres de la base de données. Par exemple, il y a ici 5 arbres dans la base tels que la proportion de fichiers au niveau 2 est de 40%.



(a) (b)

FIG. 3.4: Proportion de fichiers par niveaux dans l'arbre (a) sur la base de données, (b) relation empirique

Nous observons qu'il se dessine une tendance générale de répartition des éléments selon le niveau dans l'arbre. Nous avons modélisé cette tendance par la courbe de la figure 3.4.b. On remarque que la proportion des fichiers croît au fur et à mesure que l'on descend dans la hiérarchie arborescente. En effet, les niveaux proches de la racine correspondent aux niveaux dans lesquels se trouvent les répertoires correspondant aux différents grands thèmes des données qui sont classées plus bas dans la hiérarchie. Par conséquent, il est logique de trouver les répertoires en surnombre dans ces niveaux, permettant une hiérarchisation efficace des documents. Plus on descend dans la hiérarchie, plus les répertoires se font rares : il n'est plus nécessaire de sub-diviser les données d'un même répertoire en plusieurs sous-répertoires car le nombre de documents est raisonnable pour une navigation efficace, ou alors il n'est plus pertinent de créer des sous-thèmes. Ainsi, plus on s'éloigne de la racine, plus les fichiers sont prédominants dans le niveau de la hiérarchie.

Nous admettons la propriété suivante pour les arborescences de fichiers :

Propriété 3.5 (Composition des niveaux)
Le type d'éléments contenus dans un niveau de l'arbre dépend du niveau de l'arbre.

La relation de composition des niveaux, définie empiriquement, est montrée sur la figure 3.4.

3.1.5 Ramification des arbres et catégories d'utilisateurs

Jusqu'à présent, nous avons déterminé certaines propriétés empiriques qui concernent l'intégralité des arborescences de fichiers. Outre les propriétés partagées par la totalité des arborescences appartenant à cette catégorie particulière d'arbres, nous avons cherché à caractériser des *familles d'utilisateurs*, à l'image des travaux de Viennot *et al.*, qui ont soulevé l'importance de la propriété de ramification pour comprendre et ensuite modéliser les différents types de plantes [131]. Parce que nous n'avons pas tous le même comportement quant à l'organisation et la classification des documents électroniques, nous pouvons soupçonner l'existence de catégories d'utilisateurs selon le comportement plus ou moins organisé de ces derniers. De l'utilisateur très désordonné à l'utilisateur extrêmement méthodique, la structure arborescente devrait différer en certains points et être commune pour des utilisateurs appartenant à une même catégorie. Dans cette optique, nous proposons de caractériser ce que l'on appellera dans ce manuscrit une *famille d'utilisateurs* en déterminant les différentes stratégies de structuration de l'arborescence de fichiers.

Pour comprendre la stratégie de hiérarchisation des données de différents individus, l'idéal aurait été d'avoir accès au contenu sémantique des documents électroniques qui constituent l'arborescence de fichiers. Cependant, dans un soucis du respect de la vie privée des personnes, seule la structure arborescente des systèmes de fichiers a été capturée. Les noms des fichiers et répertoires sont alors indisponibles dans la base de données d'étude et il nous est difficile de tirer des conclusions quant au comportement des utilisateurs. En effet, certaines parties de l'arborescence peuvent correspondre à des répertoires d'installation de logiciels dont la structure est totalement indépendante de la stratégie de hiérarchisation des données de l'utilisateur. D'autre part, comme on l'a mentionné plus haut, on ne classera certainement pas nos images de la même façons que nos courriers administratifs, et le fait d'ignorer la sémantique associée à chaque élément nous empêche d'établir les lois de structuration en fonction du type d'éléments, s'il existe de telles propriétés.

3.1.5.1 Détermination des familles d'utilisateurs : étude des *arbres-squelettes*

Comme dans un tel contexte il nous est très difficile d'identifier les facteurs déterminant le choix de structuration des données, nous nous sommes basés sur la seule information disponible pour différencier les familles d'utilisateurs, à savoir les propriétés structurelles des arbres. Dans leurs travaux, Viennot *et al.* [131] se basent sur des mesures combinatoires telles que le nombre de Strahler (Déf. 29, Sec. 1.1) dans le but de caractériser la forme de l'arborescence étudiée. Ainsi, le taux de ramification d'un arbre peut être évalué grâce au nombre de Strahler à sa racine.

Dans notre cas, nous avons vu qu'il existe fréquemment, dans une arborescence de fichiers, un ou plusieurs répertoires (de tests, d'images, ...) de très haut degré. Le nombre de Strahler étant très fortement corrélé au degré des sommets dans le sous-arbre, celui-ci perd tout son sens en la présence de tels répertoires : l'indice de ramification de l'arbre (*i.e.* le nombre de Strahler à la racine) correspond au degré du sommet le plus ramifié à savoir, le sommet de plus fort degré dans l'arbre. Afin d'étudier le squelette structurel des arborescences de fichiers, nous avons cherché à supprimer ces *répertoires volumineux* (Déf. 43) (à la manière de l'élagage de Zeilberger [145]) qui "parasitent" l'évaluation de la structure sous-jacente. La technique d'élagage est la suivante (on rappelle que $\sigma(v)$ correspond au nombre de Strahler du sommet v (Déf. 29, Sec. 1.1), et $d(v)$ correspond au degré de v) :

Données: Un arbre $T = (V, E)$
Retour: L'arbre élagué $T_s = (V_s, E_s)$

$T_s := T$
$\bot_s :=$ racine de l'arbre T_s

$\sigma_s := \sigma(\bot_s)$
$R := \{v \in V_s \mid d(v) = \max_{u \in V_s} d(u)\}$

TantQue $\exists r \in R \mid r \neq \bot_s$ Et $\sigma_s = d(r)$ **Faire**
    $T_r :=$ sous$-$arbre de T_s enraciné en r
    $T_s := T_s \setminus T_r$
    $\sigma_s := \sigma(\bot_s)$
    $R := \{v \in V_s \mid d(v) = \max_{u \in V_s} d(u)\}$
FinTantQue
Retourne T_s

Algorithme 3.1: Fonction `elagage`

Ainsi, tant qu'il existe un répertoire dans l'arbre dont le degré est égal au nombre de Strahler à la racine de l'arbre (appelé *répertoire volumineux*, Déf. 43) alors on supprime ce répertoire et son contenu (s'il est différent de la racine). On obtient alors un arbre dont les répertoires de plus fort degré ont été supprimés, et il en résulte un arbre élagué qui représente en quelque sorte l'arbre *squelette*, la *charpente* de la structure arborescente initiale.

Définition 42 (Arbre-squelette)
Soit T un arbre. On appelle arbre-squelette de T, l'arbre T_s, résultat de l'élagage de T par l'algorithme 3.1.

Définition 43 (Répertoire volumineux)
Soit T une arborescence de fichiers de racine $\perp$. On appelle répertoire volumineux de T, un répertoire u tel que le degré sortant $d^+(u)$ de u est égal au nombre de Strahler $\sigma(\perp)$ de la racine de l'arbre T.

Ce sont ces *arbres-squelette* qui nous intéressent pour évaluer la stratégie de ramification des structures arborescentes étudiées.

Dans le but de définir les différentes stratégies de structuration des documents électroniques, nous avons étudié de nombreuses métriques (distribution des degrés, hauteur, largeur, composition des niveaux, ...) calculées sur les arbres-squelette de la base de données. Nous avons cherché à identifier des tendances communes sur ces mesures pour certains utilisateurs et définir ainsi des catégories comportementales. Cependant, la plupart des mesures ne laissent apparaître aucune tendance, ou bien sont similaires pour la quasi-totalité des utilisateurs. Seul le nombre de Strahler à la racine des arbres-squelette peut traduire les différentes tendances de structuration des arbres. On note que cette même caractéristique avait été retenue par Viennot *et al.* [131] pour caractériser les familles de plantes. La figure 3.5.a montre la répartition des valeurs de Strahler à la racine pour l'ensemble des arbres-squelettes.

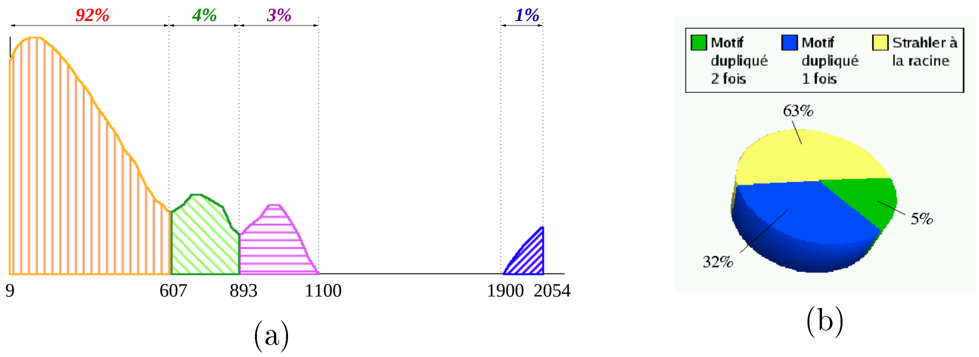


FIG. 3.5: (a) Répartition des valeurs et (b) type du nombre de Strahler à la racine des arbres élagués.

On identifie, à partir de la courbe de la figure 3.5.a, quatre tendances qui sont représentées, sur cette même figure, par les différentes zones hachurées. Les valeurs en abscisse correspondent aux nombres de Strahler qui bornent chaque zone. La proportion d'utilisateurs de la base de données appartenant à chaque tendance est exprimée au dessus de la zone correspondante. On remarque que la grande majorité des arborescences observées se situent dans la première catégorie, qui regroupe les structures de plus faible Strahler.

Définition 44 (Profil d'utilisateurs)
Nous appelerons profil d'utilisateur *l'indice de ramification (i.e. le nombre de Strahler à la racine) de l'arbre-squelette, permettant de regrouper les utilisateurs en familles d'utilisateurs.*

Nous définissons alors la notion de *famille d'utilisateur* comme suit :

Définition 45 (Familles d'utilisateurs)
Les utilisateurs sont répartis en k familles. À chaque famille F_i, $i \in [1, \cdots, k]$, correspond un intervalle $[\sigma_{min}^{(F_i)}, \sigma_{max}^{(F_i)}]$ (Fig. 3.5.a).
Un arbre T appartient à une famille d'utilisateurs F_i si le nombre de Strahler à la racine de l'arbre-squelette T_s correspondant (i.e. son profil utilisateur) est compris dans l'intervalle $[\sigma_{min}^{(F_i)}, \sigma_{max}^{(F_i)}]$ de la famille F.

Dans notre cas, on a $k = 4$: les utilisateurs sont répartis en quatre familles distinctes.

Comme le suggère la figure 3.5.b, dans la plupart des cas, le nombre de Strahler de l'arbre-squelette correspond au degré à la racine. Cela signifie que les arbres que nous étudions contiennent énormément d'éléments à la racine et quelques répertoires très denses ailleurs dans la hiérarchie. Dans les autres cas, le nombre de Strahler correspond à la présence de plusieurs sous-répertoires de même degré dans un même répertoire due à une possible duplication de motif arborescent.

Nous avons également observé une corrélation entre le nombre de Strahler à la racine d'un arbre-squelette et la taille de cette arborescence. Sur la figure 3.6 est affichée la taille de l'arbre-squelette en fonction du nombre de Strahler à la racine.

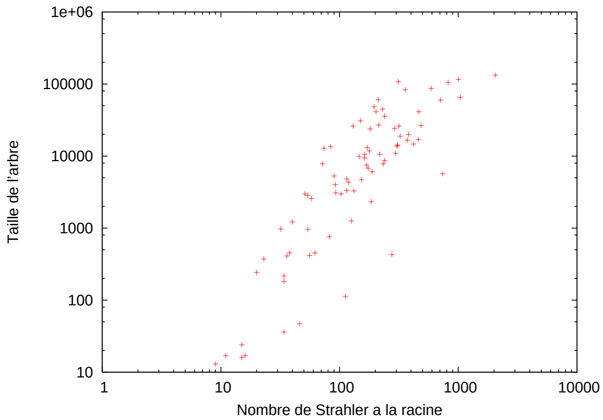


Fig. 3.6: Taille de l'arbre-squelette en fonction du nombre de Strahler à la racine.

Nous observons une relation log-log entre le nombre de Strahler à la racine de l'arbre-squelette et le nombre d'éléments de celui-ci. Nous admettons alors la propriété suivante :

Propriété 3.6 (Taille de l'arbre-squelette)
Le nombre d'éléments n de l'arbre-squelette est lié au nombre de Strahler σ à la racine. On a la relation :

$$\log(n) = a \log(\sigma) + b$$

où a et b sont déterminés empiriquement.

Outre la proposition 3.6, spécifique aux arbres-squelette, l'ensemble des propriétés structurelles que nous avons mentionnées précédemment sur les arborescences initiales restent vérifiées pour les arbres élagués. Ainsi, nous conservons après élagage une distribution des degrés qui suit une loi en puissance (propriété 3.1), une hauteur et une largeur en très forte corrélation avec le nombre d'éléments contenus dans l'arbre (propriétés 3.2 et 3.3). La répartition des éléments dans les niveaux de l'arbre suit une loi de Gauss (propriété 3.4) et enfin, la composition des niveaux reste stable (propriété 3.5). Les résultats de l'étude sur les arbres-squelette sont montrés en Annexe A.

Nous pouvons à présent nous intéresser aux propriétés des répertoires volumineux en sus de ces structures élaguées.

3.1.5.2 Propriétés des *répertoires volumineux*

Maintenant que nous connaissons la structure des arborescences dont les *répertoires volumineux* ont été ôtés, il nous faut observer les caractéristiques de ces répertoires denses.

Soit T un arbre contenant un répertoire volumineux r et soit T_r le sous-arbre de T enraciné en r. Nous avons observé que le degré du *répertoire volumineux* r de l'arbre T est lié au degré du répertoire r' de plus fort degré de l'arbre $T \setminus T_r$. À chaque suppression d'un répertoire volumineux r de la phase d'élagage de l'arbre T, nous avons relevé

- le degré $d(r)$ de r,
- le degré $d(r')$ du répertoire r' de plus fort degré restant dans l'arbre $T \setminus T_r$.

Les répertoires r et r' vérifient

$$\begin{aligned} d(r) \geq d(r') \text{ et,} \\ \forall r'', d(r') \geq d(r'') \end{aligned} \tag{3.1}$$

Sur la figure 3.7, le rapport entre la taille du répertoire volumineux r et la taille du second plus gros répertoire r' est représenté en abscisse. Le graphique représente donc la distribution des répertoires volumineux en fonction de leur taille par rapport au second répertoire de plus fort degré.

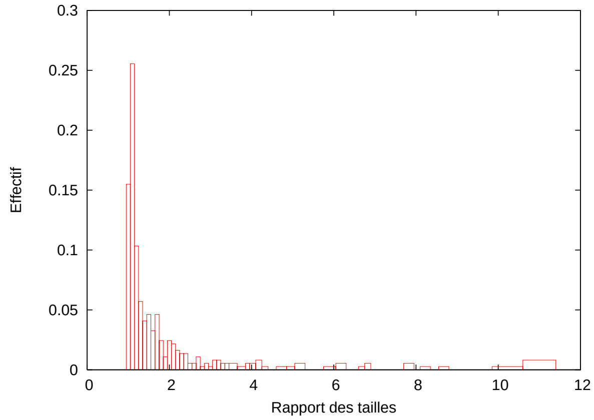


FIG. 3.7: Taille des *répertoires volumineux* en fonction du second répertoire de plus fort degré.

Au vu de la distribution de la figure 3.7, nous admettons la propriété :

Propriété 3.7 (Taille des répertoires volumineux)
Soit r un répertoire volumineux, et soit r' le second répertoire de plus fort degré (Form. (3.1)).
Le degré $d(r)$ de r dépend du degré $d(r')$ de r'
La distribution des rapports $\frac{d(r)}{d(r')}$ suit une loi en puissance.

Enfin, nous avons étudié les différents emplacements de ces répertoires au sein des arborescences. Ainsi, pour chaque répertoire volumineux, nous avons relevé sa profondeur dans l'arbre (*i.e.* distance à la racine, Déf. 25). La figure 3.8 montre la répartition des répertoires volumineux selon leur profondeur dans l'arbre.

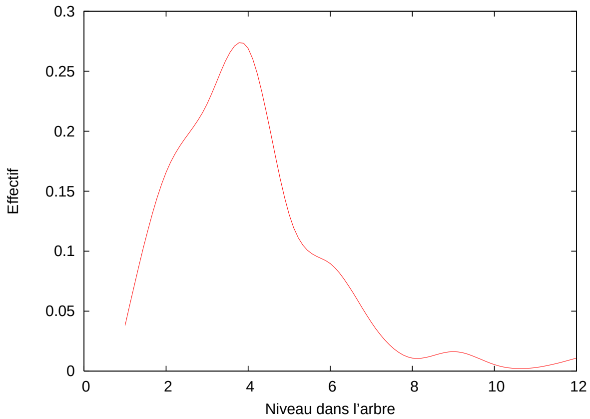


FIG. 3.8: Distribution des profondeurs des *répertoires volumineux*.

Nous observons ici une distribution qui approche une loi de Gauss avec une concentration des répertoires volumineux autour du quatrième niveau de l'arborescence.

De ces observations, nous admettons la propriété 3.8 qui suit.

Propriété 3.8 (Profondeur des répertoires volumineux)
La distribution des profondeurs des répertoires volumineux suit une loi de Gauss.

Nous venons d'établir dans cette section un certain nombre de propriétés empiriques sur les arborescences de fichiers utilisateurs à partir des observations statistiques. Nous proposons maintenant un modèle de génération aléatoire qui vise à reproduire ces propriétés remarquables.

3.2 Proposition de modèles de génération

Dans cette section, nous présentons le modèle stochastique que nous proposons pour la génération aléatoire de systèmes de fichiers utilisateurs. Nous avons vu, dans la section précédente que la caractérisation des familles d'utilisateurs n'est pas triviale. Ainsi, dans une première partie, nous proposons de définir le modèle dans sa version de base (on ne considère pas le comportement de l'utilisateur ni sa stratégie de structuration des données). Nous présentons ensuite l'extension du modèle simplifié au modèle complet en intégrant une étape permettant de prendre en compte le comportement de l'utilisateur. La troisième partie de cette section est consacrée aux résultats : nous montrons, à travers les comparaisons de résultats pour différents modèles de génération aléatoire d'arborescences, que le modèle de Morì se révèle être un modèle satisfaisant qui produit des arborescences dont les propriétés approchent celles observées sur les structures réelles.

3.2.1 Le modèle stochastique : version de base

Le modèle que nous proposons ici est basé sur les principes de construction de structures dites *sans échelle*. Nous avons vu en section 1.4.2 qu'un procédé d'*évolution croissante* de la structure arborescente combiné avec une stratégie d'*attachement préférentiel* pour le choix des liens entre les sommets ont pour effet de produire des arbres respectant la propriété sans échelle, ce qui est le cas des arborescences de fichiers. Ainsi, notre modèle est basé sur la construction progressive d'un système de fichiers utilisateur. Le principe général de l'algorithme est d'ajouter, à chaque itération, un élément de type fichier (une nouvelle feuille dans l'arbre) que l'on place dans la structure arborescente. Cet ajout de fichier peut engendrer la création d'un sommet interne (*i.e.* un nouveau dossier) ou non.

D'autre part, nous avons mentionné dans la section précédente, le phénomène de *répertoires volumineux* dans les structures arborescentes étudiées. Le simple caractère sans échelle ne permet pas de reproduire ce phénomène de répertoires volumineux. Si l'on revient à la figure 3.1, nous pouvons observer que certains points s'écartent de la tendance générale de loi en puissance. Ces points, d'ordonnée faible (le plus souvent 1) et d'abscisse très élevée, représentent les quelques répertoires dont le degré est largement supérieur à celui des autres répertoires dans les arborescences : ce sont les *répertoires volumineux* qui sont greffés aux *arbres-squelette* (voir définition 42) des systèmes de fichiers.

Le principe général de l'algorithme de génération aléatoire est donc composé de trois parties : après initialisation, on génère l'arbre-squelette par un processus de génération aléatoire de structure sans échelle, ensuite, nous intégrons des répertoires volumineux à la structure obtenue. Ce principe général est illustré sur la figure 3.9.

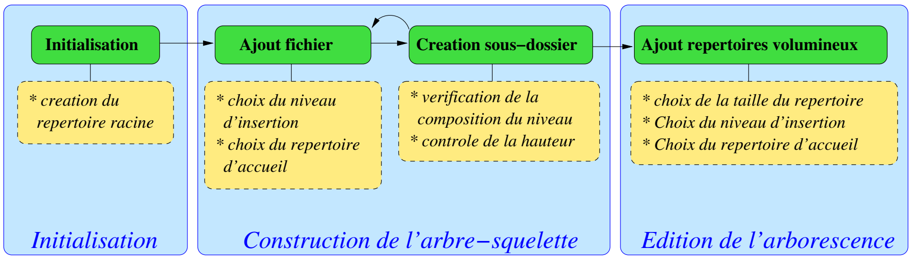


FIG. 3.9: Schéma général du modèle de génération aléatoire

Le modèle proposé est décomposé en trois parties : *(i)* la phase d'initialisation ; *(ii)* la phase de construction de l'arbre-squelette, qui consiste à générer l'arborescence de fichiers par ajouts successifs d'éléments à chaque itération ; et enfin *(iii)* la phase d'édition de la structure d'arbre-squelette qui permet d'intégrer le phénomène de répertoires volumineux.

Le coeur du processus de génération aléatoire de l'arborescence repose sur la génération de l'arbre-squelette (étape *(ii)*). Cette phase est décomposée en deux parties pour chaque itération :

1. ajout d'une feuille (création d'un nouveau fichier) et choix du répertoire d'accueil du fichier
2. si nécessaire, création d'un sous-répertoire pour accueillir le fichier

Dans ce modèle, nous distinguons les types d'éléments. Le nouveau sommet intégré à chaque itération correspond à un fichier, la création d'un nouveau répertoire n'intervient que lorsque l'ajout du fichier en question engendre cette nécessité.

3.2.1.1 L'ajout de fichier

À chaque itération de l'algorithme, un nouveau sommet est créé : l'utilisateur vient d'ajouter un fichier dans son arborescence. Il faut déterminer dans quel répertoire on va placer ce nouvel élément. Ce choix s'effectue en deux étapes : dans un premier temps, on choisit le niveau d'insertion du nouvel élément ; ensuite, on sélectionne un répertoire d'accueil dans le niveau supérieur au niveau choisi dans la première étape.

Afin de respecter la répartition des éléments par niveaux dans la hiérarchie observée sur les données réelles, pour l'insertion d'un nouvel élément, on sélectionne un niveau n dans la hiérarchie arborescente selon une loi de distribution gaussienne (voir propriété 3.4). Le fichier va donc être intégré au niveau n.

Il faut ensuite sélectionner parmi les répertoires du niveau supérieur dans la hiérarchie le répertoire qui contiendra ce fichier. Pour cette deuxième étape, on utilise la propriété sans échelle des données (propriété 3.1) : le choix du répertoire s'effectue selon une stratégie d'attachement préférentiel. Ainsi, la probabilité que le nouveau sommet soit connecté à un répertoire du niveau $n - 1$ dépend du contenu de ce répertoire (degré du répertoire). On retrouve ici la formule (1.3).

3.2.1.2 L'ajout d'un répertoire

Le modèle de génération aléatoire doit permettre la création de nouveaux répertoires lorsque cela est nécessaire. Pour décider si un nouveau répertoire doit être créé ou non, nous utilisons la propriété 3.5. Ainsi, on vérifie que l'ajout n'engendre pas un surnombre de fichiers (au vu de la composition des niveaux) au niveau n de la hiérarchie. Si tel est le cas, il est nécessaire de créer un sous-répertoire au niveau n et d'y placer le nouveau fichier afin de rééquilibrer la composition du niveau n.

Cependant, lorsque l'on se trouve sur le dernier niveau de la hiérarchie, l'intégralité des éléments qui composent ce niveau sont des fichiers. Ainsi, l'ajout d'un sous-répertoire devient systématique dès lors que l'on rajoute un fichier en bas de la hiérarchie arborescente. Par conséquent, l'arbre peut se prolonger indéfiniment en hauteur et ainsi violer la propriété 3.2. C'est pourquoi, la décision de créer un sous-répertoire nécessite la vérification de la validité de la propriété 3.2. Si cette propriété n'est pas vérifiée, alors on ne fait pas croître l'arbre en hauteur car le nombre d'éléments de l'arborescence n'est pas suffisant pour justifier de cet ajout de niveau dans la hiérarchie.

3.2.1.3 L'ajout des répertoires volumineux

La dernière étape de la génération aléatoire consiste à compléter les structures arborescentes en ajoutant des répertoires volumineux (Déf. 43). Cette phase se décompose en trois temps :

1. choix de la taille du répertoire volumineux
2. choix du niveau d'insertion du répertoire volumineux
3. choix du répertoire d'accueil

L'analyse des statistiques quant au nombre, à la taille et au placement de tels répertoires dans la base d'étude ont permis de définir un processus stochastique basé sur les propriétés observées.

Le choix de la taille du répertoire dépend du degré maximal dans l'arborescence (voir propriété 3.7). La taille du nouveau répertoire est déterminée par le choix d'un coefficient multiplicateur c tiré aléatoirement selon une loi en puissance. Ainsi, le nombre d'élément du *répertoire volumineux* à intégrer dans la structure arborescente correspond à c fois la taille du répertoire le plus gros de la hiérarchie.

En ce qui concerne le placement au sein de la structure arborescente, les mesures statistiques suggèrent une concentration des *répertoires volumineux* autour du quatrième niveau de la hiérarchie selon une distribution gaussienne (voir propriété 3.8). Le choix du niveau n d'intégration du répertoire dense suit cette loi de distribution.

Enfin, le répertoire d'accueil du dossier volumineux est choisi aléatoirement uniformément parmi l'ensemble des répertoires de l'arborescence du niveau $n - 1$.

L'inconvénient du modèle de génération aléatoire de base, tel qu'on vient de le décrire, est qu'il produit des arborescences aux mêmes caractéristiques. Or, nous avons évoqué la notion de comportement utilisateur (Sec. 3.1.5), qui influe sur la structuration des données. Nous proposons dans la section suivante une extension du modèle de base pour que les arborescences générées possèdent, outre les caractéristiques communes aux systèmes de fichiers, les spécificités induites par le comportement de l'utilisateur.

3.2.2 Le modèle stochastique : version étendue

Nous avons cherché à simuler dans notre modèle le comportement d'un utilisateur. Comme on l'a expliqué dans la section 3.1.5, l'aspect sémantique des données joue un rôle trop important pour que la simulation du comportement (stratégie de placement des éléments) soit suffisamment réaliste. D'autre part, il faudrait prendre en compte toutes les opérations possibles sur la structure arborescente (déplacement, suppression, compression, décompression d'une archive, installation d'un logiciel, ...), ce qui nécessiterai une étude approfondie du comportement des utilisateurs (capture journalière des arborescences sur une longue durée et analyse du type d'opérations effectuée). Cette étude reste à faire et constitue une perspective aux travaux réalisés au cours de cette thèse. Nous avons donc opté, dans un premier temps, pour un modèle aléatoire qui tend à respecter les propriétés structurelles spécifiques aux familles d'utilisateurs observées sur la base de données du laboratoire plutôt que de chercher à simuler le comportement d'un utilisateur.

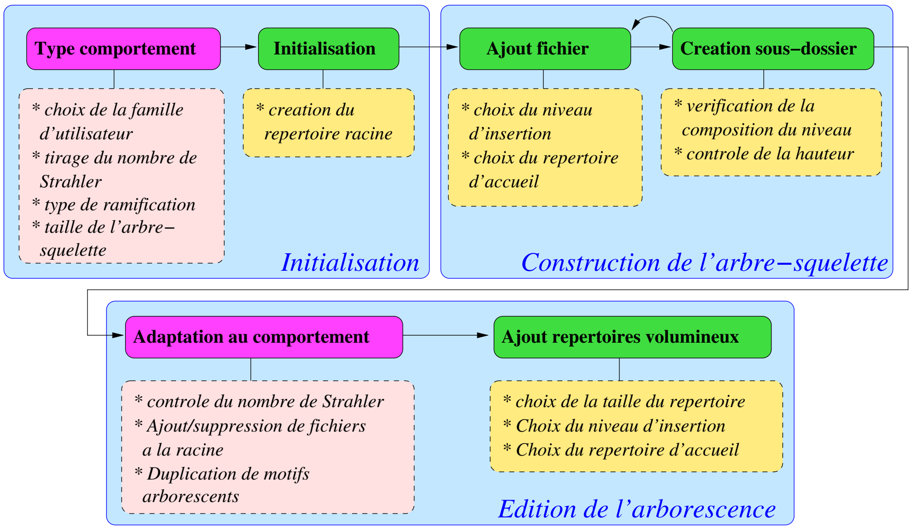


FIG. 3.10: Schéma général du modèle étendu de génération aléatoire

Le schéma de la figure 3.10 montre le principe général du modèle de génération étendu. Les étapes en sus du modèle de base (voir figure 3.9), sont colorées en rose.

Dans la phase d'initialisation, nous avons ajouté une étape qui permet de déterminer les spécificités de l'utilisateur (*i.e.* le choix du comportement, le choix de la famille d'utilisateur à laquelle il appartient). Le processus de génération aléatoire de l'arbre-squelette reste inchangé. C'est dans la phase d'édition de l'arbre-squelette pour obtenir l'arborescence définitive qu'une nouvelle étape entre en jeu. En effet, avant de procéder à l'ajout des répertoires volumineux, il nous faut vérifier que l'arbre-squelette possède les caractéristiques relatives à la famille d'utilisateur correspondante. Si tel n'est pas le cas et selon les spécificités de l'utilisateur, nous éditons l'arbre-squelette en dupliquant des motifs arborescents ou en ajoutant des éléments à la racine de l'arbre pour que cet arbre-squelette respecte les propriétés de la famille utilisateur correspondante.

Nous détaillons dans les sections suivantes ces deux nouvelles étapes.

3.2.2.1 Choix de la famille d'utilisateur

Durant la phase d'initialisation du modèle de génération aléatoire étendu, nous devons spécifier à quelle famille d'utilisateur correspond l'arbre que l'on va générer. Nous devons ensuite définir les propriétés spécifiques induites par cette appartenance.

Le processus se déroule en quatre étapes :

1. choix de la famille d'utilisateur
2. choix du nombre de Strahler
3. choix du type de ramification
4. choix de la taille de l'arbre-squelette

Pour le choix de la famille d'utilisateurs, nous nous basons sur la distribution de la figure 3.5.a. Ainsi, par un tirage aléatoire, l'arborescence sera classifiée comme appartenant à la famille d'utilisateur F_i, $i \in [1, \cdots, k]$.

Une fois la famille d'appartenance F_i identifiée, nous pouvons déterminer le nombre de Strahler à la racine de l'arbre-squelette que l'on souhaite obtenir (Déf. 45) : cette valeur est tirée aléatoirement en suivant les distributions gaussiennes que l'on peut observer sur la figure 3.5.a.

D'autre part, nous avons observé que le type de ramification est variable : la valeur du nombre de Strahler à la racine correspond parfois au degré à la racine, et parfois, elle est le résultat de la duplication d'un ou plusieurs motif(s) dans l'arborescence (Fig. 3.5.b). Afin que l'arbre-squelette réponde aux caractéristiques induites par sa famille d'appartenance, nous déterminons, dans cette phase d'initialisation, le type de ramification auquel correspond le nombre de Strahler à la racine. Le choix s'effectue, en fonction de leur probabilité d'apparition, entre les trois possibilités montrées sur la figure 3.5.b, à savoir : le degré à la racine, la duplication d'un motif arborescent dans un même répertoire, ou deux duplications d'un motif arborescent dans un même répertoire.

Enfin, à partir du nombre de Strahler de l'arbre-squelette, nous pouvons définir la taille approximative de l'arbre-squelette selon la propriété 3.6. Le nombre d'itérations du processus de construction de l'arbre-squelette est ainsi déterminé (on continue d'ajouter un élément tant que la taille de l'arbre-squelette n'est pas atteinte).

Après avoir déterminé toutes les caractéristiques spécifiques au comportement utilisateur que l'arbre-squelette devra respecter, nous procédons à la génération aléatoire de l'arbre-squelette générique. Cette phase est la même que dans le modèle de base. Nous procédons ensuite aux modifications de cette structure squelette pour qu'elle réponde aux spécificités du comportement utilisateur définies dans la phase d'initialisation.

3.2.2.2 Édition de l'arbre-squelette

Une fois l'arbre-squelette généré, nous procédons à l'édition de cette structure pour que tous les éléments déterminés dans la phase d'initialisation soient vérifiés. Dans un premier temps, on contrôle si le nombre de Strahler à la racine de l'arbre correspond au nombre de Strahler attendu (à un facteur de tolérance ϵ près). Si la valeur obtenue est inférieure au nombre de Strahler attendu, alors, selon le type de ramification de l'arborescence, on va, soit rajouter des fichiers à la racine, soit créer un motif arborescent que l'on va dupliquer une ou deux fois dans un même répertoire pour obtenir la valeur de Strahler désirée. Le motif arborescent que l'on génère pour l'instant est un répertoire qui ne contient que des fichiers, positionné dans un répertoire choisi aléatoirement uniformément parmi l'ensemble des répertoires de l'arborescence.

Si par contre, le nombre de Strahler obtenu est supérieur à la valeur fixée dans la phase d'initialisation, alors on rejette l'arbre-squelette et on procède à une nouvelle génération.

Complexité Il est clair que la complexité de l'algorithme de génération aléatoire que nous venons de décrire est linéaire (pour chacun des fichiers et répertoires volumineux nouvellement créés, nous effectuons un nombre constant d'opérations).

Nous venons de définir un nouveau modèle pour la génération aléatoire d'arborescences de fichiers. Dans la section qui suit, nous proposons une comparaison des résultats produits par notre nouveau modèle avec les arbres générés par différents modèles existants.

3.3 Résultats

Nous présentons dans cette section les résultats obtenus pour la génération aléatoire d'arborescence de fichiers en utilisant trois modèles : le modèle de Barabási-Albert [12], le modèle de Morì [96] présentés dans le chapitre d'introduction, ainsi que le modèle que nous avons proposé (section 3.2.2). Les résultats montrés ici ne concernent que la tranche des arbres de taille comprise entre 10000 et 15000 sommets. Les résultats obtenus pour les autres tranches de taille sont montrés en annexe A.

Pour les tests, nous avons généré pour chaque modèle un jeu de 100 arborescences dans la fourchette de taille considérée.

3.3.1 Distribution des degrés

Dans un premier temps, nous contrôlons si la propriété sans échelle des données générées est vérifiée pour ces modèles. La figure 3.11 montre une comparaison de la distribution des degrés des éléments de la base de données avec celles obtenues pour les arborescences générées avec les modèles de Barabási-Albert et Morì.

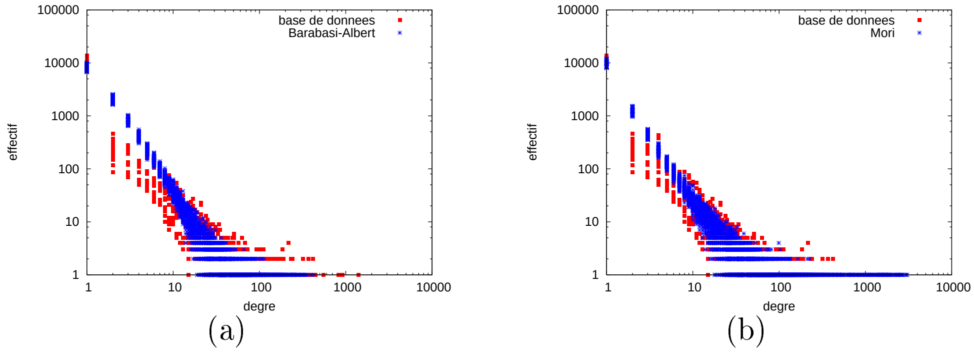


FIG. 3.11: Distribution des degrés des modèles (a) Barabási-Albert et (b) Morì.

Nous pouvons constater ici que la distribution observée sur le jeu de données original présente une légère courbure pour les faibles valeurs alors que les distributions obtenues par les modèles de Barabási-Albert et Morì sont plutôt linéaires. Ceci est dû au fait que dans ces modèles, on ne distingue pas les éléments : par convention, on admet qu'un sommet feuille correspond à un fichier et un sommet interne représente un dossier. Cependant, durant le processus de génération aléatoire, un sommet feuille peut être sélectionné comme répertoire d'accueil. On ne contrôle pas dans ce cas la création de répertoires et il résulte un surnombre de sommets de très faible degré par rapport aux données réelles.

Nous avons intégré une fonction de contrôle de création de répertoires dans les modèles de Barabási-Albert et Morì afin d'obtenir une distribution des degrés plus proches de la courbure obtenue sur les données réelles. La phase d'amélioration est la suivante : lors de l'insertion d'un nouvel élément dans l'arborescence, un sommet dans la structure existante est sélectionné comme répertoire d'accueil. Si ce sommet est une feuille, alors il ne sera plus considéré comme un fichier mais comme un répertoire. À ce moment là, nous n'autorisons la conversion fichier-répertoire qu'après avoir contrôlé que ce changement n'engendre pas un surnombre de répertoire dans le niveau concerné (voir propriété 3.5).

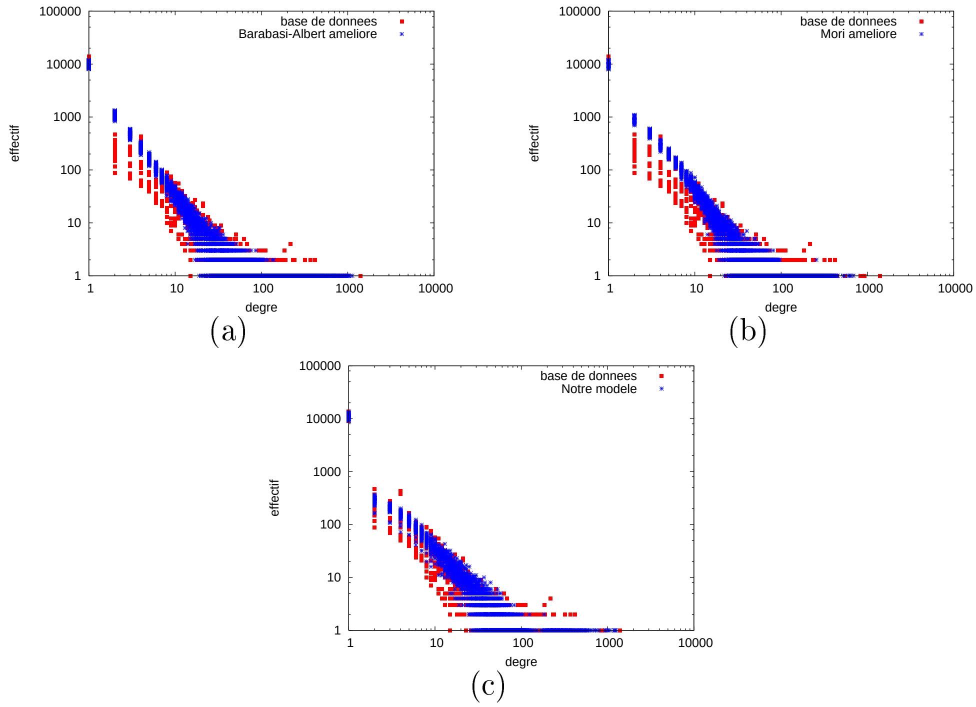


Fig. 3.12: Distribution des degrés des modèles (a) Barabási-Albert amélioré, (b) Morì amélioré et (c) notre modèle.

Les figures 3.12.a et 3.12.b montrent les distributions pour ces deux modèles améliorés. Les distributions semblent plus proches des données réelles, en particulier le modèle de Morì amélioré. Enfin, la figure 3.12.c illustre le résultat obtenu pour les arborescences générées via le modèle proposé dans cette thèse. Dans notre modèle, chaque sommet ajouté à la structure arborescente est défini comme étant un fichier ou un répertoire. Ainsi, la création de répertoires est contrôlée et la distribution obtenue ici correspond à celle du jeu de données de référence.

3.3.2 Forme des arbres

Nous étudions ici les propriétés sur la forme des arborescences. Nous avons vu que la largeur et la hauteur d'un système de fichiers étaient fortement corrélées au nombre d'éléments contenus dans l'arbre. Sur la figure 3.13, sont affichées, pour chacun des modèles, les hauteurs des arborescences en fonction du nombre d'éléments.

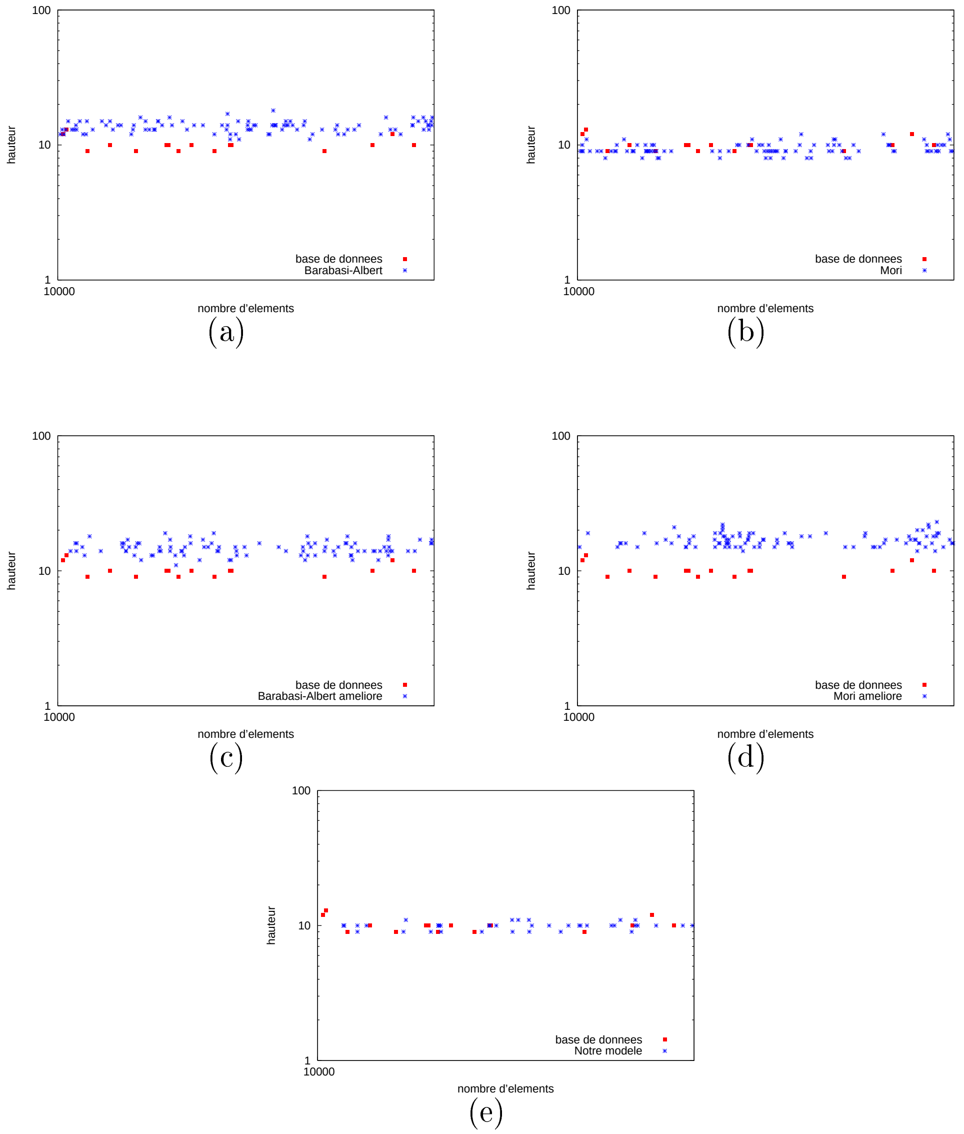


FIG. 3.13: Hauteur des arbres générés par les modèles (a) Barabási-Albert, (b) Morì, (c) Barabási-Albert amélioré, (d) Morì amélioré et (e) notre modèle.

Nous pouvons observer sur ces données que deux modèles s'approchent des caractéristiques du jeu de données initial : le modèle de Morì (Fig. 3.13.b) et notre modèle (Fig. 3.13.e). Les autres modèles ont tendance à produire des arbres trop hauts. Dans la section 3.1.3, nous avons expliqué qu'une arborescence de fichiers ne croît en hauteur que lorsque le nombre d'éléments devient trop élevé. En effet, l'utilisateur cherche à minimiser le nombre de répertoire à parcourir pour accéder à ses données. Cependant, il est parfois nécessaire de mieux hiérarchiser les données pour une prise de décision rapide au sein d'un répertoire. Dans notre modèle, nous contrôlons l'ajout de niveaux dans la hiérarchie, ainsi, la hauteur des arborescences obtenues respectent les propriétés observées sur les données réelles.

En ce qui concerne la largeur des arbres, la figure 3.14 montre la relation entre le nombre d'éléments et la largeur de l'arbre. Nous pouvons constater que le modèle de Morì simple (Fig. 3.14.b) reproduit cette propriété dans les arbres générés. Nous avons vu que les autres modèles existants génèrent des arbres qui s'étendent trop en hauteur. Il en résulte un étirement de la structure en hauteur au détriment de la largeur.

Notre modèle (Fig. 3.14.e) produit ici encore des arborescences dont la largeur concorde avec celle des données existantes.

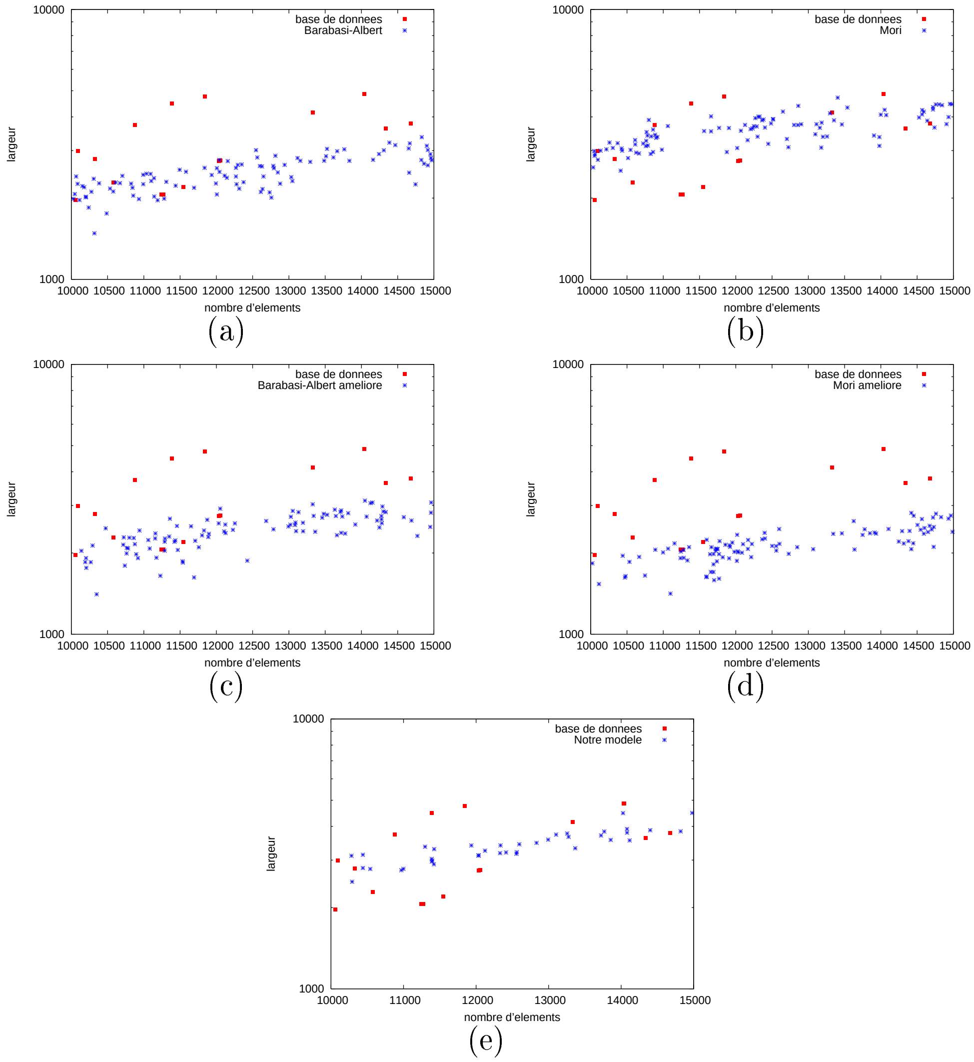


FIG. 3.14: Largeur des arbres générés par les modèles (a) Barabási-Albert, (b) Morì, (c) Barabási-Albert amélioré, (d) Morì amélioré et (e) notre modèle.

3.3.3 Distribution des éléments

Une autre propriété observée sur les données réelles concerne la distribution des éléments selon le niveau dans l'arbre. Nous avons observé que les éléments sont répartis dans les niveaux de l'arborescence selon une distribution gaussienne.

La figure 3.15 montre les distributions des éléments par niveau pour chacun des modèles testés. Nous observons sur cette figure que la distribution qui s'approche le plus des distributions existantes est celle produite par notre modèle de génération. Ici encore, cette propriété est utilisée dans notre modèle : on utilise cette distribution pour déterminer à quel niveau de l'arborescence va s'ajouter le sommet nouvellement créé.

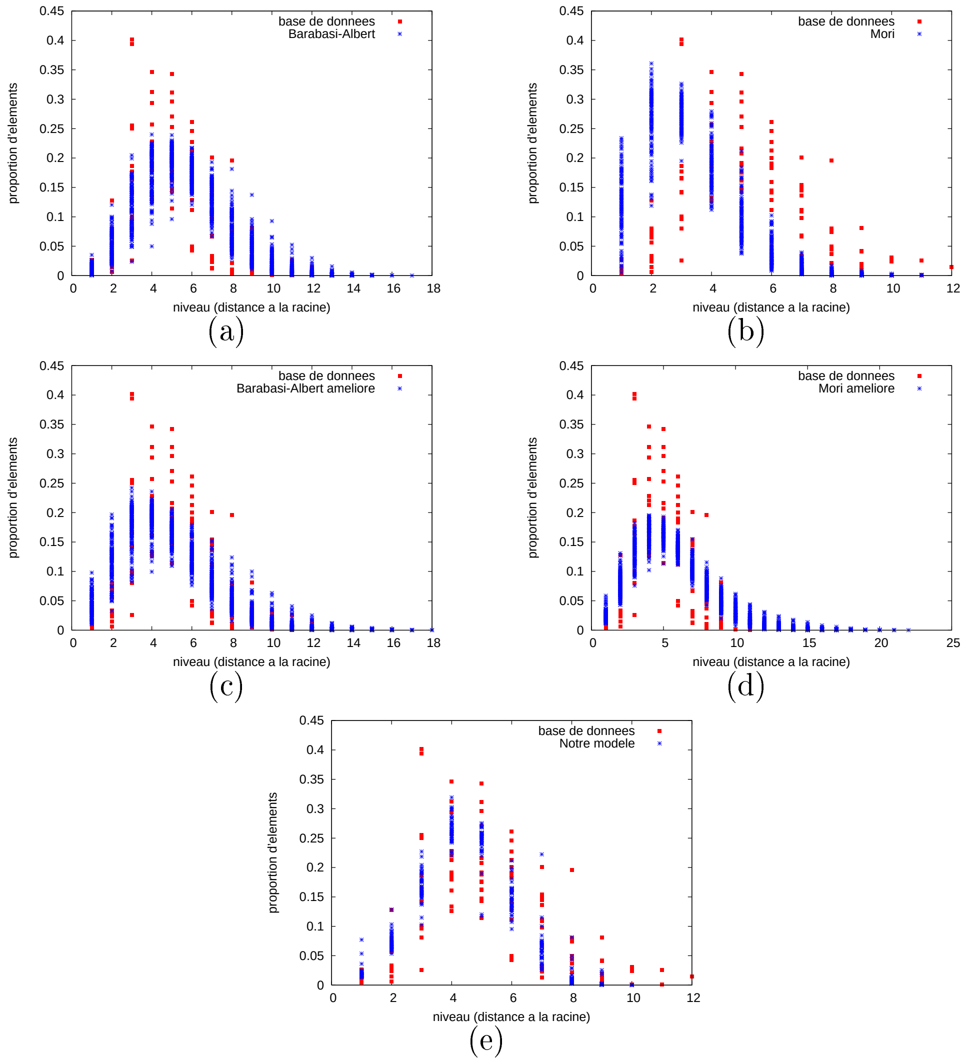


FIG. 3.15: Distribution des éléments selon le niveau dans les arborescences générés par les modèles (a) Barabási-Albert, (b) Morì, (c) Barabási-Albert amélioré, (d) Morì amélioré et (e) notre modèle.

On retrouve d'autre part les observations de la section précédente : pour les modèles de Barabási-Albert (simple et étendu) et Morì étendu, les données sont plus disparates que celles du jeu de données, on a une plus faible concentration autour du niveau moyen de l'arbre. Ainsi, ces modèles produisent des arborescences plus hautes et moins larges que les arborescences du jeu de données.

En ce qui concerne le modèle de Morì, nous avons observé une hauteur et une largeur d'arbre qui correspondent aux données réelles. La distribution des éléments quant à elle n'est pas complètement similaire. On observe sur la figure 3.15 que la courbe obtenue est légèrement décalée sur la gauche : les éléments sont plus proches de la racine.

3.3.4 Composition des niveaux

Nous nous proposons maintenant d'étudier la composition des niveaux des arbres générés. Sur la figure 3.16 sont représentées, pour chaque niveau de l'arborescence, les proportions de fichiers par rapport au nombre d'éléments total dans le niveau correspondant.

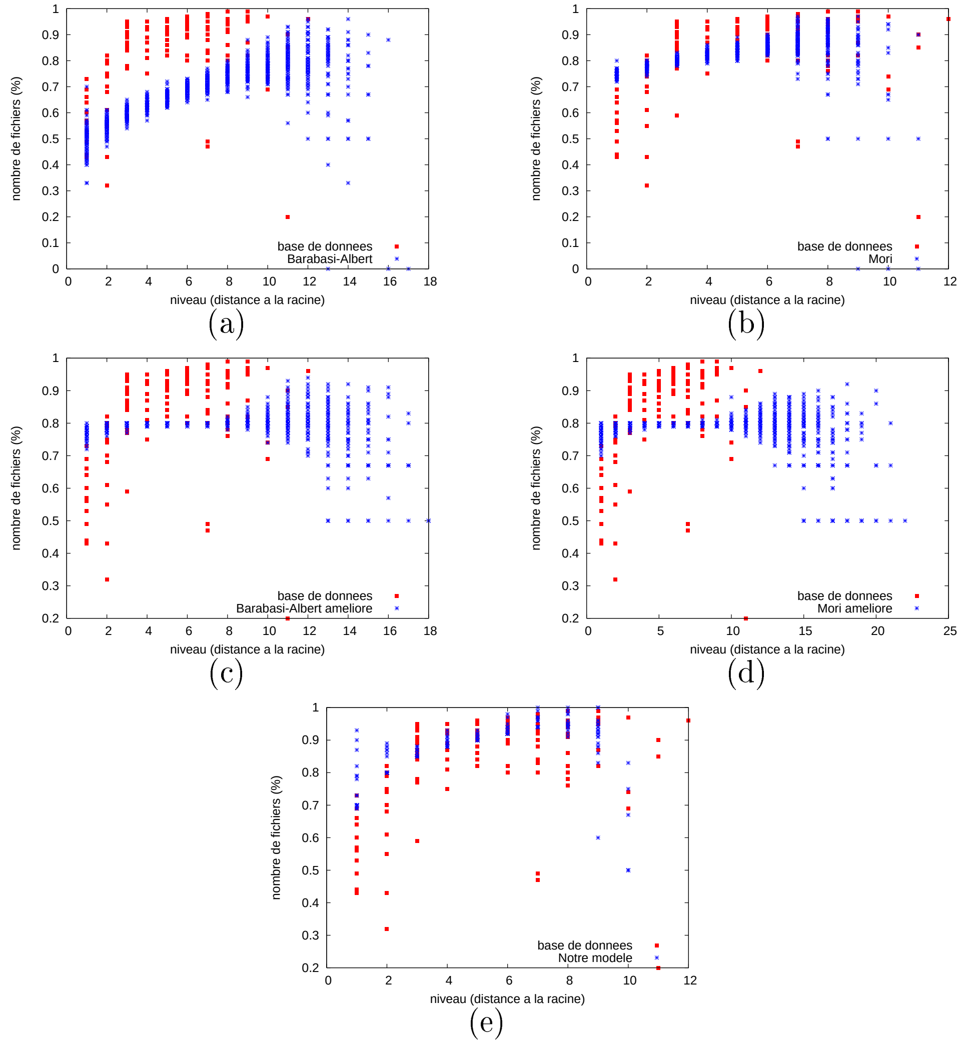


FIG. 3.16: Répartition des éléments dans les niveaux des arbres générés par les modèles (a) Barabási-Albert, (b) Morì et (c) notre modèle.

Nous pouvons observer sur cette figure qu'une fois encore, les modèles de Barabási-Albert (simple et amélioré) et le modèle de Morì amélioré produisent des données qui ne correspondent pas aux propriétés observées dans la réalité. Nous pouvons constater qu'il y a globalement plus de répertoires dans les arborescences générées par ces modèles que dans les systèmes de fichiers réels. En effet, comme nous l'avons dit précédemment, nous ne distinguons pas, durant le processus de génération aléatoire, le type de chaque sommet (fichier ou répertoire). Il en résulte la création de nombreux sommet de très faible degré qui sont alors associés à des répertoires de très petite taille.

Les deux autres modèles, à savoir le modèle de Morì simple et le modèle que nous avons proposé, produisent des résultats très proches de la réalité. Notre modèle produit des arbres dont la distribution des éléments suit la tendance générale observée sur la figure 3.4. En effet, cette propriété est utilisée pour décider de la création de répertoires.

3.3.5 Récapitulatif

Nous avons dressé le tableau récapitulatif 3.2 des résultats obtenus pour les différents modèles de génération testés. Nous avons attribué pour chaque propriété et chacun des modèles, une évaluation des performances du modèle parmi les trois notes :

- les résultats sont trop éloignés (-) ,
- les résultats sont convenables (+), et
- les résultats correspondent avec la réalité (+++).

	Barabási Albert	Barabási Albert étendu	Morì	Morì étendu	Notre modèle
Distribution des degrés	+	+	+	+++	+++
Hauteur de l'arborescence	+	+	+++	-	+++
Largeur de l'arborescence	+	+	+++	+	+++
Distribution des éléments	+	+	-	-	+++
Composition des niveaux	+	-	+++	-	+++

Tab. 3.2: Comparaison des résultats pour la génération aléatoire d'arborescences de fichiers

Parmi les modèles existants que nous avons testés, le modèle de Morì est un modèle qui permet de générer des arborescences aux caractéristiques très proches de celles observées sur les données réelles, à l'exception de la distribution des éléments selon le niveau. Ce modèle simple constitue cependant un très bon candidat pour reproduire des arbres semblables aux systèmes de fichiers.

Notre modèle offre les meilleurs résultats. Ce dernier, plus spécifique, prend de nombreuses propriétés en compte pour la génération aléatoire. Il en résulte des arborescences dont les caractéristiques sont similaires aux arbres réels.

Il reste à vérifier que notre modèle respecte le phénomène de familles d'utilisateurs. Nous nous proposons d'étudier cet aspect dans la section suivante.

3.3.6 Système de fichier complet

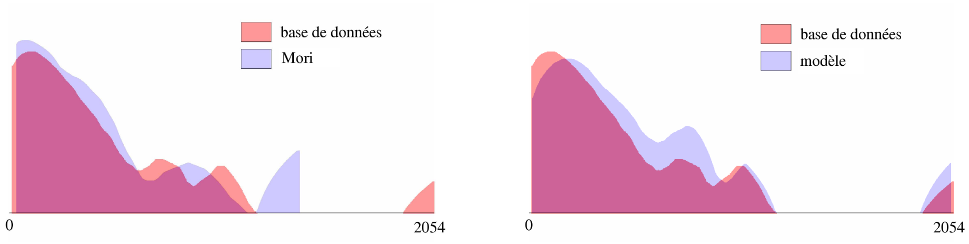
Nous venons d'étudier les résultats de la génération aléatoire pour chacune des différentes tranches de tailles d'arbres indépendemment. Il est intéressant maintenant d'observer les résultats sur un système de fichiers complet, à savoir un système comprenant les arborescences de plusieurs utilisateurs différents.

3.3.6.1 Comparaison avec le système de fichiers de référence

Nous avons choisi, pour ce test, de générer un système de fichier similaire au système de fichier réel (le plus récent) utilisé pour ces travaux. Ainsi, nous avons choisi aléatoirement uniformément, parmi les arborescences générées pour les tests précédents, 83 arbres dont la répartition par tranche suit la même distribution que celle de la table 3.1 (2ème capture).

L'objectif ici est de vérifier la présence de familles d'utilisateurs dans un système de fichiers complet artificiel. Pour ce faire, nous devons analyser les valeurs de Strahler à la racine pour l'ensemble des arbres-squelette du système de fichier complet.

La figure 3.17 montre la répartition de ces valeurs de Strahler pour des systèmes de fichiers complets suivants : le système de fichiers complet réel issu de la deuxième capture comparé avec les systèmes de fichiers générés obtenus avec le modèle de Morì d'une part (Fig. 3.17.a), et avec notre modèle d'autre part (Fig. 3.17.b).



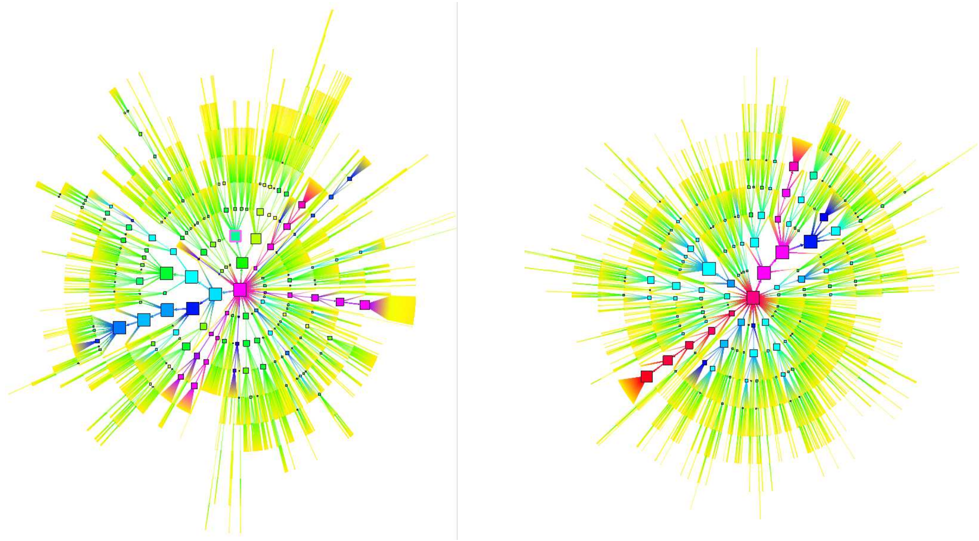
(a) (b)

Fig. 3.17: Distribution des valeurs de Strahler à la racine des arbres élagués.

Nous pouvons observer sur ces résultats que notre modèle de génération reproduit le phénomène de familles d'utilisateurs. La distribution des valeurs de Strahler à la racine des arbres-squelettes est très proche de celle obtenue sur le jeu de données. Le modèle de Morì ne présente pas une telle similitude avec les données réelles. Bien que l'on puisse distinguer des tendances (trois gaussiennes), on ne retrouve pas les mêmes quatre familles observées dans la réalité.

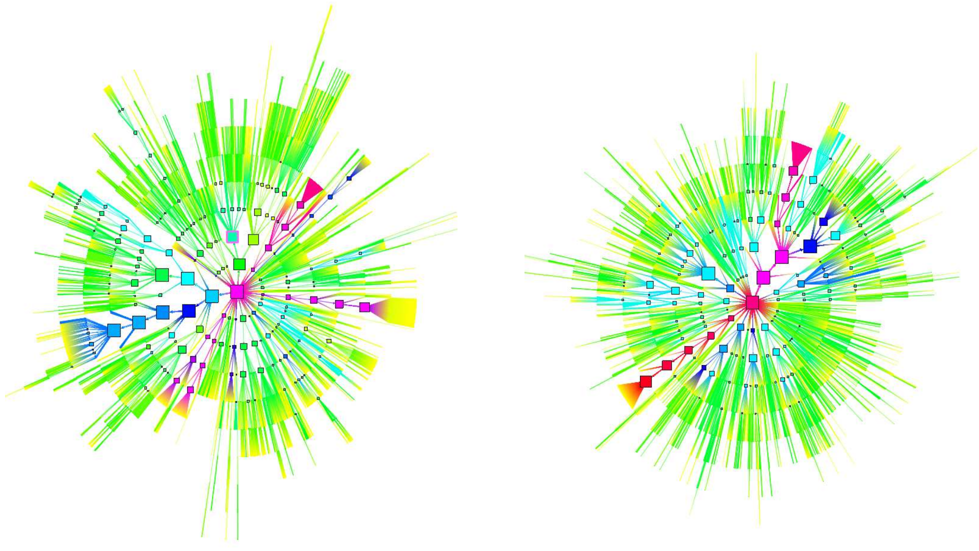
Nous avons par ailleurs cherché à vérifier que les arborescences générées par modèle de génération aléatoire proposé dans ce manuscrit et les données réelles comportent des motifs quasi-similaires. Ainsi, nous avons appliqué l'heuristique de recherche de motifs quasi-similaires décrite dans le chapitre 2 sur l'exemple précédent pour comparer le système de fichiers de référence avec le système de fichiers artificiel.

La figure 3.18 montre le résultat de la classification des sommets en familles de sommets ϵ-similaires. Dans cet exemple, nous avons retenu les propriétés structurelles suivantes : le degré du sommet, la taille du sous-arbre, le nombre de Strahler et la profondeur du sommet. L'étape de construction de motifs quasi-similaires par propagation est illustrée par la figure 3.19.



(a) (b)

FIG. 3.18: Familles de sommets ϵ-similaires entre le système de fichiers de référence (a) et un système de fichier généré avec notre modèle (b).

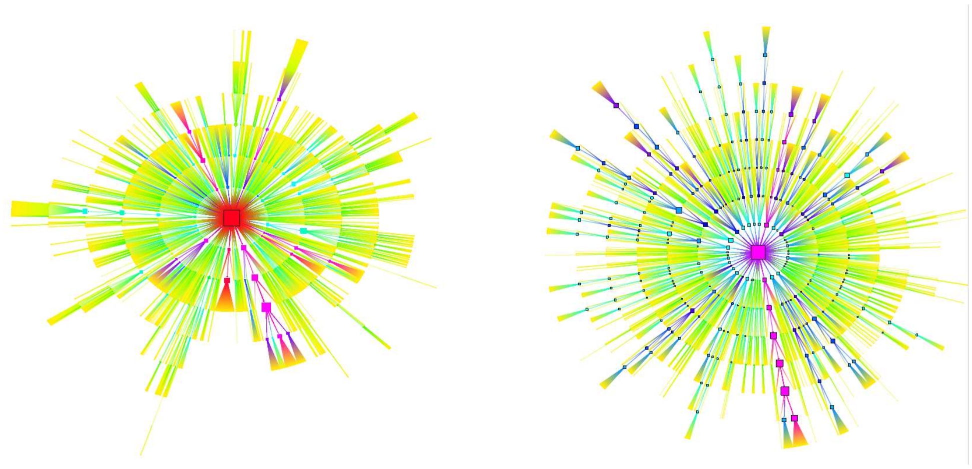


(a) (b)

FIG. 3.19: Motifs quasi-similaires entre le système de fichiers de référence (a) et un système de fichier généré avec notre modèle (b).

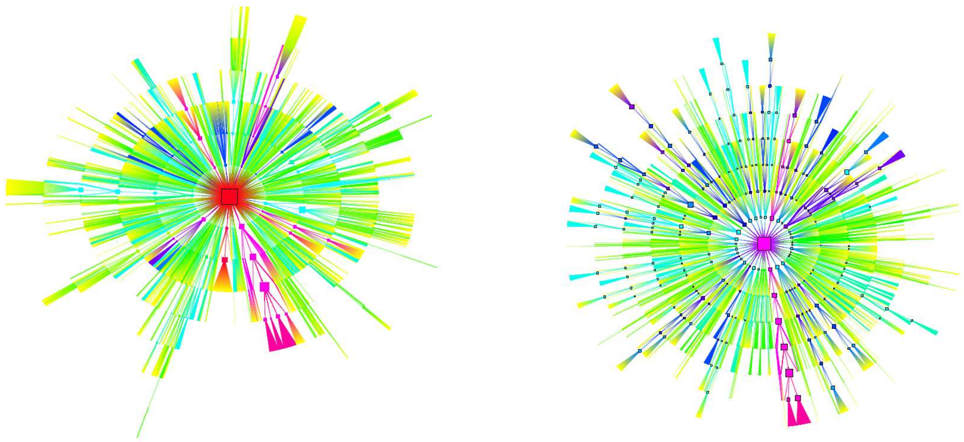
Nous pouvons observer sur cet exemple qu'un grand nombre de motifs ont été identifiés : les motifs verts très nombreux, correspondent à des arborescences filiformes, les motifs bleu ciels et rose représentent des structures plus ramifiées.

Dans l'exemple qui suit, nous avons comparé un système de fichiers artificiel généré via notre modèle avec les données fournies lors du concours Infovis 2003 [52]. L'arborescence Infovis représente le système de fichiers de l'Université du Maryland, avec entre autres, le répertoire contenant l'ensemble des pages personnelles des membres de l'Université. Nous avons extrait ce répertoire comme système de fichiers utilisateurs de référence. La figure 3.20 montre le résultats de la classification des sommets en familles et la figure 3.21 le résultat de la reconnaissance de motifs quasi-similaires par propagation. On note ici aussi de nombreuses similitudes entre les deux systèmes.



(a) (b)

Fig. 3.20: Familles de sommets ϵ-similaires entre le système de fichiers de référence (a) et un système de fichier généré avec notre modèle (b).



(a) (b)

Fig. 3.21: Motifs quasi-similaires entre le système de fichiers de référence (a) et un système de fichier généré avec notre modèle (b).

L'identification de motifs similaires entre les données réelles et les données générées tend à valider le modèle que nous avons proposé dans ce chapitre.

3.4 Conclusion et perspectives

Dans ce chapitre, nous avons proposé une étude des caractéristiques des arborescences représentant des systèmes de fichiers utilisateurs. Nous avons extrait un ensemble de propriétés structurelles qui caractérisent ces arbres particuliers. À partir de ces observations, nous avons proposé un nouveau modèle pour la génération aléatoire d'arborescences aux caractéristiques proches des systèmes de fichiers utilisateurs. Le modèle décrit dans ce chapitre intègre dans son processus de construction certaines des propriétés établies sur le jeu de données utilisé.

Nous avons comparé les résultats obtenus avec certains des modèles classiques de génération aléatoire. Parmi ces modèles existants, nous avons relevé le modèle de Morì comme étant un modèle simple qui produit des arborescences proches des systèmes de fichiers réels. Cependant, les tests ont montré que notre modèle offre de meilleurs résultats.

L'aspect dynamique des données n'a pas été exploité ici. Il serait intéressant d'effectuer une étude approfondie des changements dans les arborescences afin d'identifier comment un utilisateur transforme son système de fichiers. Il est important dans ce cas de connaître le contenu sémantique des éléments manipulés par l'utilisateur afin de différencier les différentes opérations d'édition de l'arborescence. Pour comprendre et reproduire le comportement d'un utilisateur, il faut pouvoir différencier par exemple si l'apparition d'un nouveau motif arborescent est le résultat de la décompression d'une archives ou d'un logiciel fraîchement installé. Une telle étude permettrait de caractériser clairement les opérations qu'effectue un utilisateur (création, déplacement, suppression, archivage, décompression, installation, ...) en ajoutant l'aspect sémantique aux données manipulées (image, développement, courrier administratif, ...).

Nous avons validé le modèle de génération aléatoire proposé en comparant les résultats obtenus avec les données réelles via l'heuristique de reconnaissance de motifs similaires présentée dans le chapitre précédent. En effet, des similitudes ont été détectées entre le système de fichiers artificiels et le données réelles.

Chapitre 4

Mise en correspondance d'objets issus d'une vidéo

La recherche automatique de documents visuels (vidéos, images) dans des bases de données de très grande taille est aujourd'hui essentielle. En effet, la généralisation des supports numériques, l'apparition de formats image et vidéo compressés, la chute du coût des médias de stockage, ont engendré une augmentation exponentielle de la quantité de données multimédia. Pour permettre une utilisation efficace des bases de données multimédia, il est désormais indispensable de disposer d'outils permettant de sélectionner, filtrer les données en se basant sur une indexation sophistiquée. Par exemple, la recherche d'images dans une base de données peut se faire en utilisant l'image de référence et en sélectionnant automatiquement celles qui lui ressemblent.

La gestion manuelle des bases de données multimédia est désormais impossible. Dans un tel contexte, pour que les données soient exploitables, il s'agit d'analyser des contenus extrêmement divers dans les délais les plus courts.

Dans ce chapitre, nous présentons deux méthodes que nous avons développées pour la recherche d'objets d'intérêt dans une base de données vidéo basse résolution [30, 31]. Les deux méthodes se basent sur la comparaison de graphes représentant des objets extraits du flux vidéo. En effet, nous ramenons le problème de la comparaison d'objets à un problème de mise en correspondance de graphes en associant à chaque objet vidéo un graphe basé sur la relation de voisinage entre régions. La comparaison de graphes utilise une mesure de similarité entre les régions des objets, *i.e* entre les sommets des graphes.

Ces travaux se situent dans le contexte du *paradigme d'indexation grossière* ("rough indexing paradigm") [29, 30, 31, 87] dont l'objectif consiste à fournir des outils interactifs pour l'indexation de données multimédia. Pour atteindre des temps de réponses interactifs, on s'appuie sur un décodage partiel du flux vidéo mais les données que l'on manipule sont de faible résolution, aussi bien du point de vue spatial que temporel. Les données initiales sont exploitées au niveau du bloc (8 x 8 pixels), ce qui engendre le sous-échantillonnage de l'image induisant un fort lissage de l'information colorimétrique et géométrique. De plus, nous ne considérons pas la totalité des images qui composent le flux vidéo original à 24 images par seconde. Par conséquent, deux images successives du flux partiellement décodé sont espacées dans le temps et peuvent présenter des différences notables.

Notre objectif est de proposer des méthodes adaptées à la faible résolution des données manipulées. Nous avons ainsi proposé deux méthodes de comparaison de graphes. La première repose sur les techniques de relaxation probabiliste [31] que nous avons brièvement évoquées dans la section 4.2. La seconde méthode concerne l'adaptation de la méthode heuristique présentée dans le chapitre 2 [30, 29].

Le chapitre est organisé comme suit. Dans une première section, nous décrivons la nature des données manipulées puis nous proposons une formalisation du problème. Dans les deuxième et troisième sections, nous détaillons les deux méthodes de comparaison de graphes : la première, basée sur les techniques de relaxation en section 4.2 et la deuxième, adaptation de la méthode heuristique en section 4.3. Enfin, la section 4.4 est consacrée à une étude comparative des résultats des deux méthodes développées.

4.1 La chaîne de traitement

Le résultat d'une segmentation d'image est représentable par une partition de l'image en régions, une région étant un ensemble connexe de pixels. L'une des représentations les plus usuelles pour modéliser les relations d'adjacence entre les éléments de l'image (*i.e.* les régions) est la structure de graphe d'adjacence des régions (RAG) [57, 138]. Dans cette représentation, chaque élément de la partition est associé à un sommet dans le RAG, et deux sommets sont connectés, si les régions correspondantes sont adjacentes dans l'image.

Dans cette section, nous commençons par préciser la nature des données manipulées, à savoir les sous-parties de l'image correspondant à la segmentation des objets de premier plan extraits de la vidéo. Nous proposons ensuite une discussion sur les contraintes liées au paradigme d'indexation grossière. Enfin, nous détaillons le processus de construction du graphe associé à un objet.

4.1.1 Description des données

Les données que nous manipulons dans ces travaux sont issues de la décompression partielle du flux vidéo compressé MPEG-2 [56]. Le format de compression vidéo MPEG se base principalement sur une compression spatiale et temporelle du flux en pleine résolution.

4.1.1.1 La résolution spatiale

La compression spatiale des images d'un flux MPEG est basée sur la décomposition des images en blocs de 8×8 pixels. Chacun des blocs de l'image subit une transformée DCT ("Discrete Cosine Transform", [56]) qui produit 64 coefficients correspondant à la composition de différentes fréquences spatiales. Le premier coefficient, également appelé coefficient DC, correspond à la valeur moyenne du signal dans le bloc (*i.e.* la couleur moyenne).

Dans notre contexte du paradigme d'indexation grossière, nous considérons une représentation réduite de chaque image de la vidéo. Cette image est obtenue à partir des coefficients DC, sans avoir à reconstruire le signal complet. Ainsi, l'analyse concerne des images 64 fois plus petites que les images en pleine résolution, ce qui implique un fort lissage des données colorimétriques et géométriques. On appelle ces images les *images DC*.

4.1.1.2 La résolution temporelle

Le deuxième aspect de la compression des standards MPEG repose sur la reconstruction de l'information redondante par estimation de mouvement. Il existe trois types d'images dans un flux vidéo compressé MPEG : les I-frames (intra-frames), les P-frames (predictional-frames) et les B-frames (bidirectional frames).

Les I-frames correspondent aux images du flux qui sont entièrement codées par le processus de compression par blocs mentionné dans le paragraphe précédent. Ces images sont complètement indépendantes de toute autre information de la vidéo, mais ont l'inconvénient d'être volumineuses en terme de stockage.

Les P-frames sont obtenues de manière plus complexe. En supposant que les images successives dans une vidéo sont en grande partie similaires, la stratégie consiste à trouver des blocs de l'image précédente qui ont simplement subit un déplacement et de ne coder que le vecteur de déplacement. Dans le cas où aucune correspondance n'a été détectée avec l'image précédente, le bloc est encodé comme pour une image I.

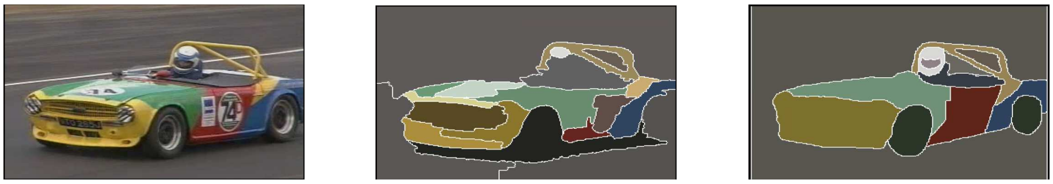
Une technique analogue est utilisée pour les B-frames, mais l'information est obtenue par interpolation entre l'image P ou I précédente et l'image P ou I suivante.

Un flux vidéo MPEG est composé d'une succession d'un même motif d'images vidéo : `IBBPBBPBB`, à une résolution temporelle de 24 images/seconde. Cependant, dans notre approche d'indexation grossière, nous ne considérons pas la totalité des images qui composent le flux car : d'une part, la base de données d'images pour une seule minute de film comprend près de 1500 images, ce qui est très élevé pour une indexation grossière ; d'autre part, les images P et B sont obtenues par reconstruction du signal et nécessitent donc un temps de calcul supplémentaire à la simple analyse des images I disponibles directement. Ainsi, nous avons choisi de ne considérer que les images I du flux vidéo pour s'affranchir des étapes de reconstruction du signal et limiter le nombre d'objets à comparer. Nous travaillons donc à une résolution temporelle d'environ 2 images/seconde, soit 160 images pour une minute de vidéo.

4.1.1.3 Extraction et segmentation des objets de premier plan

La segmentation d'images a pour but de fournir une partition de l'image en régions. Ces régions sont construites en considérant des propriétés d'homogénéité ; l'homogénéité étant définie sur la distribution de différents paramètres visuels (texture, intensité, ...). De même, les propriétés géométriques de la région peuvent être un critère de construction de la région. Ainsi, le processus de segmentation se base sur les propriétés physiques de l'image et en aucun cas sur les propriétés sémantiques. Une étape de reconnaissance consiste à regrouper entre elles certaines régions pour en donner une interprétation sémantique. Toutefois, ce regroupement sémantique peut s'avérer particulièrement délicat et doit intégrer nécessairement des connaissances *a priori*.

Nous illustrons sur un exemple la difficulté pour une méthode automatique de regrouper les régions sémantiques entre elles (Fig. 4.1). Sur la figure 4.1.b, on présente le résultat d'une segmentation automatique de l'image en zones homogènes selon un critère colorimétrique. On remarque que du point de vue sémantique, cette partition présente des défauts (les roues de la voiture sont assimilées à l'ombre du véhicule). Dans le cas d'une segmentation semi-automatique par un utilisateur humain, on obtient une partition qui ressemble à celle de la figure 4.1.c. On note que dans ce cas, l'utilisateur a intégré ses connaissances *a priori* lui permettant de distinguer les roues de la voiture de l'ombre et du fond de l'image..



(a) (b) (c)

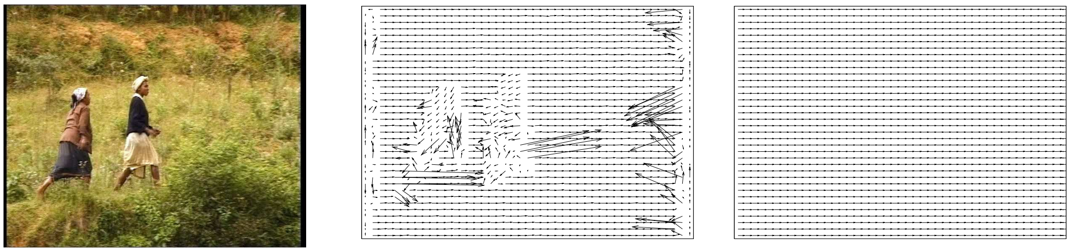
Fig. 4.1: Exemple de segmentation d'image. (a) Image originale, (b) segmentation automatique et (c) segmentation semi-automatique.

Comme illustré précédemment, dans le cadre de la segmentation d'une image, l'extraction des objets d'intérêt est une tâche relativement complexe qui requiert très souvent l'intervention d'un utilisateur (Fig. 4.2). Par contre, dans le cadre de la vidéo, un objet d'intérêt peut être caractérisé par son mouvement propre et c'est cette information qui permet d'extraire les objets de premier plan, car leur mouvement se distingue du mouvement global de la caméra.

Fig. 4.2: Objet de premier plan de la figure 4.1.

L'algorithme utilisé dans ces travaux [87] pour l'extraction d'objets de premier plan dans une vidéo repose sur l'analyse du mouvement. En utilisant les standards MPEG, on construit le champ de vecteurs de déplacement des blocs de l'image (Fig. 4.3.b). Ensuite, par un processus d'analyse hiérarchique [36], on obtient une estimation du mouvement global de la caméra comme l'illustre la figure 4.3.c. Les blocs de l'image dont le mouvement diffère de celui de la caméra sont considérés comme appartenant à un objet de premier plan.

À partir de ces informations, on construit un *masque d'objet* (Déf. 46) permettant de déterminer dans l'image DC, l'ensemble des pixels qui correspondent à l'objet de premier plan.



(a) (b) (c)

FIG. 4.3: Exemple de compensation de mouvement. (a) Image P, (b) champ de vecteurs de mouvement et (c) mouvement global de la caméra.

Définition 46 (Masque d'objet de premier plan)
Le masque d'objet de premier plan est une image binaire telle que : un pixel a pour valeur 1 *s'il appartient à l'objet, et* 0 *sinon (s'il appartient au fond).*

Une fois le masque d'objet obtenu, une segmentation, basée sur un algorithme de croissance des régions inspiré des techniques de ligne de partage des eaux, est appliquée à l'objet [87]. La technique de segmentation utilisée se base uniquement sur l'information physique de l'image, sans interprétation sémantique du contenu.

Le masque correspondant à l'image de la figure 4.3 est montré sur la figure 4.4.a. L'application du masque d'objet à l'image segmentée permet d'obtenir l'objet de premier plan segmenté (Fig. 4.4.b), qui représente les données que nous utilisons en entrée de notre méthode pour la mise en correspondance d'objets.

(a) (b)

FIG. 4.4: Exemple de masque d'objet. (a) Le masque d'objet correspondant à l'image de la figure 4.3 et (b) l'objet de premier plan segmenté.

L'étape qui vient ensuite concerne la modélisation de l'objet segmenté extrait de la vidéo sous forme de graphe.

4.1.2 Discussion sur les résultats de la segmentation

Le positionnement dans le contexte du paradigme d'indexation grossière réduit le champ des méthodes utilisables pour la comparaison des objets et par conséquent de leurs partitions.

La réduction de l'information au niveau du bloc peut impliquer une sous-segmentation de l'image. En effet, la réduction par blocs supprime des détails de l'image et, deux zones colorimétriquement proches qui diffèrent par la texture dans l'image originale peuvent être fusionnées dans l'image sous-échantillonnée.

Par opposition, des phénomènes de sur-segmentation peuvent apparaître du fait de la réduction par blocs. En effet, un pixel dans l'image DC correspond à un bloc de 8×8 pixels de l'image en pleine résolution. Or, un bloc de 8×8 pixels dans l'image en pleine résolution peut contenir des portions de régions de couleur très différentes dans le cas où le bloc contient un contour entre différents objets. Le pixel correspondant dans l'image DC sera par conséquent un mélange de ces couleurs et sera donc très différent des pixels DC avoisinants qui correspondent à des blocs de pixels homogènes. Dans ce cas, de petites régions peuvent apparaître autour des frontières de l'objet.

D'autre part, les objets sont extraits de façon automatique, par détection de zones dont le mouvement diffère de celui de la caméra. Il faut préciser que le masque d'objet obtenu ne contient pas nécessairement exactement l'objet de premier plan. Par exemple, si l'on considère un plan fixe dans lequel un personnage reste statique, alors il est impossible pour la méthode d'extraction par analyse de mouvement de détecter le protagoniste puisque son mouvement ne diffère pas de celui du fond. Nous pouvons ainsi observer fréquemment un problème de sous-détection de l'objet ou de portions de l'objet. A l'inverse, le procédé d'extraction peut inclure dans le masque d'objet des pixels appartenant au fond. Lorsqu'un objet se déplace d'une image sur l'autre, son déplacement entraîne d'une part le recouvrement de certains blocs du fond, mais cela entraîne aussi l'apparition de nouveaux blocs du fond qui n'étaient pas présents dans l'image précédente. Dans la plupart des cas, l'estimation du mouvement des blocs découverts par rapport à l'image précédente, ne suit pas le mouvement global de la caméra, et ces blocs sont alors associés à l'objet de premier plan. Cela implique la présence de petites régions correspondant au fond de l'image en bordure de l'objet en mouvement.

Outre les deux inconvénients mentionnés ci-dessus, dans le problème de la mise en correspondance d'objets dans la vidéo, les objets naturels sont souvent articulés. Cela implique des déformations locales des régions auxquelles il faut être tolérant. Cependant, même si les caractéristiques des régions varient dans le temps, la structure du voisinage d'une région reste relativement stable. En admettant cette supposition, nous proposons dans ce manuscrit deux méthodes pour la mise en correspondance des objets dans la vidéo essentiellement basées sur la prise en compte de l'information sur la structure des objets.

4.1.3 Formalisation du problème et notations

Comme mentionné en introduction, nous ramenons la comparaison d'objets dans la vidéo à faible résolution à un problème de comparaison de graphes.

Classiquement, on peut représenter une partition par son graphe d'adjacence des régions (RAG) [57, 138].

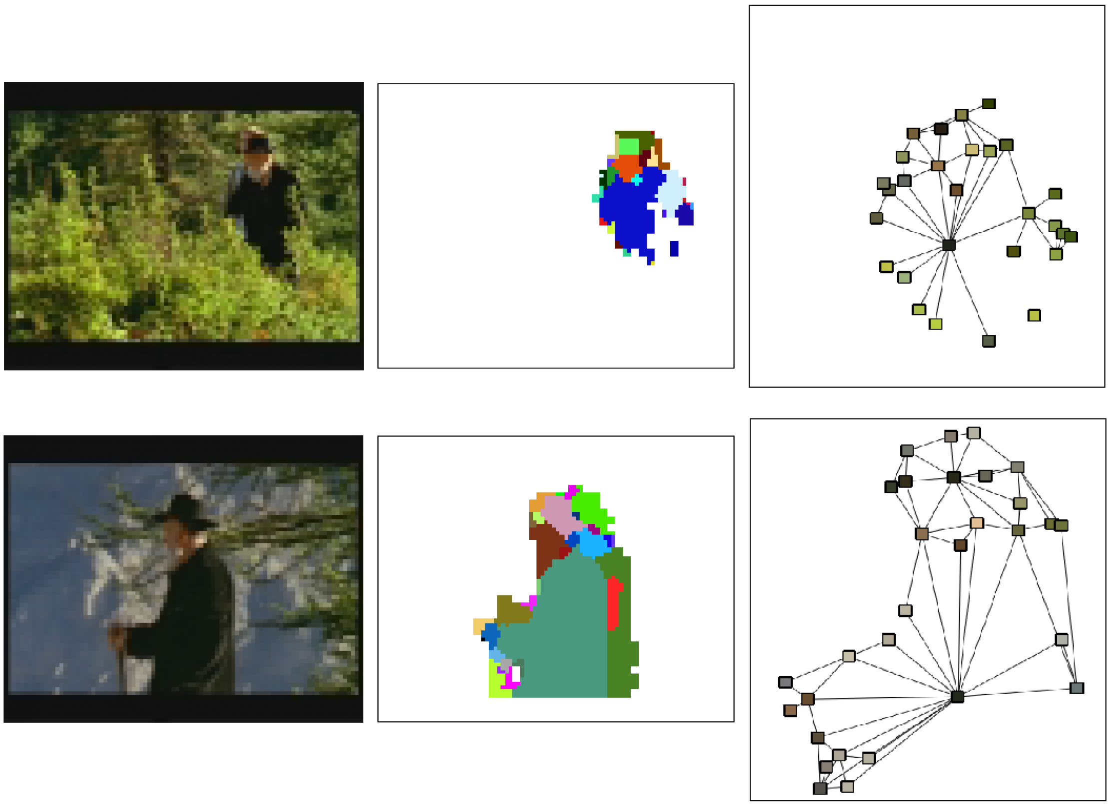
Soit une partition $\mathcal{O} = \{r_1, r_2, \cdots, r_n\}$ représentant un objet segmenté extrait de la vidéo. Le RAG associé à la partition $\mathcal{O}$ s'exprime comme suit :

Définition 47 (Graphe d'adjacence des régions (RAG))
Le graphe d'adjacence des régions $G = (V_G, E_G)$ de la partition $\mathcal{O}$ est tel que

- *chaque région r_i de la partition $\mathcal{O}$ est associée à un sommet u_i dans le graphe G*
- *il existe une arête $e = \{u_i, u_j\} \in E_G$ entre deux sommets u_i et u_j si les régions correspondantes r_i et r_j sont adjacentes dans le plan image.*

Il est possible, pour certaines applications, que l'orientation des arêtes (différenciation entre le sommet de départ et le sommet d'arrivée) joue un rôle.

Parallèlement, il existe la possibilité d'assigner des attributs aux éléments du graphe. Dans ce cas, on affecte à chacun des sommets un ensemble d'attributs correspondant à des données propres aux régions. Pour chacune des deux méthodes proposées, nous présenterons l'ensemble des attributs que nous avons retenus pour la comparaison des graphes dans les sections correspondantes.



(a) (b) (c)

FIG. 4.5: (a) Images vidéos originales, (b) les objets segmentés extraits et (c) les RAGs correspondants (28 sommets pour le RAG du haut et 34 sommets pour le RAG du bas).

Chaque objet est donc représenté par son graphe d'adjacence des régions $G(V_G, E_G)$. Par la suite, nous ne ferons pas la distinction entre la région r de l'image et le sommet u associé dans le RAG.

On rappelle que $\mathcal{N}(u)$ désigne l'ensemble des sommets voisins à u (Déf. 13). Par abus de notations, on note $\mathcal{N}_{\mathcal{O}}(r)$ l'ensemble des régions de la partition $\mathcal{O}$, adjacentes à r dans le plan image.

La figure 4.5.a montre un exemple de deux images extraites d'un flux vidéo. Le processus d'extraction et de segmentation des objets de premier plan décrit précédemment produit les partitions en fausses couleurs de la figure 4.5.b.

On remarque ici que les masques d'objet incluent des zones correspondant au fond de l'image. D'autre part, on peut observer sur cet exemple l'effet de sur-segmentation dû au lissage par blocs : dans la partition du haut, on relève la présence de nombreuses petites régions en bordure de l'objet. Ces régions de très faible surface correspondent à des blocs de l'image originale regroupant des pixels de différentes zones homogènes (ici, le manteau noir du personnage et la zone texturée verte des buissons).

Les graphes d'adjacence des régions associés aux partitions sont présentés sur la figure 4.5.c. L'objectif des méthodes proposées consiste à comparer les structures de graphes afin d'évaluer une mesure de similarité entre les objets.

Nous présentons dans les sections qui suivent les deux méthodes que nous proposons pour la comparaison d'objets basée sur la mise en correspondance de graphes.

4.2 Méthode par relaxation

La première méthode que nous proposons pour la mise en correspondance d'objets dans la vidéo à basse résolution est inspirée des techniques de relaxation probabiliste [111, 68] et plus particulièrement de la méthode pour le suivi d'objets dans la vidéo proposée par Gomila *et al.* [57]. Le diagramme de la figure 4.6 résume le fonctionnement général de la méthode. Le principe est le suivant : après la construction du RAG, l'*étape 2* consiste à supprimer les petites régions résultant d'un défaut de sur-segmentation comme présenté à la section 4.1.2.

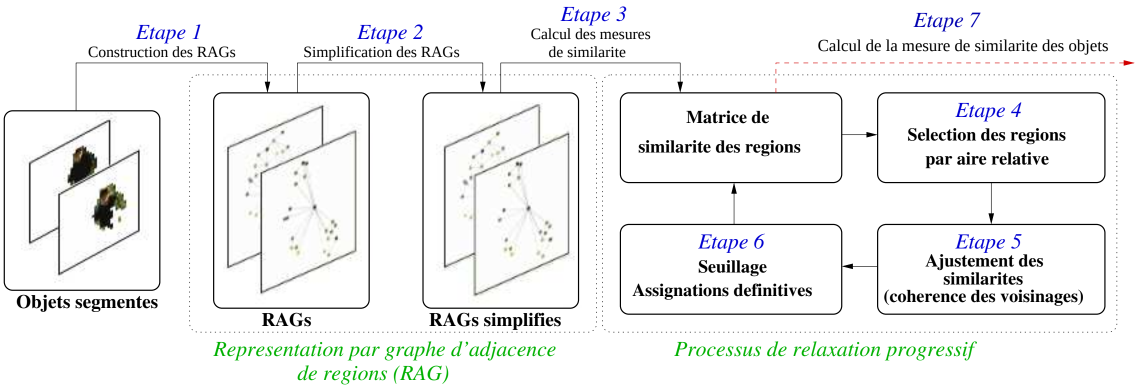


Fig. 4.6: Diagramme de la méthode par relaxation.

Le processus de mise en correspondance des RAGs se déroule ainsi : une mesure de similarité initiale entre les sommets des deux RAGs représentant les objets vidéo est calculée (*étape 3*). Cette mesure est itérativement corrigée en fonction de la contribution du voisinage local des sommets (*étape 5*). La mesure de similarité évolue à chaque étape en fonction du voisinage considéré. Durant cette étape, certaines décisions de mise en correspondance peuvent être définitives (*étape 6*). Pour minimiser le bruit dû à la sur-segmentation, nous décidons de débuter le processus itératif par les régions qui sont les plus significatives. Nous pouvons considérer que plus la surface de la région est importante, plus la région est significative. À chaque itération, de nouvelles régions seront prises en compte en fonction de leur taille. Cette stratégie (*étape 4*) renforce le résultat obtenu. À la fin du processus de mise en correspondance des régions, on calcule une mesure de similarité entre les objets basée sur la surface des objets mise en correspondance (*étape 7*).

Nous introduisons dans la section 4.2.1 la mesure de similarité entre deux régions basée sur les attributs des régions que nous avons retenus dans cette méthode. La section 4.2.2 détaille la phase de simplification du graphe d'adjacence et la méthode de mise en correspondance par relaxation est décrite en section 4.2.3.

4.2.1 Mesure de similarité entre deux régions

Habituellement, on caractérise un objet par sa couleur, sa texture et sa forme. Ce sont les critères visuels que l'on utilise le plus souvent pour différencier deux objets. Ainsi, les attributs propres aux régions associés aux sommets correspondent le plus souvent à des caractéristiques de couleur, de texture ou de forme.

Pour comparer deux régions afin d'évaluer leur similarité, nous pouvons utiliser les paramètres extrinsèques associés aux sommets du graphe. Il est ainsi possible d'évaluer la similarité de couleur, de forme ou la similarité globale entre deux régions. Dans nos travaux, on considère que si deux régions diffèrent fortement pour l'une ou l'autre des caractéristiques de régions retenues, alors elles ne sont pas similaires. On appelle cette propriété, la *propriété d'absorption*. Cela signifie que pour que deux régions soient considérées comme similaires, elles doivent être proches pour toutes leurs caractéristiques.

Les descripteurs de régions doivent être adaptés à la résolution des données. Les données que nous manipulons sont de faible résolution colorimétrique et géométrique. Par conséquent, l'utilisation de descripteurs sophistiqués (*e.g.* les descripteurs de couleur et de forme normalisés MPEG-7 [88]) sont inappropriés pour notre application. L'information sur la texture n'est pas non plus exploitable du fait du sous-échantillonnage qui atténue les textures. Ainsi, notre choix est de considérer des caractéristiques élémentaires de couleur et de forme pour décrire les régions. Nous définissons dans les paragraphes qui suivent les descripteurs de couleur et de forme que nous utilisons dans la méthode.

4.2.1.1 Descripteur de couleur

La couleur est la caractéristique la plus fréquemment utilisée pour la recherche et la mise en correspondance de données multimédia par le contenu. Parmi les approches classiques, on peut citer les histogrammes de couleurs [47, 54, 114] qui sont des outils fréquemment utilisés pour la mise en correspondance d'image et de vidéo. La définition d'une distance dans des espaces de couleur différents [57, 138] est également très usitée.

Pour notre problématique, l'histogramme de couleurs d'une région n'est pas réellement pertinent puisque les régions sont de petite surface et contiennent peu d'information. Par conséquent, nous avons choisi de considérer le descripteur colorimétrique le plus simple, à savoir la couleur moyenne d'une région. Se pose alors le problème du choix de l'espace de couleurs. Nous avons choisi l'espace RGB puisque la segmentation produit des régions homogènes dans cet espace de couleurs.

On définit la mesure de similarité couleur $\rho_c(r, r')$ entre deux régions r et r' ainsi :

$$\rho_c(r, r') = (1 - \frac{|\bar{R}(r) - \bar{R}(r')|}{255})(1 - \frac{|\bar{V}(r) - \bar{V}(r')|}{255})(1 - \frac{|\bar{B}(r) - \bar{B}(r')|}{255}) \quad (4.1)$$

Les valeurs $\bar{R}$, $\bar{V}$ et $\bar{B}$ sont comprises entre 0 et 255.

On note ici que si les deux régions sont fortement différentes sur l'un des canaux couleur, alors la mesure de similarité ρ_c sera faible. La mesure ρ_c vérifie la propriété d'absorption. Une valeur de ρ_c proche de 1 indique une forte similarité.

4.2.1.2 Descripteur de forme

Une région peut également être caractérisée par des descripteurs de forme. Ces descripteurs peuvent être classés en deux catégories : une représentation basée surface et une représentation basée contour [147]. Pour la première catégorie, les paramètres considérés sont décrits sur la surface intérieure de la région. On peut citer les moments invariants, les caractéristiques de la boîte englobante de la région comme exemples de descripteurs de ce type. La seconde approche considère la géométrie de la frontière des régions. Les descripteurs de Fourier et le descripteur de contour MPEG-7 sont des exemples de descripteurs appartenant à cette catégorie [88].

Dans les images DC, les contours sont particulièrement lissés, en raison de la réduction par blocs de l'image en pleine résolution. Ainsi, les descripteurs basés frontière sont moins appropriés pour décrire les régions dans notre contexte.

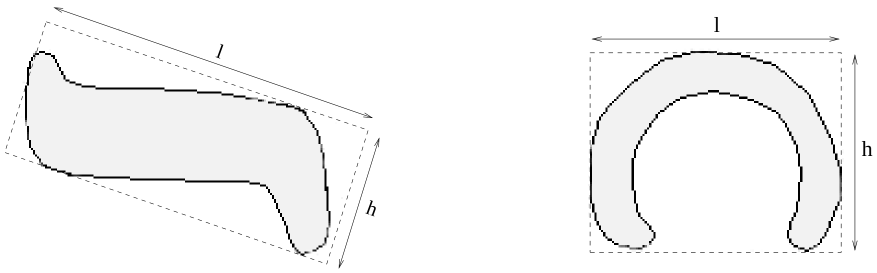
Le descripteur de forme retenu doit être invariant aux déformations usuelles telles que la rotation, la translation et le changement d'échelle. De plus, il doit être tolérant aux déformations géométriques des régions. En effet, la forme de la plupart des objets articulés change entre deux images différentes en raison des mouvements locaux au niveau des articulations (*e.g.* mouvement des membres d'un humain). Ainsi, certaines régions peuvent être déformées de manière différente des autres régions qui composent l'objet.

Un descripteur élémentaire vérifiant les propriétés précédentes est la *boîte englobante orientée* (Déf. 48) de la région. Il sera retenu comme descripteur de forme dans notre approche.

Définition 48 (Boite englobante orientée)
On appelle boîte englobante orientée d'un ensemble de pixels S le plus petit rectangle qui contient tous les éléments de S.

On appelle largeur l (resp. hauteur h) de la boîte englobante la dimension du plus grand côté (resp. plus petit côté) du rectangle.

La figure 4.7 montre deux exemples de boîtes englobantes (représentées en pointillés) de régions (en grisé).



(a) (b)

FIG. 4.7: Exemples de boîtes englobantes de régions.

Deux caractéristiques de la boîte englobante sont utilisées comme éléments du descripteur de forme :
- l'*élongation* notée $el(r)$ de la région r, correspond au ratio entre la hauteur h de la boîte englobante et sa largeur l.
- l'*eccentricité* notée $ecc(r)$ de la région, correspond au ratio entre la surface de la région et la surface de sa boîte englobante.

L'élongation et l'eccentricité sont deux informations complémentaires : dans l'exemple de la figure 4.7.a, l'élongation de la région est assez représentative de la forme, mais dans l'exemple de la figure 4.7.b, la seule information sur l'élongation ne permet pas de distinguer la région de l'exemple d'une région rectangulaire qui s'inscrit dans la même boîte englobante. L'eccentricité, qui mesure en quelque sorte le "taux d'occupation" de la région par rapport à sa boîte englobante, est une information nécessaire dans ce cas.

On note que les mesures de l'élongation et de l'eccentricité des régions sont invariantes aux transformations telles que la rotation, la translation et la transformation d'échelle et sont donc tolérantes aux déformations locales de l'objet. Dans le cas d'un changement d'échelle, l'objet dans son intégralité est soumis au même facteur d'échelle. Ainsi, outre les informations sur la boîte englobante, nous considérons l'aire relative $a(r)$ (Déf. 49) d'une région r par rapport à la surface de l'objet de premier plan auquel elle appartient.

Définition 49 (Aire relative d'une région)
On définit l'aire relative $a(r)$ d'une région r appartenant à un objet $\mathcal{O}$ comme étant le ratio entre la surface de la région r et la surface de l'objet $\mathcal{O}$. On a

$$a(r) = \frac{|r|}{|\mathcal{O}|}$$

où $|r|$ (resp. $|\mathcal{O}|$) désigne le nombre de pixels de la région r (resp. de l'objet $\mathcal{O}$).

La mesure de similarité de forme $\rho_f(r, r')$ entre deux régions r et r' est définie comme suit :

$$\rho_f(r, r') = \frac{1}{2}\Big(\frac{min(ecc(r), ecc(r'))}{max(ecc(r), ecc(r'))} + \frac{min(el(r), el(r'))}{max(el(r), el(r'))}\Big)\frac{min(a(r), a(r'))}{max(a(r), a(r'))} \quad (4.2)$$

La mesure de similarité de forme a été normalisée par la division systématique par le maximum. Les régions doivent être proches à la fois du point de vue local (caractéristiques de la boîte englobante) et global (aire relative) pour être considérées comme similaires. On retrouve ici la propriété d'absorption.

4.2.1.3 Mesure de similarité

Nous avons introduit une nouvelle mesure de similarité globale entre deux régions basée sur les mesures de similarité de couleur ρ_c (Form. (4.1)) et de forme ρ_f (Form. (4.2)) présentées dans les paragraphes précédents.

La mesure de similarité globale $\rho(r, r')$ entre deux régions r et r' est définie comme suit :

$$\rho(r, r') = \rho_c(r, r').\rho_f(r, r') \quad (4.3)$$

La valeur $\rho(r, r')$ appartient à l'intervalle $[0, 1]$. Plus la mesure de similarité est proche de 1, plus les régions sont similaires. Quand les deux régions diffèrent pour l'un des deux descripteurs, elles ne sont pas similaires. La propriété d'absorption est induite par le schéma multiplicatif de l'équation (4.3).

4.2.2 Simplification des graphe d'adjacence des régions

Nous avons mentionné dans la section 4.1.2, la présence de nombreuses petites régions due au bruit engendré par le sous-échantillonage spatial (problèmes de sur-segmentation) et au processus automatique d'extraction des objets (présence de pixels appartenant au fond). Pour éviter ces inconvénients et améliorer la mise en correspondance des objets, nous avons introduit une étape préliminaire de simplification des RAGs (*étape 2* de la figure 4.6). Dans cette phase, nous fusionnons les régions (et par conséquent les sommets des RAGs) de petite taille qui résultent de la sur-segmentation.

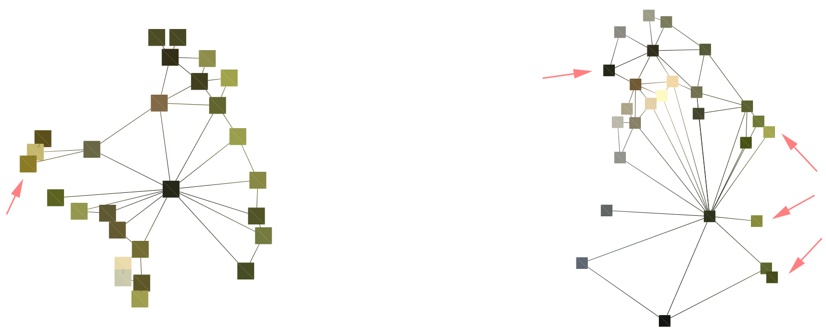
Nous distinguons trois cas pour la fusion de régions : (i) les régions très petites sont systématiquement fusionnées avec l'un de leurs voisins, (ii) les grande régions sont conservées et (iii) les régions de taille moyenne qui résultent de la sur-segmentation sont fusionnées seulement s'il existe une région dans leur voisinage de couleur proche. Ici, la couleur d'une région r' est considérée proche de la couleur d'une région r si la mesure de similarité couleur $\rho_c(r, r')$ (Form. (4.1)) est en-dessous d'un certain seuil $\delta_c \in [0, 1]$.

Formellement, soient $\epsilon_1 < \epsilon_2$ deux seuils pour l'aire relative compris entre 0 et 1. Nous considérons chaque région r selon son aire relative $a(r)$ (Déf. 49). Soit r' la région voisine de r la plus proche colorimétriquement ($i.e.$ $r' = max_{r_i \in \mathcal{N}(r)} \rho_c(r_i, r)$). Les lois de fusion des régions sont les suivantes :

(i) si $a(r) < \epsilon_1$, alors la région r est supprimée par fusion de r avec r',
(ii) si $a(r) \geq \epsilon_2$, alors r est une région pertinente et on ne la supprime pas,
(iii) si $\epsilon_1 \leq a(r) < \epsilon_2$, si $\rho_c(r, r') > \delta_c$, on fusionne r et r' car r est considérée comme une région résultant de la sur-segmentation du fait qu'elles présentent tout de même une similitude colorimétrique.

Cette étape a pour but de supprimer les petites régions résultant souvent du bruit de segmentation.

La figure 4.8 montre un exemple de la simplification des RAGs de l'exemple de la figure 4.5. Les flèches rouges indiquent les régions qui ont été fusionnées.



(a) 27 sommets (b) 27 sommets

FIG. 4.8: Simplification des graphes de la figure 4.5. Les régions fusionnées sont désignées par les flèches rouges.

L'étape qui suit consiste à comparer les objets via une mise en correspondance des RAGs simplifiés.

4.2.3 Mise en correspondance des objets basée sur les graphes

Le processus de mise en correspondance est basé sur un algorithme itératif de relaxation qui raffine progressivement la similarité entre les régions en prenant en compte la similarité de leur voisinage. Dans une telle approche, l'évaluation de la similarité entre les régions est obtenue par combinaison de la similarité initiale (basée sur les descripteurs) et des ajustements successifs (renforcement ou pénalisation) en fonction de l'information du contexte local (*i.e.* les régions du voisinage).

4.2.3.1 Présentation de l'algorithme

Les régions de grande surface forment les parties les plus significatives des objets, par conséquent, la mesure de similarité entre ces régions est la plus pertinente en terme d'importance et de précision. L'évaluation de la similarité entre les régions d'aire importante ne doit pas être faussée par les petites régions voisines. En effet, les régions de petite surface peuvent être le résultat des erreurs de segmentation ou des imprécisions du masque de l'objet de premier plan. Dans ce cas, elles pourraient pénaliser la mesure de similarité des larges régions avoisinantes. Par conséquent, nous proposons une *mise en correspondance ordonnée* des régions dans les RAGs.

Soient deux RAGs simplifiés G et G' (voir section 4.2.2 et *étape 2* Fig. 4.6) correspondants aux objets à comparer $\mathcal{O}$ et $\mathcal{O}'$ respectivement. Nous calculons dans un premier temps les mesures de similarité initiales pour chaque paire de régions de G et G' (*étape 3* Fig. 4.6). Ensuite, l'algorithme procède itérativement. Soit ν_i un seuil inversement proportionnel à i, le numéro de l'itération. À chaque itération i nous considérons l'objet réduit $\mathcal{O}_i$ (resp. $\mathcal{O}'_i$) (Déf. 50) qui est composé des régions de $\mathcal{O}$ (resp. $\mathcal{O}'$) d'aire relative supérieure au seuil ν_i (*étape 4* Fig. 4.6).

Définition 50 (Objet réduit $\mathcal{O}_i$)
On appelle objet réduit $\mathcal{O}_i$ de l'objet $\mathcal{O}$, à l'itération i, la partition telle que :
$r \in \mathcal{O}_i$ *si* $a(r) > \nu_i$, *avec* $r \in \mathcal{O}$.

Ainsi, on débute avec les plus grandes régions et on introduit des régions de plus en plus petites à chaque itération du processus de mise en correspondance.

À l'itération i, pour chaque paire de régions (r, r') des objets réduits $\mathcal{O}_i$ et $\mathcal{O}'_i$ respectivement, on calcule la mesure de similarité ajustée $\gamma_i(r, r')$ entre r et r'. Cette mesure de similarité ajustée dépend de la mesure de similarité initiale $\gamma_0(r, r') = \rho(r, r')$ (basée sur les descripteurs, voir la formule (4.3)) et de la similarité des voisinages à l'itération $i - 1$ (*étape 5* Fig. 4.6). On fixe deux seuils de similarité $\delta_{min} < \delta_{max}$. Si la valeur $\gamma_i(r, r')$ est inférieure à δ_{min}, on considère que les régions ne se correspondront jamais et nous fixons la valeur $\gamma_i(r, r')$ à 0. À l'inverse, si la valeur $\gamma_i(r, r')$ est supérieure à δ_{max} et ne présente pas d'ambiguïté avec les autres régions, on décide que ces régions sont définitivement mises en correspondance et on assigne $\gamma_i(r, r')$ à 1 (*étape 6* Fig. 4.6).

```
Données : deux partitions 𝒪 et 𝒪',
          le nombre d'itérations t

initialiserMesuresDeSimilarites (𝒪, 𝒪') // Etape 3
Pour i De 1 A t Faire
    (𝒪_i, 𝒪'_i) := selectionneRegions(𝒪, 𝒪', i) // Etape 4
    Pour (r, r') ∈ 𝒪_i × 𝒪'_i
        Si (r, r') est marquée Alors
            γ_i(r, r') := γ_{i−1}(r, r')
        Sinon
            γ_i(r, r') := compensationVoisinage(r, r', 𝒪_i, 𝒪'_i) // Etape 5
            γ_i(r, r') := seuillage(γ_i(r, r')) // Etape 6
        FinSi
    FinPour
    miseEnCorrespondance(𝒪_i, 𝒪'_i, 𝒪, 𝒪') // Etape 6
FinPour
```

Algorithme 4.1: Procédure de mise en correspondance d'objets

Dans le but d'accélérer la convergence de l'algorithme de relaxation, on fixe le nombre d'itérations à t tours de boucle. Les expérimentations montrent que faire croître t au-delà de 10 n'améliore pas le résultat de la mise en correspondance et même crée des dégradations du fait du traitement des petites régions. L'algorithme 4.1 décrit la procédure de mise en correspondance. Les étapes correspondant à la figure 4.6 sont indiquées en italique.

L'étape d'initialisation (`initialiserMesuresDeSimilarites`) fixe, pour chaque couple $(r, r') \in \mathcal{O} \times \mathcal{O}'$ la valeur initiale $\gamma_0(r, r') = \rho(r, r')$ (Form. (4.3)).

Une paire de régions est dite `marquée` lorsque sa mesure de similarité a été assignée à 1 ou 0. La fonction de compensation du voisinage est décrite dans la section 4.2.3.2. Le détail des procédures `seuillage` et `miseEnCorrespondance` est donné dans la section 4.2.3.3.

4.2.3.2 Ajustement de la similarité : contribution du voisinage

Nous allons dans cette section présenter la *fonction de compensation du voisinage* ϕ_i pour la correction de la similarité entre les régions. Le rôle de ϕ_i est d'ajuster la similarité entre deux régions en fonction de la similarité de leurs voisinages notée κ_i. Autrement dit, plus les voisinages sont similaires, plus il est probable que les régions se correspondent et inversement.

La fonction $\phi_i(r, r')$ est utilisée pour faire croître ou décroître la similarité entre deux régions r et r' en fonction de la cohérence de leurs voisinages restreints $\mathcal{N}_{\mathcal{O}_i}(r)$ et $\mathcal{N}_{\mathcal{O}'_i}(r')$ respectivement.

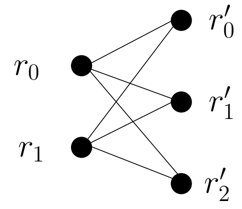
En conséquence, la mesure de similarité $\gamma_i(r, r')$ entre les régions évolue selon la fonction de compensation $\phi_i(r, r')$ basée sur la mesure de similarité des voisinages à l'étape $i - 1$:

$$\gamma_i(r, r') = \gamma_0(r, r') + \phi_i(\mathcal{N}_{\mathcal{O}_i}(r), \mathcal{N}_{\mathcal{O}'_i}(r')) \tag{4.4}$$

Nous présentons dans un premier temps la mesure de similarité de voisinage, dénotée $\kappa_i(r, r')$ basée sur le meilleur couplage que l'on peut établir entre les voisins de r et les voisins de r'. On construit le graphe biparti complet composé des voisinages $\mathcal{N}_{\mathcal{O}_i}(r)$ de r d'une part et $\mathcal{N}_{\mathcal{O}'_i}(r')$ de r' d'autre part. Les arêtes sont pondérées par la mesure de similarité $\gamma_{i-1}(r, r')$ entre les régions à l'étape $i - 1$. Nous établissons ensuite le couplage de poids maximum $\mathcal{C}$ sur le graphe biparti complet [61] qui représente la meilleure association des régions voisines en terme de similarité globale. Cet algorithme a été utilisé par Shokoufandeh *et al.* dans le contexte de la mise en correspondance d'objets [118]. Dans leurs travaux, les objets sont définis à plusieurs niveaux de détail (hiérarchie de partitions). Le couplage de poids maximal est calculé sur les graphes à chaque niveau d'abstraction.

La figure 4.9 montre un exemple de couplage de poids maximum. Le graphe de voisinages biparti tel que $\mathcal{N}_{\mathcal{O}i}(r) = \{r_0, r_1\}$ et $\mathcal{N}_{\mathcal{O}'i}(r') = \{r'_0, r'_1, r'_2\}$ est représenté sur la figure 4.9.a et la matrice de mesures de similarité $\gamma_{i-1}(r_l, r'_k)$ correspondante sur la figure 4.9.b. La figure 4.9.c illustre le couplage de poids maximum. Les arêtes en trait continu représentent les couples retenus (r_0, r'_0) et (r_1, r'_1).

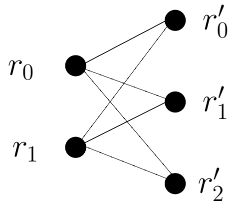
On note que les régions ne sont pas nécessairement couplées avec le meilleur candidat. Par exemple, on a $\gamma_{i-1}(r_1, r'_0) > \gamma_{i-1}(r_1, r'_1)$, cependant la région r_1 a été associée à la région r'_1. L'objectif de la recherche du couplage de poids maximum est de maximiser la somme (ou la moyenne) des poids des arêtes retenues pour le couplage. Le poids moyen noté $\bar{\mathcal{C}}$ du couplage $\mathcal{C}$, correspond à la moyenne des poids des arêtes du couplage $\mathcal{C}$. Ainsi, coupler r_0 avec r'_0 et r_1 avec r'_1 (le poids moyen du couplage est $\bar{\mathcal{C}} = 0.45$) induit une plus forte similarité globale que le couplage



(a)

γ_{i-1}	r'_0	r'_1	r'_2
r_0	0.5	0.2	0.1
r_1	0.6	0.4	0.2

(b)



(c)

FIG. 4.9: Couplage de poids maximum

(r_0, r'_1), (r_1, r'_0) de poids moyen $\bar{\mathcal{C}} = 0.4$.

Dans la définition de la similarité entre les voisinages de r et r', on propose de prendre en considération la différence de cardinalité d'une part et le poids moyen $\bar{\mathcal{C}}$ du meilleur couplage d'autre part. Ainsi, la mesure de similarité des voisinages tiendra compte à la fois de la similarité sur la structure des voisinages, et de la similarité entre les régions de ces voisinages dans l'espace des descripteurs. La mesure de similarité de voisinage $\kappa_i(r, r')$ est donnée par la formule :

$$\kappa_i(r, r') = \frac{\min(|\mathcal{N}_{\mathcal{O}_i}(r)|, |\mathcal{N}_{\mathcal{O}'_i}(r')|)}{\max(|\mathcal{N}_{\mathcal{O}_i}(r)|, |\mathcal{N}_{\mathcal{O}'_i}(r')|)}.\bar{\mathcal{C}} \tag{4.5}$$

La valeur κ décroît si les structures des voisinages diffèrent fortement de cardinalité ou si les régions ne sont pas similaires dans l'espace des descripteurs.

Pour assurer une influence significative de la similarité ou dissimilarité du voisinage dans le processus d'ajustement, on considère trois cas d'ajustement selon la valeur de $\kappa_i(r, r')$: (i) on fait croître la similarité entre les régions si leurs voisinages se correspondent fortement, (ii) on fait décroître la similarité si les voisinages sont très différents et (iii) les voisinages ne sont pas assez similaires ou dissimilaires pour prendre une décision, et on ne modifie pas la mesure de similarité entre les régions.

Soient α_1 et $\alpha_2 \in [0, 1]$ deux seuils pour différencier ces trois cas. La valeur de $\phi_i(\mathcal{N}_{\mathcal{O}_i}, \mathcal{N}_{\mathcal{O}'_i})$ est définie comme suit :

$$\phi_i(\mathcal{N}_{\mathcal{O}_i}, \mathcal{N}_{\mathcal{O}'_i}) = \begin{cases} \frac{1}{2\alpha_1}\kappa_i(r, r') - \frac{1}{2} & \text{, si } \kappa_i(r, r') \leq \alpha_1 \\ \frac{1}{2(1-\alpha_2)}\kappa_i(r, r') - \frac{\alpha_2}{2(1-\alpha_2)} & \text{, si } \alpha_1 < \kappa_i(r, r') \leq \alpha_2 \\ 0 & \text{, sinon} \end{cases} \tag{4.6}$$

Dans la pratique, on coupe l'intervalle des valeurs possibles pour κ en trois portions égales. Cela signifie que $\alpha_1 = \frac{1}{3}$ et $\alpha_2 = \frac{2}{3}$.

4.2.3.3 Mise en correspondance des régions

Lors du processus de mise en correspondance, la mesure de similarité d'une paire de régions peut atteindre une valeur très proche de 1 (similarité maximale) ou au contraire une valeur très proche de 0 (similarité minimale). Dans le premier cas, on peut considérer que la mesure de similarité entre les régions est suffisamment élevée pour prendre la décision de mettre en correspondance les deux régions. Dans le deuxième cas, on suppose que les régions sont trop peu similaires pour se correspondre et on décide de rabaisser la mesure de similarité entre ces régions à la valeur minimale possible. Ainsi, le processus global de mise en correspondance est accéléré puisque la mesure de similarité entre certaines régions est définitivement fixée.

La mise en correspondance des RAGs est accélérée par l'introduction de deux étapes. Premièrement, on fixe une similarité nulle entre les régions qui ne se correspondront probablement jamais (procédure `seuillage` de l'*étape 6* de l'algorithme global). Deuxièmement, on considère que deux régions se correspondent si leur mesure de similarité est suffisamment élevée et s'il n'y a pas d'ambiguïté avec d'autres régions (*i.e.* s'il n'existe aucun candidat potentiel pour la mise en correspondance avec l'une ou l'autre région). Cette étape correspond à la procédure `miseEnCorrespondance` de l'*étape 6* de l'algorithme global.

Soit δ un seuil de mise en correspondance, et soit β le seuil d'ambiguïté. Les conditions nécessaires à une mise en correspondance définitive dénotée par $r \approx r'$ entre deux régions r et r' sont définies comme suit :

$$r \approx r' \iff \begin{cases} \gamma(r,r') > \theta \\ \gamma(r,r') - \gamma(r,r'_j) > \beta, \quad \forall r'_j \neq r' \in \mathcal{O}' \\ \gamma(r,r') - \gamma(r_i,r') > \beta, \quad \forall r_i \neq r \in \mathcal{O} \end{cases} \tag{4.7}$$

La première condition dans (4.7) vérifie une forte valeur de la mesure de similarité entre les régions considérées. Les deuxième et troisième conditions assurent qu'il n'y a pas d'autres candidats potentiels pour la mise en correspondance, c'est à dire qu'il n'y a pas "d'ambiguïté". En pratique, on fixe $\beta = 0.08$ et $\theta = 0.8$.

La limitation des mises en correspondance par ces conditions (4.7) ont pour objectif d'éviter l'abondance de fausses assignations. En effet, comme le processus est itératif, la mesure de similarité γ évolue au fur et à mesure de la prise en compte du voisinage. Un couplage qui était le meilleur à une étape i peut changer à l'étape $i+1$. C'est pourquoi, nous ne prenons de décision définitive que lorsque les possibilités d'autres candidats sont écartées.

Si une paire de région $\{r, r'\}$ vérifie $r \approx r'$ (Form. (4.7)), une mise en correspondance définitive peut être effectuée. Alors, pour tout $j \geq i$ on a :

$$\begin{aligned} \gamma_j(r,r') &= 1 \\ \forall r_k \in \mathcal{O}, r_k \neq r, \quad \gamma_j(r_k,r') &= 0 \\ \forall r'_l \in \mathcal{O}', r'_l \neq r', \quad \gamma_j(r,r'_l) &= 0 \end{aligned}$$

Le processus de mise en correspondance décrit ci-dessus est effectué par la procédure `miseEnCorrespondance` de l'algorithme global 4.1 et il correspond à l'*étape 6* du diagramme de la méthode de la figure 4.6.

À l'issue de ce processus de mise en correspondance, on dispose de deux ensembles R et R' de régions de $\mathcal{O}$ et $\mathcal{O}'$ respectivement qui ont été mutuellement mises en correspondance. L'objectif maintenant, est d'évaluer la mesure de similarité entre les objets $\mathcal{O}$ et $\mathcal{O}'$ en fonction des appariements qui ont été effectués.

4.2.3.4 Mesure de similarité des objets

La mesure de similarité entre deux objets que nous avons introduit correspond à la moyenne des aires relatives des régions mises en correspondance. À la fin du processus de mise en correspondance (*étape 7* de la figure 4.6), la mesure de similarité $\theta(\mathcal{O}, \mathcal{O}')$ entre $\mathcal{O}$ et $\mathcal{O}'$ est calculée comme suit :

$$\theta(\mathcal{O}, \mathcal{O}') = \frac{1}{2}(\sum_{r \in R} a(r) + \sum_{r' \in R'} a(r')) \tag{4.8}$$

On rappelle que $a(r)$ représente l'aire relative d'une région r par rapport à la surface de l'objet (Déf. 49).

La mesure de similarité θ évalue la proportion moyenne des aires des objets qui ont été mises en correspondance. Cela signifie que l'on calcule dans un premier temps les aires relatives totales qu'occupent les régions des ensembles R et R' par rapport à la surface des objets $\mathcal{O}$ et $\mathcal{O}'$ respectivement. La mesure de similarité correspond alors à la moyenne de ces deux valeurs.

Complexité À chaque itération i de la boucle principale, on considère chaque couple $(r, r') \in \mathcal{O}_i \times \mathcal{O}'_i$. Soient $\bar{V}$ et $\bar{V'}$ le cardinal moyen de $\mathcal{O}_i$ et $\mathcal{O}'_i$ au cours des itérations $i \in [1, t]$. La boucle interne est exécutée $\bar{V}\bar{V'}$ fois. Pour chaque exécution de la boucle interne, on considère les voisinages de r et r'. Soient $\bar{v}$ et $\bar{v'}$ le nombre moyen de voisins de r et r' respectivement. Le calcul de la contribution du voisinage (Sec. 4.2.3.2) utilise l'algorithme de couplage de poids maximal appliqué sur le graphe biparti complet des voisinages. Cet algorithme nécessite $(\bar{v} + \bar{v'})^{\frac{5}{2}}$ opérations [61]. Ainsi, la complexité moyenne de l'algorithme global est $\Theta(t\bar{V}\bar{V'}(\bar{v} + \bar{v'})^{\frac{5}{2}})$, avec t un nombre fixé d'itérations.

Le meilleur cas de complexité est alors en temps constant $(\Omega(t2^{\frac{5}{2}})$, et le pire des cas est $O(tnm[(n-1)+(m-1)]^{\frac{5}{2}}$, avec n et m, le nombre de régions de $\mathcal{O}$ et $\mathcal{O}'$ respectivement. On note que le cas le pire n'est pas réaliste étant donné qu'il correpond à un graphe d'adjacence des régions complet. Ce qui est impossible pour des cartes de segmentations dont le nombre de régions dépasse 3.

4.2.4 Conclusion

Nous venons de définir dans cette section une première méthode pour la mise en correspondance d'objets dans la vidéo à faible résolution. La méthode est basée sur un processus de relaxation probabiliste qui consiste à utiliser l'information sur le voisinage local des régions pour compenser l'imprécision des descripteurs des régions.

Nous avons introduit une nouvelle mesure de similarité entre les régions adaptée aux données et une technique de simplification des graphes d'adjacences des régions pour éliminer les régions résultant du bruit de segmentation ou de l'imprécision du masque de l'objet.

Une autre contribution concerne la mise en correspondance progressive des regions selon leur aire relative. Cette nouvelle approche, combinée à un processus de seuillage, permet à l'algorithme de privilégier les régions importantes des objets.

La section qui suit porte sur la présentation d'une autre méthode que nous avons développée pour la même problématique. Cette seconde approche est une adaptation de la méthode heuristique décrite dans le chapitre 2.

4.3 Méthode heuristique

Nous présentons dans cette section la deuxième méthode que nous avons élaborée. Cette méthode adapte l'heuristique pour la comparaison de graphes décrite dans le chapitre 2. Une vue générale de la méthode proposée est donnée sur le diagramme de la figure 4.10. Dans un premier temps, nous procédons à la construction d'un DAG à partir du RAG en orientant les arêtes afin d'obtenir une représentation "hiérarchique" de chaque objet (*étape 2* de la figure 4.10). Les DAGs sont ensuite comparés à l'aide d'une adaptation de la version 1 de l'heuristique (voir section 2.1.2.2) dont nous rappelons le principe. Le processus de mise en correspondance des DAGs s'effectue en deux étapes. La première étape consiste à calculer un ensemble de descripteurs structurels sur les sommets des DAGs (*étape 3* de la figure 4.10) puis nous classifions les sommets en familles (*étape 4*). Nous utilisons ensuite l'étiquetage obtenu pour la construction progressive de motifs quasi-similaires. La propagation (*étape 5*) est basée sur l'hypothèse que deux sommets structurellement similaires (appartenant à une même famille) correspondent à la même région dans l'objet si la sous-structure (*i.e.* les sommets fils) de ces sommets se ressemblent.

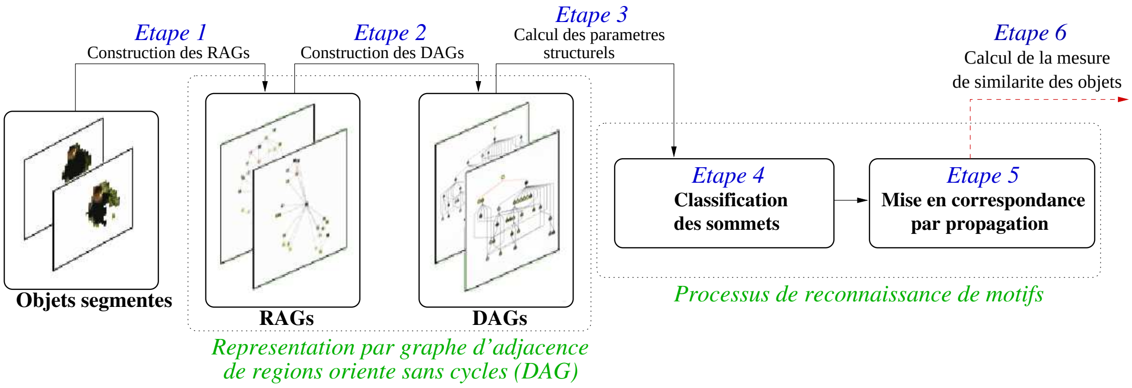


FIG. 4.10: Diagramme de l'approche heuristique.

Nous proposons dans un premier temps une technique pour orienter les arêtes des RAGs en vue d'introduire une notion de hiérarchie entre les régions. En effet, la méthode heuristique est essentiellement basée sur les caractéristiques structurelles des DAGs. Or, nous disposons d'informations propres aux régions, comme la surface, la couleur ou encore la forme d'une région. Nous avons choisi d'exploiter la surface des régions en introduisant une hiérarchie sur les sommets traduisant l'importance d'une région dans un objet.

Le deuxième aspect de l'adaptation de la méthode générique concerne l'introduction de l'information colorimétrique sur les régions. Le processus de construction de motifs se base sur les paramètres structurels des sommets et deux régions de couleurs très différentes peuvent être mises en correspondance si elles se correspondent du point de vue structurel. Ainsi, nous avons introduit dans la procédure de reconnaissance de motifs, une notion de *tolérance adaptative* aux déformations structurelles qui prend en compte l'information colorimétrique des régions.

Dans la section 4.3.1, nous décrivons la technique de construction des DAGs à partir des RAGs représentant les objets. Nous détaillons ensuite la méthode pour la classification des sommets en familles dans la section 4.3.2 et enfin, le processus de construction de motifs quasi-similaires est décrit dans la section 2.1.2.

4.3.1 Construction d'un DAG par objet

Dans le même esprit que pour la méthode par relaxation probabiliste (Sec. 4.2), nous avons décidé de privilégier les régions d'aire importante. Le RAG est donc transformé en DAG (Déf. 17) en orientant les arêtes des régions d'aire la plus élevée vers les régions voisines plus petites.

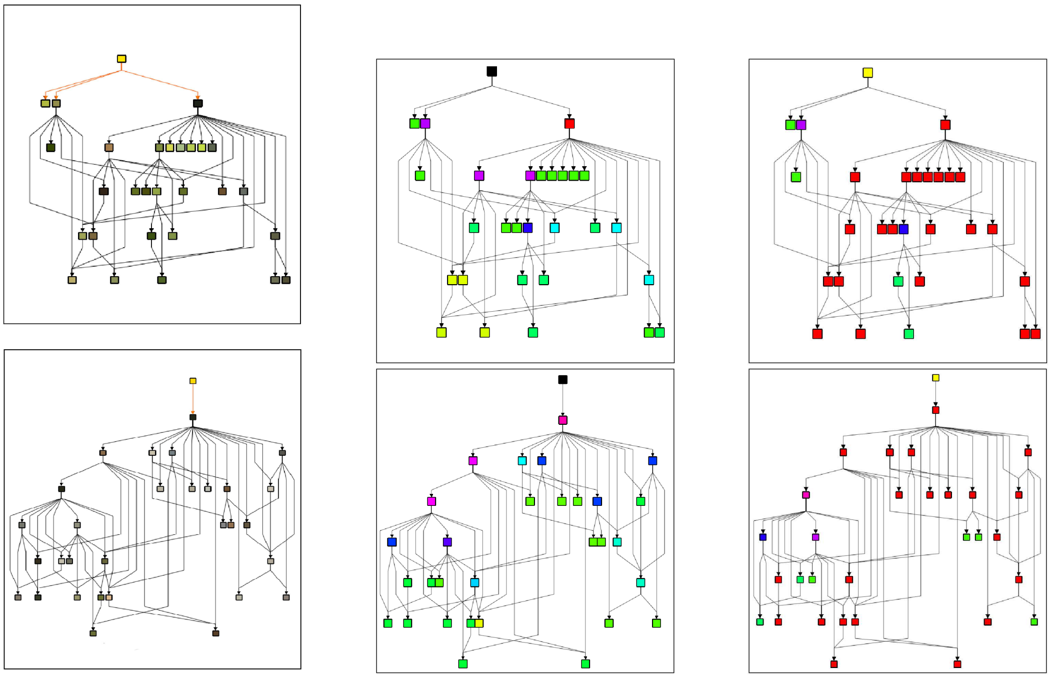
Les expériences ont montré que la considération d'autres orientations n'améliore pas la mise en correspondance (ordre sur les région de plus proche couleur, de plus proche forme, arbre recouvrant de coût minimum).

La construction du DAG $D = (V_D, E_D)$ à partir du RAG $G = (V_G, E_G)$ s'effectue comme suit. On a $V_D = E_D$ et $E_D = E_G$ où les arêtes de E_G sont des arcs (arêtes orientés). Dans un premier temps, chaque composante connexe du RAG (Déf. 18) est associée à une composante connexe du DAG. Les sommets entrants des composantes connexes du DAG représentent les régions qui ont une aire supérieure à celle de tous leurs voisins. Soit $e = \{u, u'\}$ une arête de G. L'arc correspondant e dans D est orientée de u vers u' (*i.e.* $e = (u, u')$) si $a(u) > a(u')$, avec $a(u)$ l'aire relative de la région r associée au sommet u (Déf. 49).

Afin de n'avoir qu'un seul DAG connexe pour chaque objet, nous avons choisi d'ajouter un sommet fictif u_{racine} comme racine du DAG D. On ajoute une arête de la racine vers chaque sommet de D ayant un degré entrant nul. Ainsi, les successeurs de la racine u_{racine} sont les régions d'aire relative élevée et les feuilles de ce DAG correspondent aux régions de plus petite surface. On note que fréquemment, plus un sommet est proche de la racine, plus son degré est élevé, en raison de la surface élevée de la région associée. Désormais, il existe un chemin (Déf. 15) depuis le sommet fictif vers chacun des sommets du DAG.

Sur la figure 4.11.a, on montre les DAGs construits à partir des RAGs de l'exemple de la figure 4.5. L'objet associé au DAG de gauche est constitué de deux composantes connexes. Ainsi, le sommet fictif est relié aux sommets correspondant à la région la plus grosse de chaque composante (deux arc). Le troisième arc connecte le sommet fictif à un sommet représentant une grosse région dont les voisines sont toutes plus petites (le sommet a un degré entrant nul).

Nous disposons désormais des structures de graphes orientés pour la mise en correspondance. La méthode proposée se déroule en deux temps. Premièrement, nous calculons un ensemble de mesures structurelles que l'on associe à chaque sommet. Ces métriques nous servent alors à classifier les sommets par ressemblance



(a) (b) (c)

FIG. 4.11: (a) DAGs associés aux RAGs de la figure 4.5. (b) Classification des sommets par similarité structurelle. (c) Mise en correspondance des sommets par propagation.

structurelle. La deuxième phase de la méthode concerne la construction des motifs quasi-similaires dans les DAGs, basée sur l'étiquetage obtenu lors de la phase de classification.

4.3.2 La classification des sommets

Dans cette section, nous décrivons l'ensemble des métriques intrinsèques que nous utilisons dans cette application pour nous aider à détecter les parties quasi-similaires entre les DAGs.

4.3.2.1 Les métriques

Nous avons retenu comme caractéristiques structurelles de chaque sommet u, les trois métriques suivantes :

- le degré du sommet, dénoté par $\delta(u)$
- le nombre de sommets du sous-DAG enraciné en u, dénoté par $\nu(u)$
- le nombre de Strahler du sommet, dénoté par $\sigma(u)$ (Déf. 29)

Le degré $\delta(u)$ traduit la caractéristique structurelle locale du sommet. Si la région r associée au sommet u est adjacente à de nombreuses régions, alors le degré de u est élevé. Le nombre de sommets du sous-DAG $\nu(u)$ capture le nombre de régions qui ne sont pas directement adjacentes à r, mais qui peuvent être atteintes depuis r suivant une séquence de régions adjacentes, en respectant l'orientation du DAG.

Enfin, le nombre de Strahler $\sigma(u)$ est utilisé pour mesurer le taux de ramification du sous-DAG enraciné en u. Dans un certain sens, le nombre de Strahler évalue la "centralité" d'un sommet dans la structure.

On note que δ, ν et σ n'appartiennent pas au même intervalle de valeurs. Nous normalisons les valeurs de chaque métrique indépendamment, de manière à ce qu'elles appartiennent à l'intervalle $[0,1]$. Nous calculons ainsi, pour chaque sommet u, les valeurs $\tilde{\delta}(u)$, $\tilde{\nu}(u)$ et $\tilde{\sigma}(u)$ comme indiqué dans la section 2.1.1.1.

4.3.2.2 Construction des familles de sommets ϵ-similaires

Le processus de construction de familles de sommets ϵ-similaires est décrit dans la section 2.1.1.2.

La formule (2.2) permettant de déterminer si deux sommets u et v sont ϵ-similaires devient, avec les trois métriques mentionnées ci-dessus :

$$(\tilde{\delta}(u) - \tilde{\delta}(v))^2 + (\tilde{\nu}(u) - \tilde{\nu}(v))^2 + (\tilde{\sigma}(u) - \tilde{\sigma}(v))^2 < \epsilon \tag{4.9}$$

À l'issu du processus, on obtient un étiquetage λ des sommets de D et D' en L familles de sommets ϵ-similaires.

La figure 4.11.b montre le résultat de la classification des sommets par similarité structurelle des DAGs de la figure 4.11.a. Une même couleur est utilisée pour une même valeur d'étiquette.

4.3.3 La construction des motifs quasi-similaires

Dans cette étape, on identifie des motifs quasi-similaires par propagation. Le détail de la méthode générique est donné dans la section 2.1.2. Nous utilisons dans cette application la version 1 de l'heuristique.

Dans l'algorithme générique, seule l'information structurelle (traduite par l'étiquetage en famille) est exploitée pour la construction de motifs similaires. Dans notre cas, nous ne pouvons pas nous fier uniquement aux descripteurs structurels, car deux objets ayant des mêmes caractéristiques structurelles peuvent être différents. Nous proposons d'introduire un paramètre extrinsèque (la couleur moyenne des régions) afin de renforcer la première supposition donnée par la similarité structurelle des sommets. La couleur moyenne d'une région r associée à un sommet u est notée $(\bar{R}(r), \bar{V}(r), \bar{B}(r))^T$, où $\bar{R}(r)$, $\bar{V}(r)$ et $\bar{B}(r)$ correspondent à la valeur moyenne sur les canaux rouge, vert et bleu respectivement de la région r.

L'information sur la couleur des régions est utilisée comme paramètre de contrôle à la *tolérance adaptative* aux déformations. C'est à dire que plus les régions représentées par les sommets s et s' que l'on compare sont proches en terme de couleur (distance euclidienne), plus le processus de propagation est tolérant aux changements des sous-structures. Nous intégrons à la méthode générique un ajustement de la tolérance aux différences qu'il existe entre les étiquette des ensembles $C(s)$ et

$C(s')$ de successeurs de s et s' respectivement, par une mesure de similarité couleur $\rho'_c(s, s')$ définie comme suit :

$$\rho'_c(s, s') = 1 - \sqrt{\sum_{C \in \{R,V,B\}} (\bar{C}_{R(s)} - \bar{C}_{R(s')})^2} \tag{4.10}$$

La notion de τ-similarité de la définition 38 est alors modifiée en notion de τ-*similarité adaptative* à la couleur. Ainsi, deux ensembles de sommets C et C' sont définis comme τ-similaires s'ils vérifient :

$$D_1(C, C') \leq \tau.\rho'_c \tag{4.11}$$

avec D_1 la mesure de dissimilarité d'étiquetage (Déf. 39).

Ici, l'algorithme principal de reconnaissance de motifs 2.3 est inchangé. La procédure récursive de propagation 2.2 est la même à l'exception de la ligne 6 de test de τ-similarité, auquel il faut ajouter le contrôleur de tolérance comme indiqué dans l'équation (4.11).

Lors de la propagation de motifs, l'ordre dans lequel sont visités les sommets a une influence. Dans cette application, nous disposons d'une information sur l'importance des régions : l'aire relative (Déf. 49). Afin de favoriser la comparaison des paires qui sont les plus proches en terme de surface, nous avons choisi de visiter les sommets par ordre décroissant selon leur aire relative. Le processus de propagation débute la comparaison par les deux sommets fictifs qui représentent les objets.

La figure 4.11.c illustre le résultat du processus de propagation appliqué aux DAGs de la figure 4.11.b. Les couleurs représentent les familles de sommets similaires (lors de la construction des motifs, la couleur des parents a été propagée sur les sommets successeurs). Les parties des DAGs colorées en rouge correspondent au motif quasi-similaire qui a été identifié.

La reconnaissance des objets ne se base pas sur la mise en correspondance des sommets fictifs des DAGs. Le processus a pour but de reconnaître des motifs (sous-DAGs) dans les DAGs. Les ensembles de sommets S de D et S' de D' appartenant à des motifs sont identifiés comme sommets mis en correspondance. Ce sont ces ensembles que nous utilisons pour évaluer la similarité entre les deux objets.

4.3.3.1 Mesure de similarité des objets

Nous utilisons dans cette méthode la même mesure de similarité des objets que celle que nous avons défini pour la méthode par relaxation probabiliste (voir section 4.2.3.4).

4.3.3.2 Conclusion

Nous venons de présenter dans cette section une seconde approche pour la mise en correspondance d'objets dans la vidéo basse résolution. La méthode est une adaptation de l'heuristique pour la recherche de motifs quasi-similaires dans des graphes décrite dans le chapitre 2 de cette thèse.

Dans cette approche, la mise en correspondance est basée sur l'information structurelle des objets que l'on cherche à comparer. Les données extrinsèques aux régions (surface, couleur) sont utilisées pour renforcer les comparaisons basées structure.

Nous avons introduit une technique de construction de DAGs à partir des RAGs représentant les objets afin d'apporter une notion de hiérarchie entre les éléments des graphes. En se basant sur l'aire relative des régions et l'ajout d'un sommet fictif racine, nous construisons un unique graphe connexe orienté sans cycle comme modèle de représentation.

La seconde adaptation que nous avons apportée à la méthode générique concerne la notion de tolérance adaptative. En effet, plus les régions correspondant aux sommets à comparer sont proches en terme de couleur, plus on est tolérant aux déformations de structure pour l'identification de motifs. Cette étape permet d'éviter une reconnaissance systématique erronée de structures simples.

Dans la section qui suit, nous présentons un ensemble de résultats sur les deux méthodes que nous venons de proposer.

4.4 Résultats

Nous avons testé les méthodes pour la mise en correspondance d'objets sur des séquences vidéo issues de documentaires CERIMES au format MPEG-2. Les objets segmentés sont extraits des images DC de taille 76×92 pixels à une résolution temporelle d'environ deux images par seconde.

Les séquences utilisées dans ces travaux sont issues des documentaires vidéo CERIMES intitulées *Aquaculture en méditerranée, De l'arbre à l'ouvrage, Le chancre* et *Hiragasy* et contiennent près de 5000 images desquelles ont été extraits les objets. À une résolution temporelle de moins de 2 images/seconde, la base de données correspond à 52 minutes de vidéo. Pour les expériences, 100 objets correspondant à des personnes ont été choisis aléatoirement dans la base de données. La figure 4.12 montre un aperçu des séquences utilisées pour les tests.

(a) (b) (c)

Fig. 4.12: Exemples de séquences issues de la base de données.

4.4.1 Exemples de requêtes

Nous avons évalué les performances de nos méthodes dans le contexte de la requête par l'exemple. Les systèmes de recherche par l'exemple présentent le plus souvent le résultat en terme de k meilleures mises en correspondance [138, 54]. Un objet retourné est défini comme correct si il est proche de l'objet requête.

Nous avons comparé les résultats de nos méthodes avec la méthode IRM [80] utilisée dans le système SIMPLIcity [138]. L'algorithme a été implémenté en utilisant notre mesure de similarité entre les régions basée sur nos descripteurs (Form. (4.3)). La faible qualité des données implique que l'information sur la texture est lissée et les moments colorimétriques d'ordre supérieur à la moyenne ne sont pas représentatifs. Or, la méthode IRM est basée sur ces caractéristiques sophistiquées, elle est par conséquent moins efficace avec des descripteurs plus grossiers.

Deux exemples de requêtes pour nos méthodes en comparaison avec la méthode IRM sont présentés sur les figures 4.13, 4.14 et 4.15. Les scores sous les images correspondent à la mesure de similarité d'objets θ (Form. 4.8). L'objet requête est encadré de rouge. La figure 4.13 présente les résultats de la méthode IRM, la figure 4.14 correspond au résultats de notre méthode par relaxation probabiliste et les résultats de notre méthode heuristique sont montrés sur la figure 4.15.

FIG. 4.13: Méthode IRM : mesures de similarité avec la requête pour les 5 meilleures mises en correspondance.

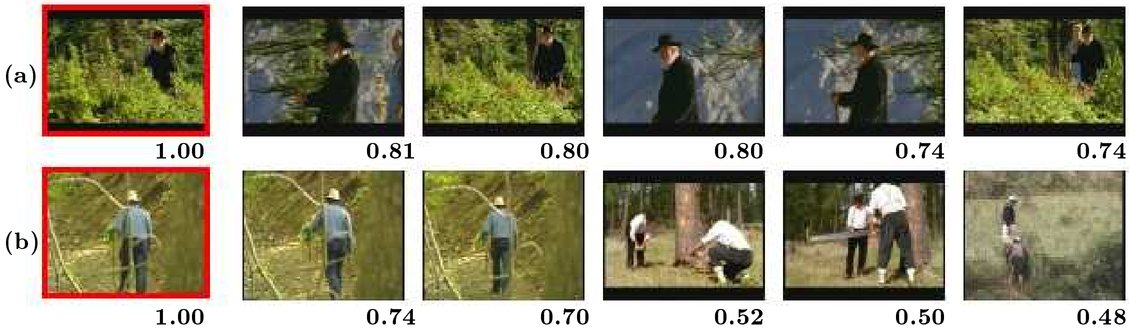


FIG. 4.14: Notre méthode par relaxation probabiliste : mesures de similarité avec la requête pour les 5 meilleures mises en correspondance.

L'exemple de requête (a) illustre l'aptitude de nos méthodes à retrouver le même objet dans différentes conditions : les mesures de similarités sont élevées même si le vieil homme apparaît dans deux séquences différentes. Pour nos deux méthodes, la présence de candidats, issus d'un autre plan que la requête, produisent un meilleur score que certains objets provenant du même plan que la requête. Cela est dû à la

Fig. 4.15: Notre méthode heuristique : mesures de similarité avec la requête pour les 5 meilleures mises en correspondance.

qualité du masque qui définit l'objet de premier plan. Le masque d'objet est obtenu de manière automatique par une analyse du mouvement [87] et ne correspond pas exactement à l'objet de premier plan contenu dans l'image (les parties statiques peuvent ne pas être détectées par l'analyse de mouvements et de petites régions du fond qui étaient cachées dans les images précédentes sont souvent incluses dans le masque). Par conséquent, la méthode ne sera pas capable de mettre en correspondance correctement les objets en raison de l'imprécision du masque de l'objet. On note que la méthode IRM est largement pénalisée par l'imprécision des descripteurs : la méthode utilise exclusivement l'information sur les caractéristiques des régions sans exploiter la structure de la partition.

Ainsi, l'apport de l'information sur le voisinage permet de limiter les problèmes d'imprécision des descripteurs liée à la faible résolution des données. La considération du voisinage dans le contexte du paradigme d'indexation grossière est alors justifiée.

Dans l'exemple de requête (b), les deux meilleurs candidats pour la méthode par relaxation, et les quatre meilleurs candidats pour la méthode heuristique représentent le même objet que la requête. Les autres objets retournés ne correspondent pas au même objet. Cependant, les structures des deux objets considérés (un homme debout avec un pantalon foncé et une chemise claire) sont très proches l'une de l'autre. La topologie des objets est suffisamment similaire pour que les méthodes de mise en correspondance ne soient pas perturbées par la différence colorimétrique.

Les deux exemples montrent l'intérêt de l'utilisation de l'information sur la structure pour compenser la faiblesse des descripteurs sur les données. Dans la méthode par relaxation probabiliste, on démarre par une mesure de similarité entre les paires de sommets qu'on ajuste itérativement en faisant croître ou décroître la valeur de similarité en fonction de la ressemblance de leurs voisinages. Dans cette approche, la stratégie consiste à utiliser la structure locale des objets pour raffiner une mesure de similarité basée sur les caractéristiques des régions. Dans l'approche heuristique, on propose de renverser le problème. Le processus débute par une capture de la similarité sur la structure et utilise ensuite les propriétés visuelles des régions pour guider le processus de propagation.

On note que les résultats produits par la méthode heuristique sont meilleurs pour les deux requêtes montrées en exemple. Ainsi, la structure des objets semble être une information qui reste stable pour les objets après sous-échantillonage.

4.4.2 Précision des méthodes

Une comparaison des performances des trois méthodes sur un ensemble de 100 objets (représentant des personnes) choisis aléatoirement dans la base de données est présentée sur la figure 4.16. La figure montre la précision (voir définition 33) de chaque méthode pour différents nombres de candidats considérés k.

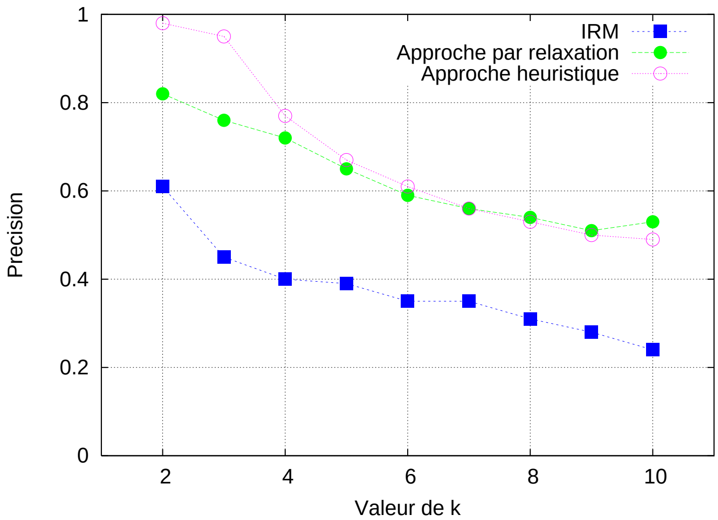


FIG. 4.16: Précision des méthodes pour différentes valeurs de k, nombre de meilleurs candidats retenus.

Les performances de la méthode IRM sont nettement inférieures aux méthodes que nous avons proposées dans ce chapitre, car celle-ci est fortement pénalisée par l'imprécision des descripteurs sur les régions. Les deux autres méthodes semblent compenser cette lacune grâce à l'utilisation de la structure des objets. Les deux approches présentées dans ce chapitre produisent des résultats comparables. L'heuristique est plus précise sur les trois meilleures réponses alors que la méthode par relaxation offre une meilleure précision pour les réponses de plus de 8 candidats. Dans la méthode par relaxation, la structure globale des objets n'est pas prise en compte et, deux grosses régions qui sont suffisamment proches au vu des descripteurs pour être mises en correspondance peuvent induire une forte mesure de similarité. Ces problèmes sont évités dans l'approche heuristique car la structure globale des objets, le voisinage local et les caractéristiques de taille et de couleur sont combinées pour l'identification des motifs communs entre les deux objets à comparer. Dans le cadre de la méthode par relaxation, certaines décisions définitives peuvent être prise pour les régions de taille importante. Ces assignations ne sont pas remises en cause lors des itérations suivantes. Ainsi, deux régions importantes avec des caractéristiques visuelles similaires peuvent être mises en correspondance sans tenir compte de leur voisinage. Cela peut induire une forte similarité entre les objets. Par contre cette configuration ne sera pas prise en compte par l'heuristique qui elle, décide des assignations en fonction des caractéristiques structurelles des objets. La relaxation probabiliste capture la structure au fur et à mesure des itérations alors que la méthode heuristique la considère dès le départ de la mise en correspondance.

4.4.3 Altérations d'images

Les descripteurs que nous avons définis sur les régions sont invariants aux transformation usuelles telles que la translation, la rotation et le changement d'échelle. Les performances des deux méthodes proposées ne sont par conséquent pas altérées par ces transformations.

Les altérations d'images telles que la variation de contraste, d'intensité ou l'introduction de bruit peuvent pénaliser la mise en correspondance dans le cas de la méthode par relaxation qui se base sur les descripteurs des régions. Dans le cas de la méthode heuristique, seul le paramètre de similarité colorimétrique (Form. (4.10)) est altéré par ces changements ce qui ne perturbe pas ou très peu la mise en correspondance.

Nous avons testé la robustesse de la méthode par relaxation aux altérations d'images. Les figures 4.17 et 4.18 résument les résultats. Le graphique de la figure 4.17 montre la précision sur les 5 meilleurs résultats pour une altération d'image croissante. La méthode est extrêmement robuste aux variations de contraste comme le montre la figure 4.17. Elle est également stable aux variations d'intensité.

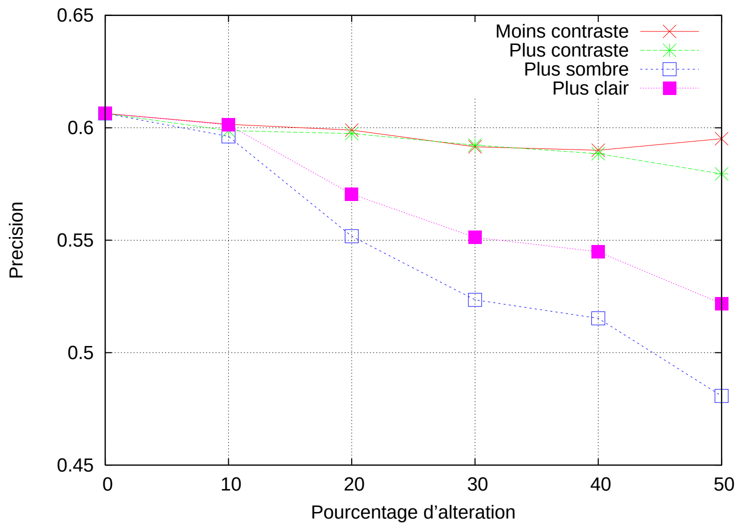


Fig. 4.17: Robustesse de la méthode par relaxation au variations de contraste et d'intensité. La précision est calculée sur les 5 meilleurs candidats.

La figure 4.18 montre des exemples de requêtes d'objets ayant subit une altération d'image telle que la variation de l'intensité, la variation du contraste et l'introduction de bruit aléatoire.

FIG. 4.18: Robustesse de la méthode par relaxation aux altérations d'image.

4.5 Conclusion

Nous avons présenté dans ce chapitre deux méthodes basées sur la comparaison de graphes pour la mise en correspondance d'objets dans la vidéo dans le contexte du paradigme d'indexation grossière. Dans ce contexte, l'information est pauvre en raison du sous-échantillonnage. Ainsi, les méthodes classiques de reconnaissance basées sur le contenu, qui utilisent le plus souvent des descripteurs sophistiqués pour caractériser les régions, sont fortement pénalisées dans notre problématique d'imprécision sur les données.

La première méthode proposée est inspirée des méthodes de relaxation probabiliste. La méthode effectue une mise en correspondance basée sur une mesure de similarité calculée entre les régions. Elle représente une amélioration des techniques de relaxation pour les objets vidéo de faible résolution modélisés par des RAGs.

La deuxième méthode est une adaptation de la méthode heuristique présentée dans cette thèse. Dans cette approche, on utilise des paramètres intrinsèques pour comparer les structures des objets. Les sommets sont classifiés par ressemblance, et deux sommets portant une même étiquette ont une structure quasi-similaire. L'information colorimétrique est utilisée en combinaison à la comparaison structurelle pour aider à la construction des motifs similaires.

Les approches produisent de bons résultats dans le contexte du paradigme d'indexation grossière. Le domaine d'application de ces méthodes peuvent concerner la recherche de plans vidéo qui contiennent un objet donné, l'inventaire sémantique des plans vidéo en chapitres ou scènes. D'autre part, on note que les méthodes n'exploitent pas l'information temporelle des données hormis pour l'extraction du masque de mouvement. La mise en correspondance ne nécessite aucune information quant à la dynamique des données. Ainsi, nos méthodes sont génériques et non spécifiques à la vidéo. Elles peuvent être appliquées pour la mise en correspondance d'images statiques.

En perspective de ces travaux, nous envisageons d'appliquer et adapter les méthodes pour les images en pleine résolution en définissant de nouveaux descripteurs sur les régions, adaptés aux données non dégradées. Nous projetons également l'adaptation des deux méthodes à la recherche et l'indexation d'images statiques, dans leur intégralité. Enfin, nous avons mentionné à plusieurs reprises dans ce chapitre les problèmes liés à l'imprécision du masque d'objet extrait automatiquement. En perspective directe à ce travail, nous souhaiterions tester les performances des méthodes proposées sur une base de données d'objets extraits de la vidéo avec précision, tels que les Video Object du standard MPEG-4.

Chapitre 5
Évolution logicielle

La maintenance du code existant est l'activité principale dans un projet de développement logiciel. Elle occupe plus de 80% de l'activité totale [107] pour un personnel de 60 à 80% de l'effectif travaillant sur le projet [32]. Depuis quelques années, de nombreux outils et techniques ont émergé pour aider à la maintenance logicielle. On peut citer tout d'abord les systèmes de gestion de versions du projet tels que CVS ou Subversion. Ces outils permettent aux développeurs de conserver un historique de versions du logiciel. Des outils aidant à la détection de clones dans le code ont aussi fait leur apparition dans les dix dernières années [17, 73, 1, 44, 137, 50]. En effet, l'existence de clones dans le code peut introduire de nombreuses instabilités dans une application logicielle : un changement dans une méthode peut entraîner plusieurs changements dans de nombreuses autres méthodes du programme. Ces outils sont extrêmement utiles aux développeurs.

Les managers d'équipes de développement ou le responsable du projet ont besoin d'une vision plus globale de l'évolution du projet pour identifier les problèmes ou les changements majeurs. Un nouveau type d'outils de visualisation destinés aux responsables du développement a commencé à voir le jour récemment. On trouve ainsi des outils comme CVSscan [136] permettant de visualiser au niveau de détails de la ligne de code (analyse d'un fichier) l'évolution du code (quelles lignes de code ont été ajoutées, supprimées, modifiées, qui a effectué ces modifications sur le code, quelles portions du code sont instables, ...). Le logiciel CVSgrab [134] permet de visualiser l'évolution d'un projet au niveau de détails du fichier (le projet est analysé). Cet outil aide à répondre aux questions telles que : quelle est la partie du projet la plus active ? Quand ont été créés, modifiés, supprimés les fichiers, et par qui ? Comment les tâches sont-elles distribuées aux développeurs ? ...). Ce nouveau type d'outils permet aux responsables d'avoir une vue globale de l'activité autour du projet, ils peuvent ainsi plus facilement identifier les éléments qui posent problème et pour lesquels il faut engager plus de moyens. À l'inverse, le responsable peut identifier une période de stabilité dans le projet et ainsi choisir de livrer une version fonctionnelle au client.

Dans ce chapitre, nous proposons une nouvelle méthode pour la détection et la visualisation de l'évolution de motifs dans le code C++. L'objectif est de comprendre comment la structure détaillée du code source change au cours de l'évolution d'un projet. Les développeurs sont intéressés de voir quelles parties du code ont changé, et comment. Nous avons cité l'outil CVSscan [136] qui permet de visualiser les changements au niveau de détails de la ligne de code. Cependant, ce dernier ne peut détecter et montrer les évolutions plus complexes comme les changements au niveau de détails du bloc de code (motif). Parmi les informations que l'on souhaite extraire, on peut citer le déplacement d'un bloc de code depuis une fonction (ou une classe) vers une autre ; la suppression d'un bloc de code correspondant à une boucle, un test, une fonction ; la fusion de deux fonctions, ou au contraire l'éclatement d'une fonction en plusieurs sous-fonctions. Toutes ces questions relèvent de l'analyse de motifs de code. La méthode que nous proposons est inspirée de la méthode heuristique présentée dans le chapitre 2 de cette thèse pour la reconnaissance de motifs quasi-similaires.

Le principe de l'algorithme proposé est le suivant. Dans un premier temps, on construit un arbre, appelé *arbre syntaxique annoté* (AST) représentant la hiérarchie syntaxique de chaque version du code source donné que l'on souhaite analyser. Pour cela, nous utilisons un analyseur syntaxique C++ [82] pour extraire l'arbre syntaxique complet de chaque version de chaque fichier qui nous intéresse dans le dépôt de versions. Nous utilisons ensuite une adaptation de la version 3 de l'heuristique pour la reconnaissance de motifs similaires afin de détecter les fragments de code similaires dans les ASTs correspondant aux versions consécutives. Au coeur de la technique, nous avons une mesure de similarité spécifique qui combine l'information sémantique et structurelle pour identifier ce qui est constant et ce qui a changé dans le code. Nous proposons également une méthode de visualisation pour présenter l'évolution de la structure du code qui souligne à la fois les motifs constants et les motifs qui ont évolué.

Le chapitre est organisé comme suit. Dans une première section, nous détaillons la méthode permettant d'obtenir les arbres syntaxiques annotés à partir du code source C++. Dans la section 2.1.1.2, nous introduisons la mesure de similarité entre les sommets des arbres basée sur l'information structurelle et sémantique que nous utilisons pour la phase de classification des sommets par similarité. La section 5.3 décrit la phase de détection des motifs similaires dans le code. La technique de visualisation est ensuite présentée dans la section 5.4.

5.1 Construction des arbres syntaxiques

La représentation du code source sous forme d'arbre syntaxique est une approche classique à l'analyse du code [17, 137, 73, 50]. Le principe est de modéliser la structure hiérarchique du code définie par les éléments syntaxiques tels que les classes, les fonctions, les identifieurs, les boucles, *etc* (voir section 1.6.3.3).

Dans ces travaux, la technique d'extraction d'arbres syntaxiques est basée sur l'approche décrite dans [82]. À chaque sommet u est associé le type d'élément hiérarchique du sommet noté $t(u)$ (`class`, `for`, `expression`, ...) permettant d'identifier le composant syntaxique représenté. Il existe 152 types syntaxiques dans la version des ASTs que l'on utilise, dont la plupart sont superflus pour notre application.

Dans la suite de ce chapitre, nous considérons N versions d'un fichier source $f_1, f_2, \cdots, f_N$. Pour chacune des versions f_i, on extrait l'AST complet associé T_i. Le principe est de comparer les paires d'arbres successives T_i, T_{i+1} et d'identifier les sous-arbres quasi-similaires. Pour cela, nous utilisons une adaptation de la version 3 de l'heuristique présentée dans le chapitre 2 du manuscrit. La section qui suit rappelle le principe de la classification des sommets par similarité et introduit une nouvelle mesure de similarité adaptée à la problématique.

5.2 Classification des sommets

Dans la première étape de la mise en correspondance, nous groupons les sommets des arbres T_i et T_{i+1} de deux versions consécutives en familles de sommets similaires. Les sommets appartenant à une même famille sont les racines potentielles de sous-arbres quasi-similaires. La construction des familles s'appuie sur une distance $d(u, v)$ définie entre deux sommets u et v.

Dans cette section, nous définissons une nouvelle mesure de similarité pour la classification des sommets d'arbres syntaxiques de code source.

Dans la version générique de l'heuristique, la mesure de similarité entre les sommets est basée sur l'information structurelle. Le principe est de classer dans une même famille les sommets dont les propriétés structurelles sont proches. Dans notre problématique de comparaison de code, il est important de considérer l'aspect sémantique des éléments. Ainsi, nous avons choisi d'introduire l'information sur le type du sommet dans notre mesure de similarité afin de classer les sommets structurellement proches et de type équivalents dans une même famille.

5.2.1 Distance structurelle

La distance basée sur les propriétés structurelles des sommets u et v reprend le principe de la méthode générique (voir 2.1.1).

Nous définissons la distance structurelle $d_{struct}(u, v)$ comme la différence entre trois métriques intrinsèques calculées sur les sommets u et v.

Les métriques que nous avons choisi de retenir pour chaque sommet u sont :

- le degré du sommet, dénoté $\delta(u)$
- le nombre de sommets du sous-arbre enraciné en u, dénoté par $\nu(u)$
- le nombre de Strahler du sommet, dénoté par $\sigma(u)$.

La combinaison de ces métriques permet une description complète de la situation structurelle d'un sommet u dans l'arbre syntaxique. En effet, le degré $\delta(u)$ indique la ramification locale du sommet u, la taille du sous-arbre $\nu(u)$ mesure l'importance du sommet au niveau hiérarchique et le nombre de Strahler $\sigma(u)$ capture en quelque sorte la "centralité" du sommet dans la structure (voir la définition 29).

La distance structurelle $d_{struct}(u, v)$ est basée sur les valeurs normalisées $\tilde{\delta}, \tilde{\nu}, \tilde{\sigma}$ des métriques (Form. (2.1)). Elle correspond à la mesure de similarité définie dans la méthode générique (Form. (2.2)) :

$$\rho_{struct}(u, v) = ((\tilde{\delta}(u) - \tilde{\delta}(v))^2 + (\tilde{\nu}(u) - \tilde{\nu}(v))^2 + (\tilde{\sigma}(u) - \tilde{\sigma}(v))^2 \qquad (5.1)$$

5.2.2 Distance de type

Lors de la comparaison de code, il est important de considérer des sommets de même type sémantique. Chaque sommet u d'un AST est associé à un type $t(u)$ (déclaration, fonction, affectation, boucle pour, ...). Nous utilisons ces types pour améliorer la mesure de distance entre les sommets.

Tout d'abord, il n'est pas cohérent de comparer des sommets de types complètement décorrélés. Par exemple, une expression arithmétique `a+b` et une déclaration `int x` ont une même structure d'AST. Dans les deux cas, nous obtenons un arbre constitué d'un arbre à deux feuilles : les identifieurs `a` et `b`, feuilles de l'opérateur `+` pour l'expression arithmétique `a+b` ; le type `int` et le déclarateur `x` sont les feuilles du sommet déclaration pour l'instruction `int x`. Cependant, les deux instructions sont différentes et ne constituent pas un motif de code similaire. Ainsi, la seule information sur la structure est insuffisante.

Nous modélisons cette différence par une seconde distance $d_{type}(u, v)$ entre les types $t(u)$ et $t(v)$. La solution la plus simple serait de considérer l'égalité stricte entre les deux types. Cependant, cette mesure est trop stricte. On cherche souvent à identifier de motifs similaires en permettant une certaine tolérance aux déformations *i.e.* des motifs qui ne sont pas strictement identiques, mais ressemblants, syntaxiquement proches. Pour cela, nous avons défini différentes classes de types pour les types syntaxiques C++, en se basant sur une définition empirique de ce que l'on considère comme similaire ou non.

La distance $d_{type}(u, v)$ est définie comme suit :

$$d_{type}(u,v) \begin{cases} 0 & \text{, si } t(u) \text{ et } t(v) \text{ appartiennent à une même classe} \\ 1 & \text{, sinon} \end{cases} \tag{5.2}$$

Des exemples de classes de types utiles sont les boucles `for, while` et `do` si l'on est intéressé par le suivi de structures itératives ; et classes, structures et unions si l'on cherche à suivre des types d'agrégation. Toutes les classes peuvent être redéfinies selon ce que chacun considère comme équivalent au vu de l'évolution de code. Dans notre exemple précédent, une déclaration de classe (`class`) C++ qui est modifiée en une structure (`struct`) mais qui conserve la même structure interne, est considéré comme un code identique. Nous ne détaillons pas les autres classes ici, puisque leur définition dépend de la tâche à résoudre, et pas seulement sur la grammaire C++ utilisée ici.

5.2.3 Distance globale

Les deux distances structurelles d_{struct} (Form. (5.1)) et sur le type d_{type} (Form. (5.2)) mènent à la distance globale $d(u, v)$:

$$d(u,v) = [1 + (w(u) + w(v))d_{type}(u,v)]d_{struct}(u,v) \tag{5.3}$$

Dans la formule (5.3), $w(u)$ est une valeur réelle déterminant un *poids*, ou un facteur d'importance, du type $t(u)$ d'un sommet u. Si les types des sommets comparés u et v sont dans une même classe de type, alors $d(u, v)$ équivaut à la distance structurelle $d_{struct}(u, v)$. Si tel n'est pas le cas, *i.e.* $d_{type} = 1$, alors $d(u, v)$ est supérieure si les types sont considérés comme importants. Le poids w d'un type est utile pour souligner certains types de changements lors de l'évolution de code. Par exemple, si nous portons notre intérêt sur l'évolution de l'architecture, alors tout les types de sommets qui concernent des déclarations globales (comme les fonctions, les classes, *etc*) seront importants, alors que les types relatifs à l'implémentation (comme les expressions, les déclarations, les structures itératives, *etc*) seront peu pertinents. Dans cet exemple, on pourra fixer une valeur élevée w pour les sommets de type formels (structure, fonction, classe) et un faible poids aux autres types. Pour un point de vue neutre, tous les poids w pour tous les sommets des AST sont fixés à 1.

Nous utilisons la distance $d(u, v)$ de la formule (5.3) pour répartir les sommets de T_i et T_{i+1} en familles de sommets équivalent, comme décrit dans la méthode heuristique générique (Sec. 2.1.1.2). Ici, deux sommets sont considérés comme ϵ-similaires si la distance globale $d(u, v)$ est strictement inférieure à ϵ. Ainsi, la condition de la formule (2.2) de la méthode générique est, dans notre application, remplacée par :

$$d(u, v) < \epsilon$$

Les sommets des arbres T_i et T_{i+1} sont alors répartis en L familles distinctes par le processus d'étiquetage défini par l'algorithme 2.1.

5.3 Construction des motifs similaires

La première phase de la mise en correspondance que nous venons de décrire dans la section précédente répartit les sommets de chacune des deux versions consécutives i et $i + 1$ en familles. Nous ne pouvons déterminer à ce stade quels sont les sous-arbres qui sont effectivement quasi-similaires. Cette deuxième phase consiste en une analyse en profondeur qui trouve et marque tous les sous-arbres identifiés comme quasi-similaires.

Dans cette application, les structures d'arbres syntaxiques représentent la hiérarchie du code. Ainsi, pour un sommet u donné dans un AST, la totalité du sous-arbre fait partie intégrante de l'instruction définie par le sommet u. On ne peut dissocier un sommet ancêtre de ses successeurs dans l'arbre car chaque élément du sous-arbre correspond à une sous-composante de l'élément hiérarchique parent dans l'arbre. Ainsi, la reconnaissance de motifs similaires dans les ASTs doit considérer une similarité qui prend en considération l'ensemble de la sous-structure de deux sommets comparés pour établir une correspondance. La version de l'heuristique adaptée dans ce cas est la version 3 qui identifie la similarité entre des sous-structures entières (voir section 2.1.2.4).

Pour optimiser les comparaisons de sommets appartenant à une même famille de sommets équivalents, nous avons choisi de trier les sommets par ordre lexicographique. Ainsi, nous favorisons l'appariement de structures identiques au niveau du texte avant de considérer des paires de sommets similaires.

Nous n'avons pas apporté de modifications à l'algorithme de propagation proposé dans la version générique de l'heuristique. Le processus complet d'identification de sous-arbres similaires est décrit dans la section 2.1.2.4.

À l'issue du processus, nous connaissons les sommets qui représentent les blocs de code similaires au cours des différentes versions du fichier considéré. Nous avons proposé une interface de visualisation pour présenter à l'utilisateur une vision globale de l'évolution du code. La technique de visualisation est présentée dans la section qui suit.

5.4 Interface de visualisation

Dans cette section, nous décrivons la technique de visualisation que nous avons développée pour présenter le résultat de l'identification de blocs de code similaire au cours des versions successives d'un même fichier source. L'objectif est de souligner les évènements importants dans l'évolution, tels que la refactorisation de code, le déplacement de code, ou les changements d'implémentation et d'architecture.

5.4.1 La technique basique

La visualisation utilise le principe esquissé dans la figure 5.1. Chaque arbre syntaxique T_i pour chaque version i de l'évolution d'un fichier source est dessiné en utilisant une technique de représentation 2-D par niveaux similaire à la technique "icicle plot" décrite dans [16].

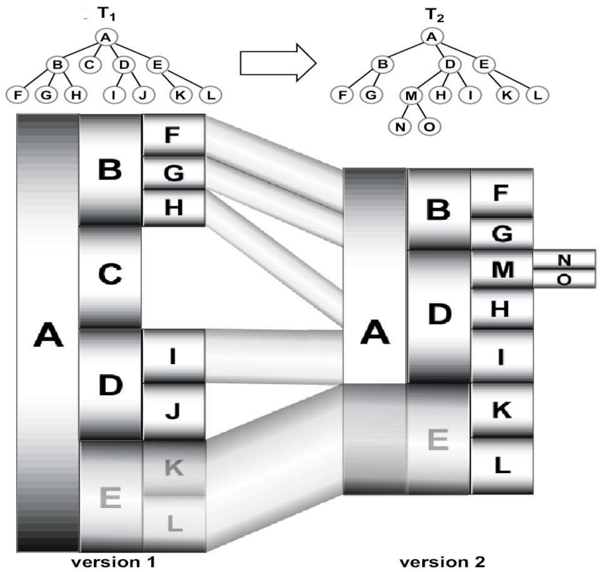


FIG. 5.1: Visualisation de deux versions de code et de leurs arbres syntaxiques T_1 et T_2.

Un arbre apparaît comme une bande verticale, où la racine est représentée à gauche et les feuilles à droite. La structure hiérarchique de l'arbre est modélisée par un effet visuel de sous-ensembles traduit par des zones rectangulaires. Chaque niveau hiérarchique de l'arbre constitue une sous-bande verticale. Cette sous-bande est subdivisée en plusieurs zones rectangulaires qui définissent les différents sommets du niveau. Dans l'exemple de la figure 5.1, les sommets B, C, D et E appartiennent au même niveau hiérarchique, ils sont donc représentés comme des éléments d'une même bande verticale. La notion de successeur est traduite par l'alignement horizontal des rectangles intra-bandes. Ainsi, des rectangles d'une bande inscrits entre les bords supérieurs et inférieurs d'un rectangle r situé dans la bande à gauche représentent des successeurs du sommet associé au rectangle r. Dans l'exemple de la figure 5.1, le sommet D de l'arbre T_1 a pour successeurs les sommets I et J.

La figure 5.2 montre un exemple de la technique de visualisation appliquée à un arbre syntaxique d'une fonction simple. L'arbre syntaxique est représenté en utilisant un algorithme de dessin d'arbre classique (a) et en utilisant notre technique de visualisation (b).

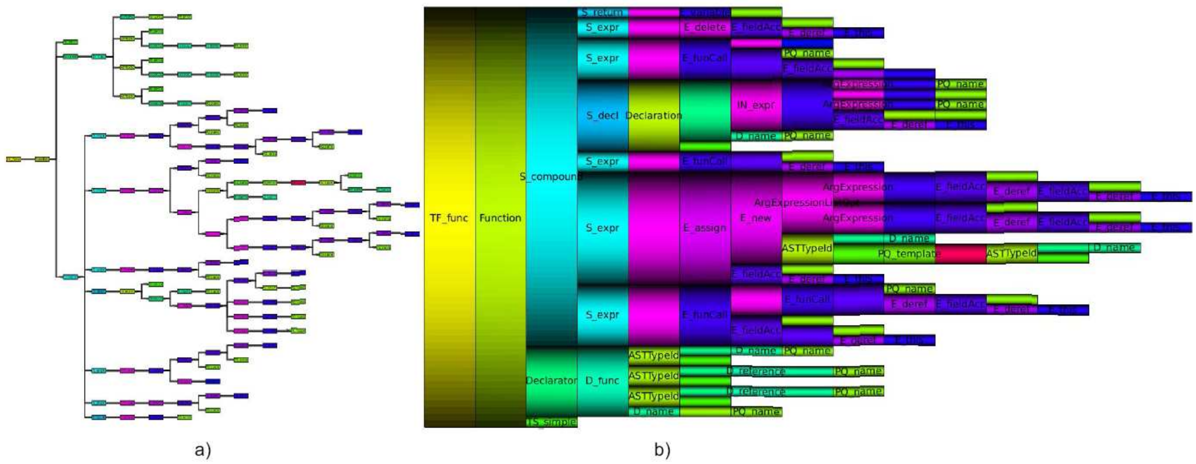


FIG. 5.2: Arbres syntaxiques visualisés par une méthode classique (a) et en utilisant la techniques de coussins ombragés (b).

Les ASTs d'un code typique C++ constitué de plusieurs milliers de lignes sont très larges (dizaines de milliers de sommets) mais pas très profonds (10 à 40 niveaux). Ainsi, les arbres apparaissent en bandes verticales fines et hautes. La variation de la profondeur des arbres syntaxiques est traduite par l'épaisseur de la bande (voir les figures 5.3 et 5.4 pour des captures d'écrans réelles).

Pour souligner la structure, nous avons introduit un effet de "coussins ombragés", comme il a été utilisé avec succès dans d'autres outils de visualisation [136, 82], en appliquant une texture uni-dimensionnelle de variation de la luminance. La visualisation complète est implémentée via le système de visualisation de graphes Tulip [5].

Après avoir dessiné tous les arbres de toutes les versions considérées du fichier, nous affichons les correspondances. Comme il est expliqué dans la section 5.3, étant donnés N arbres $T_1, \cdots, T_N$ pour les versions de fichiers $f_1, \cdots, f_N$, on ne cherche des correspondances qu'entre des versions successives T_i et T_{i+1}. On représente chaque correspondance entre les rectangles des sommets racines des sous-arbres similaires en dessinant des tubes, comme sur la figure 5.1. Les tubes sont faciles à interpréter. S'ils sont droits et horizontaux, les tubes montrent les blocs de code qui restent constants au fil des versions. Les arbres syntaxiques extraits sont ordonnés, et du code constant correspond à des sous-arbres constants, à la même place. Les tubes inclinés indiquent que du code a été déplacé dans l'arbre syntaxique. Des tubes qui se croisent traduisent une inversion de blocs de code. Et comme l'ordre des sommets de haut en bas dans la vue correspond à l'ordre d'apparition des fragments de code dans le fichier, de longs tubes indiquent des fragments de code qui ont été déplacé considérablement dans le fichier source. Les insertions et suppressions de lignes de code apparaissent comme des régions dans l'arbre qui ne sont pas connectées par des tubes. Enfin, l'épaisseur des tubes traduit la quantité de lignes de code du motif identifié.

Les tubes peuvent être dessinés de deux manières. Dans une première version, comme il est montré pour les sommets F à I de la figure 5.1, les tubes sont dessinés depuis le bord droit du sommet dans la version i au bord gauche de la racine dans la version $i+1$. La deuxième solution, utilisée pour le sommet E de la figure 5.1, le tube est dessiné de façon à recouvrir les deux sommets mis en correspondance et leurs successeurs (K, L) en utilisant un effet de transparence. Nous avons retenu la deuxième version qui nous semble meilleure pour indiquer clairement le groupement de sommets dans les motifs. La seconde option pour le dessin des coussins a un avantage plus subtil. Si nous regardons en détails les deux sommets H de la figure 5.1, le tube qui connecte les deux sommets ne semble pas être aligné avec le sommet de l'arbre T_2. En réalité, c'est un effet d'optique surprenant dû au remplissage avec l'effet coussin. L'utilisation de la deuxième alternative pour dessiner les tubes, qui utilise la superposition avec transparence, corrige ce problème.

La figure 5.3 montre le résultat de la technique de visualisation sur deux versions d'un fichier d'environ 850 lignes de code C++. Dans la représentation de gauche, l'effet de coussins n'a pas été utilisé. Les sommets sont colorés par une teinte qui indique leur type C++, en utilisant différentes teintes pour les classes, les fonctions, les méthodes, les expressions, les déclarations, les structures itératives et les structures conditionnelles. Les tubes sont colorés en utilisant une teinte légèrement plus foncée que les sommets connectés. La représentation de droite montre les mêmes données. Cette fois-ci, les sommets et les tubes sont colorés en utilisant une teinte différente (définie aléatoirement) pour chaque motif identifié. Les tubes sont dessinés avec l'effet de transparence mentionné ci-dessus. Les sommets résultant de l'insertion ou la suppression de code, c'est à dire les sommets non appareillés, sont colorés de gris. La figure 5.3.b montre clairement plusieurs événements entre les deux versions : le bloc C a été supprimé, à l'exception de quelques lignes de code indiquées par les filaments rouges au milieu. Les blocs A et B ont été inversés sans avoir subi de modifications. Les fragments D et E sont restés constants.

Nous pouvons toutefois observer une limitation à notre méthode. Bien que les fragments D et E soient constants, ils sont identifiés (et dessinés) comme un ensemble de multiples blocs au lieu d'un seul et même bloc, comme on peut l'observer pour les blocs A et B. La raison est que A et B correspondent à des sous-arbres

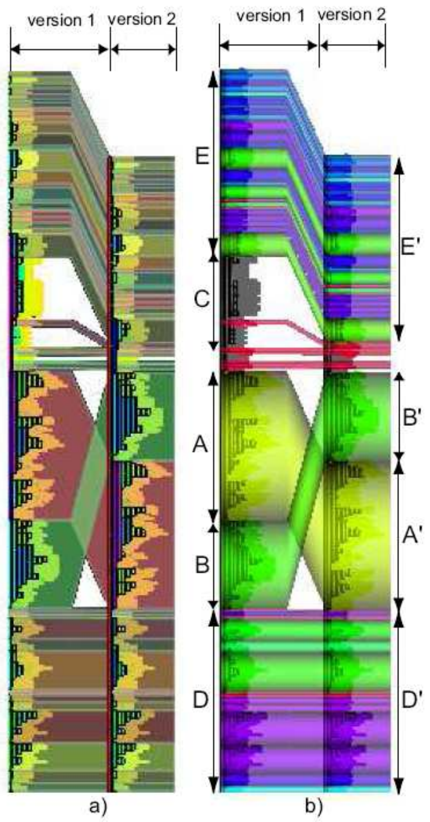


FIG. 5.3: Deux versions d'un fichier source de 850 lignes de code environ.

simples dans l'arbre syntaxique, alors que D et E sont constitués de plusieurs sous-arbres qui ne partagent pas un même parent. Dans cet exemple, A et B sont deux méthodes globales, tandis que D et E sont des ensembles de sommets représentant de nombreuses déclarations d'une même fonction ayant subi des modifications (bloc C). Comme notre méthode met en correspondance uniquement des racines de sous-arbres, il peut apparaître une large quantité de sous-fragments. On note que, si on le souhaite, on peut éviter cet effet par un processus *a posteriori* groupant les ensembles de motifs contigus partageant le même parent en un seul et même motif.

5.4.2 Les améliorations de rendu

Plusieurs améliorations peuvent être ajoutées à la technique de visualisation basique décrite ci-dessus. Premièrement, l'utilisateur peut utiliser le *niveau de détails* pour lequel les arbres syntaxiques sont montrés. Étant donné un niveau de détails, seulement les sommets de type syntaxique d'un niveau supérieur au niveau sélectionné, et les correspondances associées, sont affichés. Cela permet de simplifier la visualisation en supprimant les détails du code qui apparaissent comme des bandes supplémentaires à la droite des visualisations des différentes versions (*e.g.* les sommets $F, G, \cdots, L$ de la figure 5.1). Un effet similaire peut être obtenu en adaptant la fonction de distance sur la structure d_{struct} (voir section 5.2.1). La différence réside dans le fait que le changement de la distance structurelle implique potentiellement l'identification de motifs différents, alors que l'occlusion de détails dans la visualisation a pour seul effet de montrer moins de correspondances pour une mise en correspondance déjà calculée.

5.5 Résultats

Pour valider les résultats, nous avons appliqué la méthode complète sur différentes versions d'un fichier extrait de différentes versions d'un projet C++, et vérifié les évolutions de motifs détectées en inspectant manuellement le code source. L'exemple considéré concerne un programme complet de traitement d'images d'approximativement 5000 lignes de code. Suite aux discussions avec les développeurs, nous savons que les versions correspondent à une succession de refactorisations de code à petite échelle. La figure 5.4 montre la visualisation de six versions d'un des fichiers contenant près de 1500 lignes de code de notre base C++. La couleur des sommets indique le type syntaxique des sommets. Les tubes ont la couleur des sommets qu'ils mettent en correspondance.

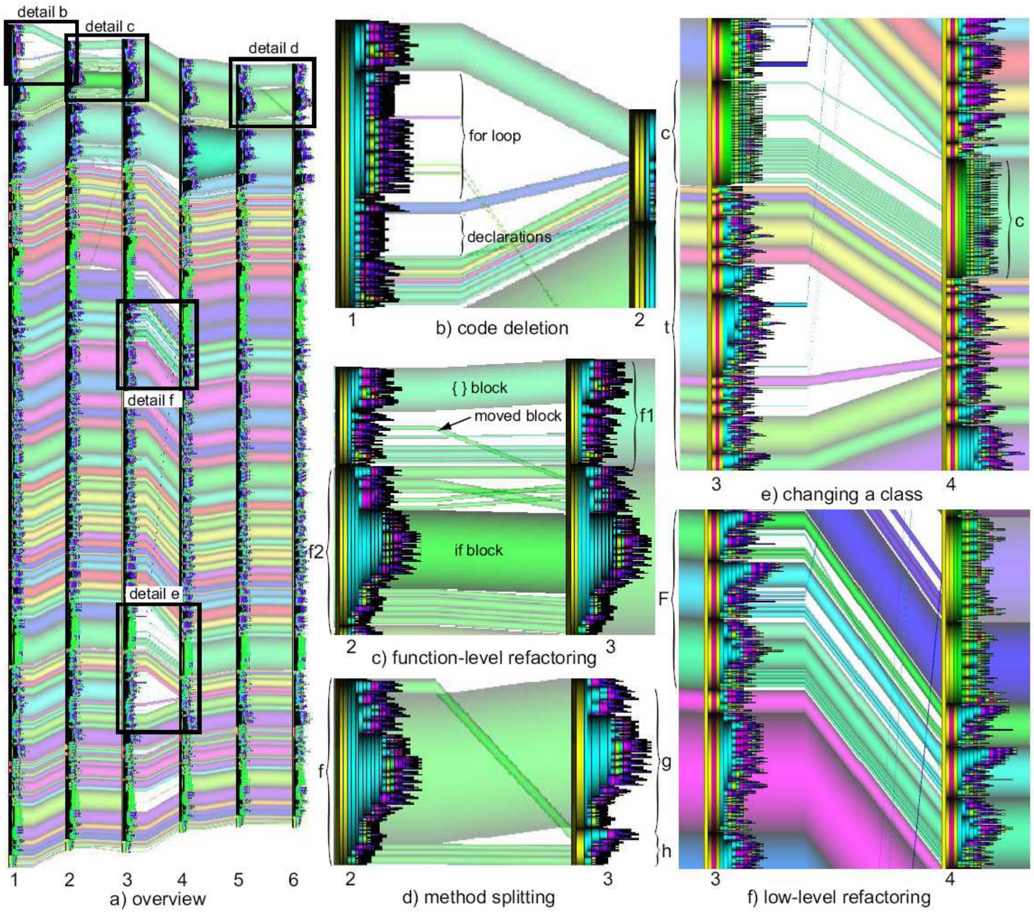


Fig. 5.4: Deux versions d'un fichier source de 500 lignes de code environ.

Nous pouvons rapidement identifier quelques zones du code où des changements importants ont eu lieu (les zones marquée b-f dans la figure 5.4). Pour le reste du code, les tubes parallèles qui parcourent les versions de gauche à droite dans l'image globale (Fig. 5.4.a) indiquent des blocs de code constants ou presque constants. Globalement, nous pouvons affirmer que le nombre de lignes de code s'est graduellement réduit au fil des versions.

Concentrons-nous maintenant sur les vues de détails (figures 5.4.b-f). La figure 5.4.b montre deux suppressions entre la version 1 et la version 2 du code. La première est une boucle `for` (approximativement 30 lignes de code) et la seconde est un ensemble de déclarations de variables locales (10 lignes de code). Le tube qui sépare les deux blocs supprimés correspond à un bloc de code qui est resté inchangé, localisé entre les deux portions de code supprimées. La figure 5.4.c montre une modification plus complexe, impliquant deux fonctions `f1` et `f2`. La mise en correspondance a détecté un bloc d'instruction `{ ... }` dans `f1` et une large structure conditionnelle `if` dans `f2` qui sont constants. Nous notons également un petit bout de code qui a migré de la fonction `f1` vers `f2`. De plus, nous pouvons voir des éléments de code à la fin de la fonction `f2`, juste en dessous du bloc `if`, qui ont été mis en correspondance. Cependant, il n'y a pas de tubes qui connectent l'intégralité du contenu de `f1` et `f2` dans les deux versions : bien que de nombreux blocs aient été identifiés comme stables, les deux fonctions ont été considérées comme trop différentes pour être mises en correspondance.

La figure 5.4.d montre une fonction `f` qui a été découpée en deux fonctions `g` et `h`. La plupart des lignes de code de `f` de la version 2 ont été déplacées dans la fonction `g` de la version 3. Le code indiqué par le tube en diagonale et les tubes fins en bas de la fonction `f` ont été déplacés dans la fonction `h`.

La figure 5.4.e montre une classe `c` qui a subi de nombreuses modifications entre les versions 3 et 4. Les tubes qui traversent en diagonale indiquent les membres de la classe qui sont restés les mêmes. Cependant, comme pour les fonctions `f1` et `f2` de la figure 5.4.c, la classe entière n'est pas mise en correspondance parce qu'il y a trop de changements, *e.g.* plusieurs méthodes et membres de classes ont été remplacés. Dans la même image, nous pouvons voir une classe paramétrée (`template`) `t` qui a subi une suppression de code massive entre les versions 3 et 4.

Enfin, la figure 5.4.f montre l'évolution d'une portion de l'implémentation `F` qui a subi plusieurs changements minimes, tels que l'insertion ou la suppression de variables locales, des changements de termes dans les expressions arithmétiques, et des substitution d'expression de type `a<b ? a : b` par des structures conditionnelles `if-else`. Bien que ces changements soient peu complexes, ils sont suffisamment nombreux pour que les éléments de la portion `F` soient identifiés comme similaires et donc ils apparaissent dans la visualisation.

5.6 Conclusion

Nous avons présenté une méthode pour l'identification et la visualisation de l'évolution de motifs similaires dans un code source C++. Au centre de la méthode, nous avons proposé une technique de mise en correspondance basée sur la structure et le type appliquée aux arbres syntaxiques. La méthode est une adaptation de l'heuristique étudiée dans ce manuscrit. Les sous-arbres mis en correspondance sont visualisés par une méthode d'affichage basée sur la représentation dense des arbres et des techniques de rendu. Notre méthode est utile pour détecter et analyser les changements de petite et moyenne échelle entre les différentes versions d'un code source, tels que la refactorisation de code au niveau de l'édition de classe. Nos travaux remplissent la lacune entre les outils d'analyse au niveau de détails de la ligne de code tels que CVSscan ou WinDiff [136] et les outils au niveau de détails du fichier ou de l'architecture tels que CVSgrab ou sv3d.

Nous envisageons plusieurs perspectives à ces travaux. Dans un premier temps, la visualisation présentée est juste un prototype pour afficher les correspondances syntaxiques. De nombreuses variantes de l'idée basique peuvent être explorées, tels que l'omission des fragments de code inchangés pour ne montrer que les ajouts, les suppressions et les modifications. Deuxièmement, les différentes fonctions de distances peuvent être améliorées en ajoutant de l'information additionnelle venant compléter les données sur la structure des arbres syntaxiques et le type de sommet. Nous pensons entre autres aux données spécifiques au projet ou à l'utilisateur telle que les conventions d'appellation de variables, pour détecter différent types d'évènements d'intérêt pour des tâches particulières. Troisièmement, la visualisation proposée peut être améliorée pour permettre de montrer plus de versions de code contenant plus de lignes. Enfin, les deux techniques d'identification et de visualisation ont le potentiel d'être appliquées dans le contexte de la recherche de clones dans le code.

Conclusion

Nous avons présenté dans ce manuscrit un ensemble de méthodes et algorithmes basés sur les graphes pour traiter différents aspects du problème de la comparaison de graphes.

Recherche de motifs quasi-similaires

Nous avons proposé une formalisation de l'heuristique pour la recherche de motifs quasi-similaires généralisée aux graphes. Nous avons décliné en trois variantes la stratégie de propagation pour la construction des motifs. Cette méthode est générique et peut être adaptée à différentes applications.

Nous avons testé la méthode heuristique sur différentes problématiques :

- comparaison de systèmes de fichiers utilisateurs (arbres)
- comparaison de réseaux du trafic aérien à deux périodes différentes (graphes non orienté)
- comparaison de réseaux de migrations professionnelles à deux instants différents (graphes orientés).
- mise en correspondance d'objets dans la vidéo basse résolution (graphes orientés sans cycles)
- reconnaissance de blocs de code similaires pour la visualisation de l'évolution de code d'un projet logiciel (arbres).

Dans les exemples du trafic aérien et des réseaux de migrations professionnelles, l'information propre aux données n'a pas été exploitée : la méthode heuristique a été appliquée en n'utilisant que des paramètres intrinsèques sur les sommets. Les résultats sont très satisfaisants dans les deux cas. Dans les autres cas, la méthode a été adaptée en introduisant de l'information propre aux données manipulées. Les résultats sont également très prometteurs dans ces cas.

Il existe de nombreuses perspectives à ces travaux. L'heuristique pour la comparaison de structures de graphes peut être adaptée à de nombreuses autres applications. Nous envisageons, entre autres, d'approfondir la recherche sur la comparaison de réseaux de trafic (aérien, routier, ferroviaire) en proposant une interface de visualisation de l'évolution du trafic. Nous projetons également d'utiliser la méthode pour l'étude des réseaux métabolomique pour l'identification des effets d'un traitement thérapeutique [24].

Modèle de génération aléatoire de système de fichiers

Nous avons proposé et expérimenté un nouveau modèle pour la génération aléatoire de systèmes de fichiers utilisateurs. Le modèle est basé sur un ensemble de propriétés observées sur un jeu de données réel. Nous avons proposé deux versions du modèle : le modèle de base qui produit des arborescences aux propriétés proches de la réalité, et le modèle étendu, qui permet de prendre en considération le profil de l'utilisateur. Dans cette deuxième version, le profil de l'utilisateur est caractérisé par des propriétés structurelles observées sur un système de fichier réel complet. Une étude comparative des résultats avec les modèles de génération aléatoire classiques montre l'intérêt de notre modèle.

En perspective à ces travaux, nous envisageons une étude plus approfondie du comportement des utilisateurs. L'acquisition à des instants très rapprochés (de l'ordre d'une capture par jour) des arborescences est nécessaire pour identifier clairement l'évolution du système de fichiers. Pour analyser les tendances dans l'évolution des arborescences, une interface de visualisation mettant en évidence les modifications apportées au fil du temps est envisageable.

Mise en correspondance d'objets vidéo

Nous avons proposé deux méthodes pour la mise en correspondance d'objets dans la vidéo à très faible résolution. La première se base sur des techniques de relaxation probabiliste. La seconde utilise une adaptation de la méthode heuristique de recherche de motifs quasi-similaires. Nous avons montré l'importance de considérer l'information sur la structure des objets dans le paradigme d'indexation grossière pour compenser l'imprécision des descripteurs visuels sur les données sous-échantillonnées. Les deux méthodes proposées offrent des résultats comparables sur les expérimentations.

L'imprécision du masque de l'objet qui est extrait automatiquement implique des difficultés pour la mise en correspondance, car les objets de premiers plans ne sont pas toujours correctement détectés (nombreuses sous-détections et sur-détections). Ainsi, quelle que soit la méthode de mise en correspondance, si le masque d'objet présente trop de différences avec la zone d'intérêt de l'image, alors il est impossible d'identifier cet objet. Des expérimentations supplémentaires sur des objets vidéo plus précis (*i.e.* extraits manuellement ou via des méthodes semi-automatiques), comme les Video Objects du standard MPEG-4 seront effectuées très prochainement.

D'autre part, nous envisageons d'adapter les méthodes à la mise en correspondance d'objets en pleine résolution, pour lesquels les descripteurs visuels sont plus pertinents.

Évolution de code

Nous avons proposé une nouvelle méthode pour la visualisation de l'évolution de motifs de code dans un projet logiciel. La technique de recherche de motifs utilise une adaptation de la méthode heuristique appliquée aux arbres syntaxiques annotés correspondant aux différentes versions d'un fichier source. L'interface de visualisation proposée met en évidence les blocs de code mis en correspondance via un algorithme de représentation des arbres orientée pixel et différentes techniques de rendu.

Les résultats obtenus dans les expérimentations sont très prometteurs. Les modifications de petite et moyenne échelle dans le code sont rapidement identifiables via l'interface de visualisation proposée. Toutefois, de nombreuses autres alternatives sont envisageables. Pour les fichiers comprenant de nombreuses lignes de code, la méthode actuelle présente le défaut de nécessiter un large espace pour représenter les arbres syntaxiques. Des expérimentations supplémentaires en utilisant d'autres méthodes de représentation des résultats sont nécessaires : une visualisation ne montrant que les changements entre les différentes versions permettrait de limiter le problème de l'espace de représentation. D'autre part, l'algorithme proposé n'est applicable qu'aux projets logiciels programmés en langage C++. Nous projetons d'étendre la méthode de reconnaissance de motifs à d'autres langages de programmation en prenant en compte les spécificités des arbres syntaxiques associés.

Annexe A

Génération aléatoire

Nous présentons dans cette annexe un ensemble de résultats complémentaires au chapitre 3.

A.1 Les résultats sur les arbres squelette

Dans cette section, nous montrons le résultat de l'étude des propriétés des *arbres-squelette* (voir définition 42) de notre base de données réelles.

La figure A.1 illustre la distribution des degrés des arbres-squelette de la base de données pour différentes fourchettes de taille.

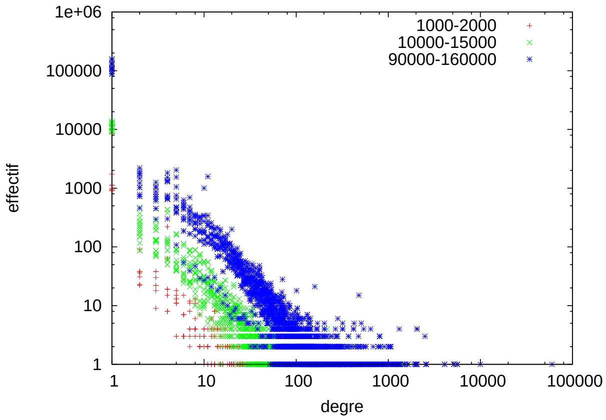


FIG. A.1: Distribution des degrés sur le jeu de données pour différentes tranches d'arbres.

La figure A.2.a (resp. Fig A.2.b) montre la relation entre le nombre d'éléments des arbres-squelette et la hauteur (resp. la largeur).

La figure A.3 représente, pour une hauteur h, la distribution moyenne des éléments dans les niveaux des arbres-squelette.

La figure A.4 représente, pour chaque niveau dans l'arbre-squelette, le pourcentage de fichiers par rapport au nombre total d'éléments dans le niveau correspondant.

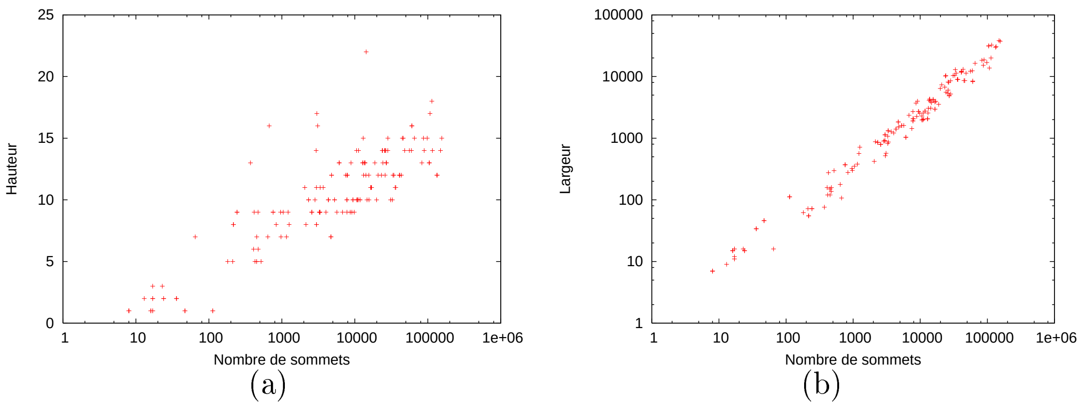


FIG. A.2: Hauteur (a) et largeur (b) des arbres en fonction de leur taille.

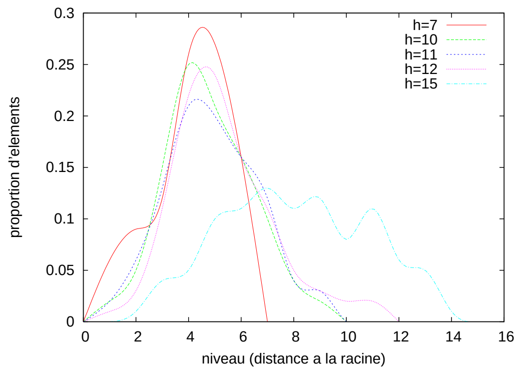


FIG. A.3: Distribution des éléments selon le niveau dans l'arborescence de fichiers.

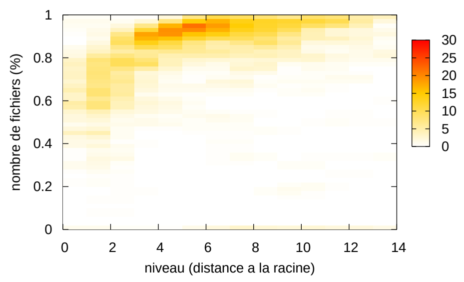


FIG. A.4: Proportion de fichiers par niveaux dans l'arbre.

A.2 Les résultats sur les autres tranches

Dans cette section, nous présentons un complément aux résultats montrés dans la section 3.3.

A.2.1 Distribution des degrés

Les figure A.5.a-k montrent la distribution des degrés dans les arborescences par tranches de taille d'arbres.

A.2.2 Forme des arbres

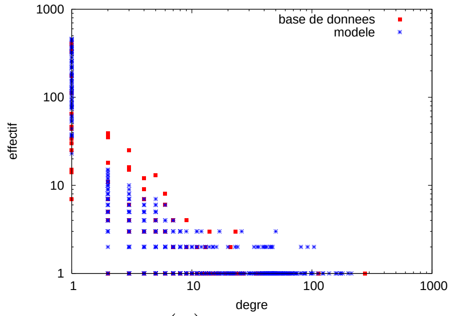
Sur les figures A.6.a et A.6.b sont affichées respectivement la hauteur et la largeur de chaque arbre en fonction du nombre d'éléments.

A.2.3 Répartition des éléments

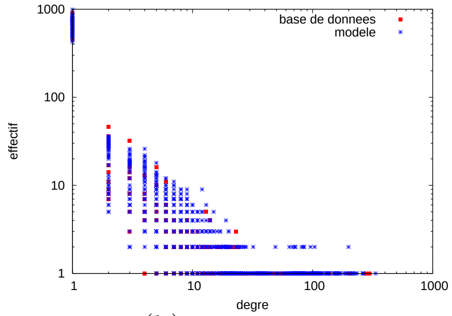
Nous montrons sur les figures A.7.a-k la répartition des éléments selon le niveau dans l'arbre, par tranches de taille.

A.2.4 Composition des niveaux

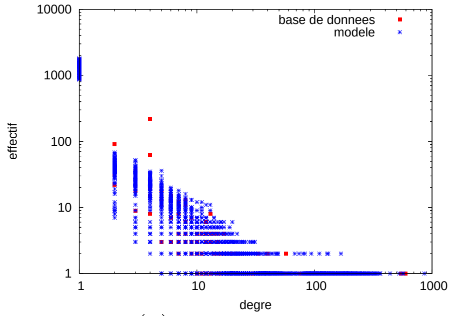
La figure A.8 montre la proportion de fichiers selon le niveau dans l'arbre. On représente ici la carte d'élévation pour l'ensemble des arbres-générés.



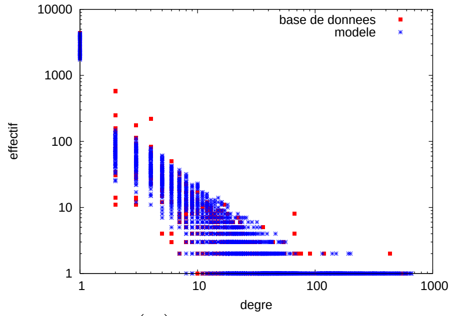
(a) 0-500



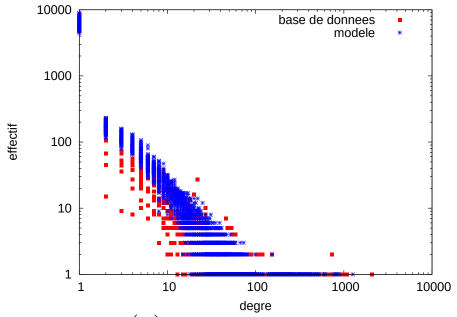
(b) 500-1000



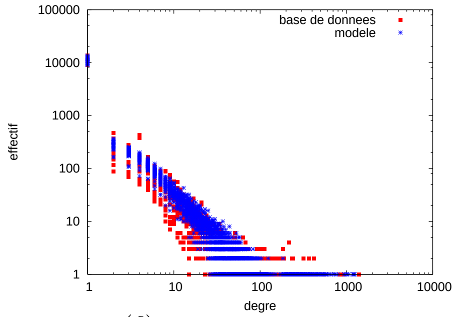
(c) 1000-2000



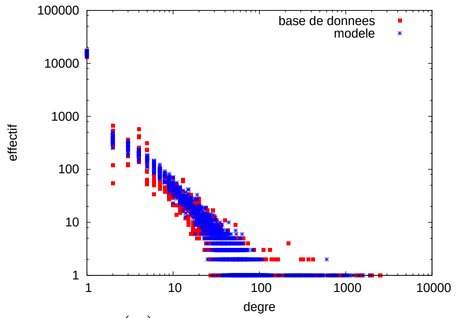
(d) 2000-5000



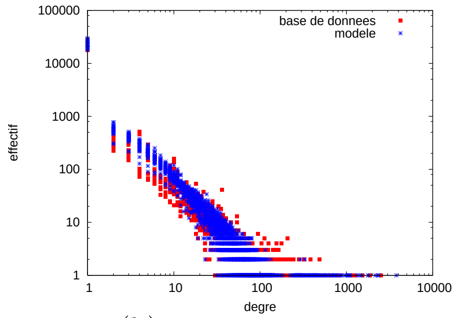
(e) 5000-10000



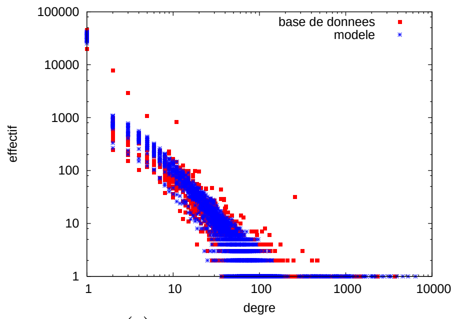
(f) 10000-15000



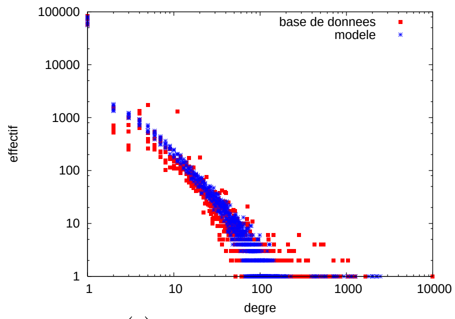
(g) 15000-20000



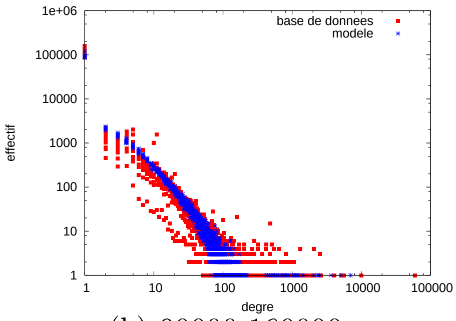
(h) 20000-30000



(i) 30000-50000

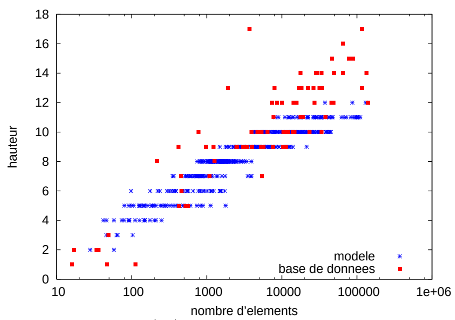


(j) 50000-90000

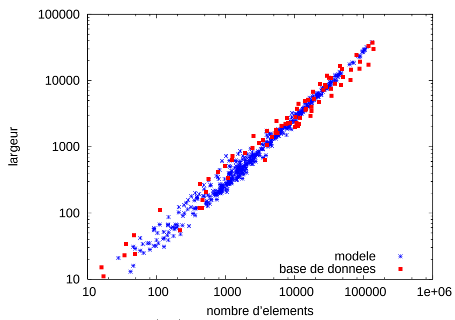


(k) 90000-160000

FIG. A.5: Distribution des degrés sur les différentes tranches de taille.

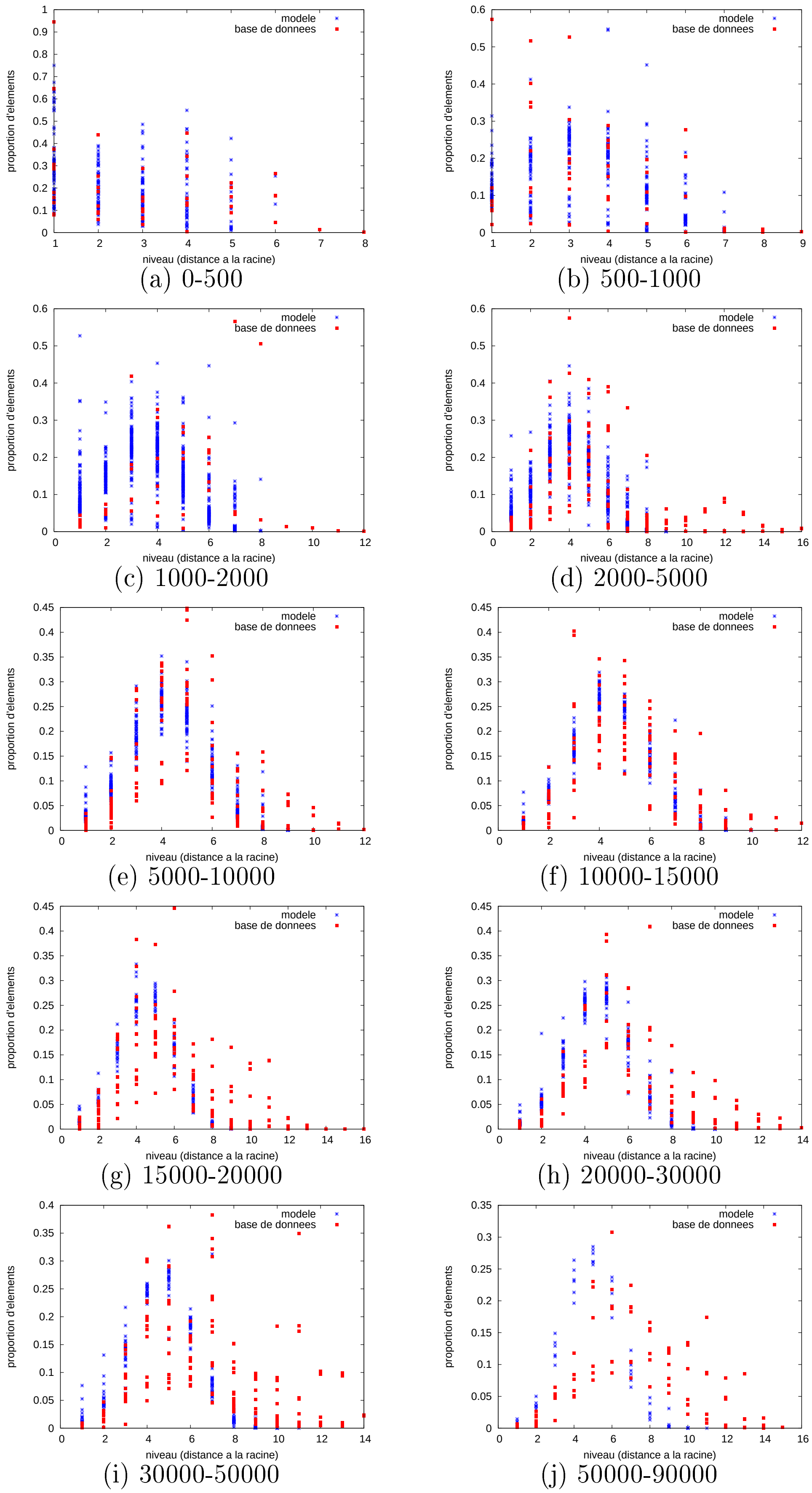


(a) Hauteur

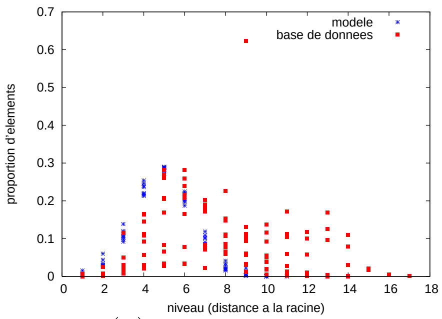


(b) Largeur

FIG. A.6: Forme des arbres.



(a) 0-500
(b) 500-1000
(c) 1000-2000
(d) 2000-5000
(e) 5000-10000
(f) 10000-15000
(g) 15000-20000
(h) 20000-30000
(i) 30000-50000
(j) 50000-90000



(k) 90000-160000

FIG. A.7: Distribution des degrés sur les différentes tranches de taille.

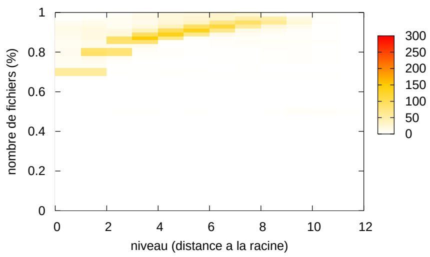


FIG. A.8: Composition des niveaux.

Bibliographie

[1] E. Adar and M. Kim. Softguess : Visualization and exploration of code clones in context. In *IEEE International Conference on Software Engineering (ICSE'07)*, pages 762–766, 20-26 May 2007.

[2] W. Aiello, F. Chung, and L. Lu. A random graph model for power law graphs. In *STOC '00 : Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 171–180, New York, USA, 2000. ACM press.

[3] W. Aiello, F. Chung, and L. Lu. *Handbook of massive data sets table of contents*, chapter Random evolution in massive graphs, pages 97–122. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[4] R. Albert and A.-L. Barabási. Topology of evolving networks : local events and universality. *Physical review letters*, 85(24) :5234–5237, 11 December 2000.

[5] D. Auber. *Graph drawing software*, chapter Tulip - A Huge Graph Visualization framework. Verlag, 2003.

[6] D. Auber, M. Delest, J.P. Domenger, and S. Dulucq. Efficient drawing and comparison of rna secondary structure. *Journal of Graph Algorithms and Applications*, 10(2) :329–351, 2006.

[7] D. Auber, M. Delest, J.P. Domenger, P. Ferraro, and R. Strandh. EVAT : Environment for visualization and analysis of trees. In *IEEE Symposition on Information Visualisation Contest*, volume www.cs.umd.edu/hcil/iv03contest/, pages 124–126, 2003.

[8] D. Auber, M. Delest, J.M. Fédou, J.P. Domenger, and P. Duchon. New strahler numbers for rooted plane trees. In M. Drmota, P. Flajolet, D. Gardy, and B. Gittenberger, editors, *Third Colloquium on Mathematics and Computer Science, Algorithms, Trees, Combinatorics and Probabilities*, Trends in Mathematics, pages 203–215. Vienna University of Technology, Birkhauser, 2004.

[9] L. Babai and E.M. Lucks. Canonical labeling of graphs. In *Proc. 15th ACM Symposium, Theory and Comp.*, pages 171–183, 1983.

[10] L. Babai and E.M. Luks. Canonical labeling of graphs. In *STOC*, pages 171–183. ACM, 1983.

[11] A.-L. Barabási. *Linked : The New Science of Networks.* Cambridge, perseus edition, 2002.

[12] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286 :509–512, 1999.

[13] A.-L. Barabási, R. Albert, and H. Jeong. Scale-free characteristics of random networks : the topology of the world-wide web. *Physica A*, 281 :69–77, 2003.

[14] A.-L. Barabási and Z.N. Oltvai. Network biology : Understanding the cell's functional organization. *Nature Reviews Genetics*, 5 :101–113, 2004.

[15] A.-L. BarabÃ¡si and E. Bonabeau. Scale-free networks. *Scientific American*, 288 :60–69, 2003.

[16] T. Barlow and P. Neville. A comparison of 2d visualizations of hierarchies. In *IEEE InfoVis*, pages 131–139, 2001.

[17] I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. Clone detection using abstract syntax trees. In *IEEE International Conference on Software Maintenance*, pages 368–377, Bethesda, MD, USA, 16-20 November 1998.

[18] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *ACM International Conference Proceeding Series, Proceedings of the 23rd international conference on Machine learning*, volume 148, pages 97–104, Pittsburgh, Pennsylvania, 2006. ACM Press.

[19] A. Del Bimbo. *Visual information retrieval*. Morgan Kaufmann Publishers Inc., 1999.

[20] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications*. Elsevier Science, 1976.

[21] H. Le Borgne and N. O'Connor. Natural scene classification and retrieval using ridgelet-based image signatures. In *Proc. of Advanced Concepts for Intelligent Vision Systems, 2005*, 2005.

[22] H.G. Borrow and R.M. Burstall. Subgraph isomorphism, matching relational structures and maximal cliques. *Information Processing Letters*, 4 :83–84, 1976.

[23] N. Boujemaa, J. Fauqueur, M. Ferecatu, F. Fleuret, V. Gouet, B. Le Saux, and H. Sahbi. Ikona for interactive specific and generic image retrieval. In *Proc. of MMCBIR 2001*, 2001.

[24] R. Breitling, M. Barrett, and D. Gilbert. Metanetter : inference and visualization of high-resolution metabolomic networks. *Bioinformatics*, 2007.

[25] L. Carminati, J.Benois-Pineau, and M. Gelgon. Human detection and tracking for video surveillance applications in low density environment. *SPIE VCIP'2003*, 5150 :51–60, 2003.

[26] C. Carson, S. Belongie, H. Greenspan, and J. Malik. Blobworld : Image segmentation using expectation-maximization and its application to image querying. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(8) :1026–1038, August 2002.

[27] S-F. Chang, W. Chen, H.J. Meng, H. Sundaram, and D. Zhong. VideoQ : an automated content based video search system using visual cues. In *Proc.of the fifth ACM Multimedia'97*, pages 313–324, 1997.

[28] F. Chevalier, D. Auber, and A. Telea. Structural analysis and visualization of c++ code evolution using syntax trees. In *9th International Workshop on Principles of Software Evolution (IWPSE'07)*, 2007.

[29] F. Chevalier, M. Delest, and J.P. Domenger. A heuristic for the retrieval of objects in low resolution video. In *5th International Workshop on Content Based Multimedia Indexing (CBMI'07)*, pages 144–151. IEEE, jun 2007.

[30] F. Chevalier, M. Delest, and J.P. Domenger. A heuristic for the retrieval of objects in video in the framework of the rough indexing paradigm. *Signal Processing : Image Communication on Content-Based Multimedia Indexing (SPIC)*, 22 :622–634, 2007.

[31] F. Chevalier, J.P. Domenger, J. Benois-Pineau, and M. Delest. Retrieval of objects in video by similarity based on graph matching. *Pattern Recognition Letters*, 28(8) :939–949, June 2007.

[32] D. Coleman, D. Ash, B. Lowther, and P. Oman. Using metrics to evaluate software system maintainability (abstract). *IEEE Computer*, 27(8) :44–49, 1994.

[33] CollabNet. Subversion. Internet address : http ://subversion.tigris.org/.

[34] C. Cooper and A. Frieze. A general model of web graphs. *Random Structures Algorithms*, 22 :311–335, 2003.

[35] D.G. Corneil and C.C. Gotlieb. An efficient algorithm for graph isomorphism. *Journal of the ACM*, 17 :51–64, 1970.

[36] F. Coudert, J. Benois-Pineau, and D. Barba. Dominant motion estimation and video partitionning with a 1d signal approach. In *SPIE VVDC*, volume 3527, pages 283–294, October 1998.

[37] Dailymotion. Dailymotion. Internet address : http ://www.dailymotion.com.

[38] R. Datta, D. Joshi, J. Li, and J.Z. Wang. Image retrieval : Ideas, influences, and trends of the new age. Technical report, ACM Computing Surveys, under final revision, 2007. Also Penn State University Technical Report CSE 06-009, 2006.

[39] P. de Reffye, C. Edelin, con J. Fran M. Jaeger, and C. Puech. Plant models faithful to botanical structure and development. In *15th annual conference on Computer graphics and interactive techniques (SIGGRAPH'88)*, volume 22(4), pages 151–158. ACM press, August 1988.

[40] M. F. Demirci, A. Shokoufandeh, Y. Keselman, L. Bretzner, and S. Dickinson. Object recognition as many-to-many feature matching. *International Journal of Computer Vision*, 69(2) :203–222, August 2006.

[41] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, third edition, 2005.

[42] Y. Dinitz, A. Itai, and M. Rodeh. On an algorithm of Zemlyachenko for subtree isomorphism. *Information Processing Letters*, 703 :141–146, 1999.

[43] A. Don. *Indexation et navigation dans les contenus visuels : approches basÃ©es sur les graphes*. PhD thesis, University of Bordeaux 1, 2006.

[44] E. Duala-Ekoko and M.P. Robillard. Tracking code clones in evolving software. In *IEEE International Conference on Software Engineering (ICSE'07)*, pages 158–167, 20-26 May 2007.

[45] S. Ducasse, M. Rieger, and S. Demeyer. A language independent approach for detecting duplicated code. In *IEEE International Conference on Software Maintenance*, page 109, 1999.

[46] P. Erdös and A. Rényi. On random graphs i. *Publ. Math.*, 9 :290–297, 1959.

[47] B. Erol and F. Kossentini. Retrieval of video objects by compressed domain shape features. In *The 7th IEEE International Conference on Electronics,Circuits and Systems*, volume 2, pages 667–670, Jounieh, Lebanon, 17-20 December 2000.

[48] B. Erol and F. Kossentini. Color content matching of mpeg-4 video objects. In Springer-Verlag, editor, *PCM '01 : Proceedings of the Second IEEE Pacific Rim Conference on Multimedia*, pages 891–896, 2001.

[49] A.P. Ershov. On programming of arithmetic operations. *Communication of the A.C.M*, 1(8) :3–6, 1958.

[50] R. Falke, P. Frenzel, and R. Koschke. Clone detection using abstract syntax suffix trees. In *13th Working Conference on Reverse Engineering*, pages 253–262, 2006.

[51] J.M. Fédou. Nombre de strahler sur les arbres généraux. In GDR ALP, editor, *Ecole jeunes chercheurs en algorithmique et calcul formel*, May 1999.

[52] J.D. Fekete and C. Plaisant. Infovis contest 2003 - visualization and pair wise comparison of trees. In IEEE, editor, *IEEE Symposium on Information Visualization*, volume www.cs.umd.edu/hcil/iv03contest/, 2003.

[53] B. Fenner and H. Zeller. Cvsview. Internet address : http ://www.viewvc.org/, 2001.

[54] M. Flickner, H. Sawhney, W. Niblack, and al. Query by image and video content : The qbic system. *IEEE Computer*, 28(9) :23–32, September 1995.

[55] Flickr. Flickr. Internet address : http ://www.flickr.com/, 2006.

[56] C. Fogg, D.J. LeGall, J.L. Mitchell, and W.B. Pennebaker. *MPEG Video Compression Standard*. Springer, 1996.

[57] C. Gomila and F. Meyer. Graph-based object tracking. In *IEEE International Conference on Image Processing (ICIP)*, volume 2, pages 41–44, 2003.

[58] D. Grune and B. Berliner. Concurrent version system. Internet address : http ://www.nongnu.org/cvs/.

[59] V. N. Gudivada and V. V. Raghavan. Content-based image retrieval systems. *Computer*, 28(9) :18–22, 1995.

[60] A. Hauptmann and M. Smith. Text, speech, and vision for video segmentation : The Informedia project. In *Proc. of AAAI Fall 1995 Symposium on Computational Models for Integrating Language and Vision*, 1995.

[61] J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal of Computing*, 2 :225–231, 1973.

[62] J.E. Hopcroft and R.E. Tarjan. Isomorphism of planar graphs. In *Complexity of Computer Computations*, pages 131–152, 1972.

[63] R. Horaud and T. Skordas. Stereo correspondence through feature grouping and maximal cliques. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(11) :1168–1180, 1989.

[64] P. Howarth and S. Rüger. Evaluation of texture features for content-based image retrieval. In *Third International Conference on Image and Video Retrieval*, pages 326–334, Dublin, Ireland, July 21-23 2004.

[65] T. Huang and Y. Rui. Image retrieval : past, present and future. In *International Symposium on Multimedia Information Processing*, pages 1–23, 1997.

[66] B. Huet and E.R. Hancock. Inexact graph retrieval. In *IEEE CVPR99 Workshop on Content-based Access of Image and Video Libraries (CBAIVL-99)*, pages 40–44, Fort Collins, Colorado USA, June 22 1999.

[67] B. Huet, N.Kern, G. Guarascio, and B. Merialdo. Relational skeletons for retrieval in patent drawings. In *Proc. of ICIP'01*, volume 2, pages 737–740, 2001.

[68] R. Hummel and S. Zucker. On the foundations of relaxation labeling processes. *IEEE Trasaction of Pattern Analysis and Machine Intelligence*, 5 :267–287, 1983.

[69] I.B.M. Clearcase. Internet address : http ://www-306.ibm.com/software/awdtools/clearcase/.

[70] IBM. QBIC at hermitage museum. Internet address : http ://www.hermitagemuseum.org/, 2001.

[71] K. Idrissi, G. Lavoué, J. Ricard, and A. Baskurt. Object of interest-based visual navigation, retrieval, and semantic content identification system. *Computer Vision and Image Understanding*, 94 :271–294, 2004.

[72] A.K. Jain and F. Farrokhnia. Unsupervised texture segmentation using gabor filters. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 14–19, Los Angeles, USA, 4-7 Nov 1990.

[73] L. Jiang, G. Misherghi, Z. Su, and S. Glondu. Deckard : Scalable and accurate tree-based detection of code cones. In *IEEE 29th International Conference on Software Engineering (ICSE)*, pages 96–105, Minneapolis, MN, USA, 20-26 May 2007.

[74] P. Joly and K-K. Kim. *Efficient Automatic Analysis of Camera Work and Microsegmentation of Video Using Spatio-Temporal Images*, volume 8(4) of *Signal Processing : Image Communication*, pages 295–307. Elsevier, Eurasip, Amsterdam, mai 1996.

[75] Y. Keselman and S. Dickinson. Generic model abstraction from examples. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 27(7) :1141–1156, July 2005.

[76] J.M. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A.S. Tomkins. The web as a graph : measurements, models, and methods. *Lecture Notes in Computer Science*, 1627 :1–17, 1999.

[77] S.H. Kwok and J. Leon Zhao. Content-based object organization for efficient image retrieval in image databases. *Decision Support Systems*, 42 :1901–1916, 2006.

[78] C. Langreiter. Retrievr. Internet address : http ://labs.systemone.at/retrievr/, 2006.

[79] E. Lee and J. MacGregor. Minimizing user search time in menu retrieval systems. *Human Factors*, 27 :157–162, 1985.

[80] J. Li, J.Z. Wang, and G. Wiederhold. Irm : Integrated region matching for image retrieval. In *Proceedings of the Eighth ACM Multimedia Conference*, pages 147–156, 2000.

[81] J. Lladós, E. Martí, and J.J. Villanueva. Symbol recognition by error-tolerant subgraph matching between region adjacency graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10) :1137–1143, 2001.

[82] G. Lommerse, F. Nossin, L. Voinea, and A. Telea. The visual code navigator : An interactive toolset for source code investigation. In *IEEE Infovis*, pages 4–11, 2005.

[83] E.M. Lucks. Isomorphism of graphs of bounded valence can be tested in polynomial time. In *Proc. 21st IEEE FOCS Symposium*, pages 42–49, 1980.

[84] E.M. Lucks and C.M. Hoffman. An $o(n^3 \log(n)$ deterministic and an $o(n^3)$ las vegas isomorphism test for trivalent graphs. *Journal of the ACM*, 34 :513–531, 1987.

[85] J. Luo and A.E. Savakis. Indoor vs outdoor classification of consumer photographs using low-level and semantic features. In *Proc. of ICIP'01*, volume 2, pages 745–748, 2001.

[86] Y. Luo, T.-D. Wu, and J.-N. Hwang. Object-based analysis and interpretation of human motion in sports video sequences by dynamic bayesian networks. *Computer Vision and Image Understanding*, 92 :196–216, 2003.

[87] F. Manerba, J. Benois-Pineau, and R. Leonardi. Real-time rough extraction of foreground objects in mpeg1,2 compressed video. In *Proc. Workshop on Image Analysis For Multimedia Interactive Services (WIAMIS)*, Montreux, Switzerland, April 2005.

[88] B. Manjunath, P. Salembier, and T. Sikora, editors. *Introduction to MPEG-7*. Wiley, 2002.

[89] J. Mayrand, C. Leblanc, and E. Merlo. Experiment on the automatic detection of function clones in a software system using metrics. In *International Conference on Software Maintenance*, page 244, 1996.

[90] T. Meiers, T. Sikora, and I. Keller. Hierarchical image database browsing environment with embedded relevance feedback. In *Proc. of ICIP*, volume 2, pages 593–596, 2002.

[91] B.T. Messmer and H. Bunke. A new algorithm for error-tolerent subgraph isomorphism detection. *IEEE Trasaction of Pattern Analysis and Machine Intelligence*, 20(5) :493–504, 1998.

[92] P.A. Mlsna and N.M. Sirakov. Intelligent shape feature extraction and indexing for efficient content-based medical image retrieval. In *Proc. of SouthWest'04*, pages 172–176, 2004.

[93] A. Mojsilovic and B. Rogowitz. Capturing image semantics with low-level descriptors. In *Proc. of ICIP'01*, volume 1, pages 18–21, 2001.

[94] M. Molloy and B. Reed. A critical point for random graphs with a given degree sequence. *Random Structures and Algorithms*, 6(2-3) :161–179, 1995.

[95] T.F. Móri. On random trees. *Studia Sci. Math. Hungar.*, 39 :143–155, 2003.

[96] T.F. Móri. The maximum degree of the barabási-albert random tree. *Combinatorics, Probability and Computing*, 14 :339–348, 2005.

[97] M. Newman, A.-L. Bababási, and D.J. Watts, editors. *The Structure and Dynamics of Networks*. Princeton University Press, 2006.

[98] M.E.J. Newman. Scientific collaboration networks : I. network construction and fundamental results. *Physical Revue E*, 64 :016131, 2001.

[99] M.E.J. Newman. Scientific collaboration networks : Ii. shortest paths, weighted networks, and centrality. *Physical Revue E*, 64 :016132, 2001.

[100] M.E.J. Newman. *Handbook of Graphs and Networks*, chapter Random graphs as models of networks. S. Bornholdt and H. G. Schuster (eds.), Wiley-VCH, Berlin, 2003.

[101] M.E.J. Newman. Mixing patterns in networks. *Physical Revue E*, 67 :026126, 2003.

[102] N.J. Nilsson. *Principles of Artificial Intelligence*. Palo Alto, California : Tioga Publishing Company, 1980.

[103] E. Osuna, R. Freund, and F. Girosi. Training Support Vector Machines : an application to face detection. In *Proc. of IEEE CVPR 1997*, pages 130–136, 1997.

[104] C. Papadimitriou. *Computational complexity.* Addison Wesley, 1994.

[105] A. Pentland, R. Picard, and S. Sclaroff. Photobook : Content-based manipulation of image databases. *International Journal of Computer Vision*, 18(3) :233–254, 1996.

[106] Perforce Software. Perforce. Internet address : http ://www.perforce.com/.

[107] S.L. Pfleeger and J.M Atlee. *Software Engineering : Theory and Practice.* Prentice Hall, 2005.

[108] W. W. Powell, D. R. White, K. W. Koput, and J. Owen-Smith. Network dynamics and field evolution : The growth of interorganizational collaboration in the life sciences. *American Journal of Sociology*, 110 :1132–1205, 2005.

[109] P. Prusinkiewicz. Modeling of spatial structure and development of plants : a review. *Scientia Horticulturae*, 1-2 :113–149, April 1998.

[110] J. Puzicha, Y. Rubner, C. Tomasi, and J.M. Buhmann. Empirical evaluation of dissimilarity measures for color and texture. In *IEEE International Conference on Computer Vision*, pages 1165–1172, Kerkyra, Corfu, Greece, 20-25 September 1999.

[111] A. Rosenfeld, R. Hummel, and S. Zucker. Scene labeling by relaxation operators. *IEEE Transaction of Systems, Man and Cybernetics*, 6 :420–453, 1976.

[112] Y. Rubner, C. Tomasi, and L.J. Guibas. A metric for distributions with applications to image databases. In *IEEE Proceedings of the Sixth International Conference on Computer Vision table of contents*, page 59, 1998.

[113] Y. Rui, A. She, and T. Huang. Modified fourier descriptors for shape representation – a practical approach. In *First International Workshop on Image Databases and Multi Media Search*, 1996.

[114] E. Saykol, U. Güdükbay, and Ö. Ulusoy. A histogram-based approach for object-based query-by-shape-and-color in image and video databases. *Image and Vision Computing*, 23(13) :1170–1180, 2005.

[115] M. Seo, B. Ko, H. Chung, and J. Nam. ROI-based medical image retrieval using human-perception and MPEG-7 visual descriptors. In *Proc. of CIVR'06*, pages 231–240, 2006.

[116] L.G. Shapiro and R.M. Haralick. Structural descriptions and inexact matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3 :504–519, 1981.

[117] L.G. Shapiro and R.M. Haralick. A metric for comparing relational descriptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(1) :90–94, 1985.

[118] A. Shokoufandeh, I. Marsic, and S. Dickinson. View-based object recognition using saliency maps. *Image and Vision Computing*, 17(5-6) :445–460, 1999.

[119] J. Sivic, F. Schaffalitzky, and A. Zisserman. Object level grouping for video shots. *International Journal of Computer Vision*, 67(2) :189–210, 2006.

[120] J.R. Smith and S.-F. Chang. Visualseek : A fully automated content-based image query system. In *ACM Multimedia*, pages 87–98, Boston, 1996.

[121] J.R. Smith and S-F. Chang. An image and video search engine for the world-wide web. In *Proc. of SPIE Electronic Imaging*, volume 5, San Jose, CA, February 1997.

[122] R. Solomonoff and A Rapoport. Connectivity of random nets. *Bulletin of Mathematical Biophysics*, 13 :107–117, 1951.

[123] A.N. Strahler. Hypsomic analysis of erosional topography. *Bulletin Geological Society of America*, 63 :1117–1142, 1952.

[124] H. Tamura, S. Mori, and T. Yamawaki. Textural features corresponding to visual perception. *IEEE Transaction on Systems, Man and Cybernetics*, 8(6) :460–472, June 1978.

[125] B. Tjaden. Hollywood collaboration graph. Internet address : http ://oracleofbacon.org/.

[126] J. Travers and S. Milgram. An experimental study of the small world problem. *Sociometry*, 32 :425–443, 1969.

[127] J.R. Ullman. An algorithm for subgraph isomorphism. *Journal of the Association for Computing Machinery*, 9 :31–42, 1976.

[128] A. Vailaya, H.J. Zhang, C. Yang, F.I. Liu, and A.K. Jain. Automatic image orientation detection. In *ICIP99*, volume 2, pages 600–604, 1999.

[129] R.C. Veltkamp and M. Tanase. Content-based image retrieval systems : A survey. Technical report, Institute of Information and Computing Sciences, Utrecht University, 2000. Internet address : http ://give-lab.cs.uu.nl/cbirsurvey/cbir-survey.pdf.

[130] G. Viennot. Trees everywhere. In A. Arnold, editor, *Colloquium on Trees in Algebra and Programming*, volume 431 of *Lecture Notes in Computer Science*, pages 18–41. Springer, 1990.

[131] G. Viennot, G. Eyrolles, N. Janey, and D. ArquÃ¨s. Combinatorial analysis of ramified patterns and computer imagery of trees. In *16th annual conference on Computer graphics and interactive techniques (SIGGRAPH'89)*, volume 23(3), pages 31–40, Boston, July 1989. ACM press.

[132] S. Virtanen. Properties of nonuniform random graph models. Research Report A77, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, May 2003.

[133] J. Vleugels and R.C. Veltkamp. Efficient image retrieval through vantage objects. *Pattern Recognition*, 35(1) :69–80, 2002.

[134] L. Voinea and A. Telea. Multiscale and multivariate visualizations of software evolution. In *ACM symposium on Software visualization (SoftVis'06)*, pages 115–124, 2006.

[135] L. Voinea and A. Telea. Visualizing debugging activity in source code repositories. In *4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pages 156–157, Banff, Canada, 24-25 June 2007.

[136] L. Voinea, A. Telea, and J.J. van Wijk. Cvsscan : visualization of code evolution. *Software Visualization*, pages 47–56, 2005.

[137] V. Wahler, D. Seipel, J.W. v. Gudenberg, and G. Fisher. Clone detection in source code by frequent itemset techniques. In *Fourth IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'04)*, pages 128–135, 15-16 Sept. 2004.

[138] J. Z. Wang, J. Li, and G. Wiederhold. Simplicity : Semantics-sensitive integrated matching for picture libraries. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 23(9) :947–963, 2001.

[139] C. Ware. *Information Visualization : Perception for design*. Interactive Technologies. Margn Kaufmann, 2000.

[140] R.C. Wilson and E.R. Hancock. Structural matching by discrete relaxation. *IEEE Trasaction of Pattern Analysis and Machine Intelligence*, 19(6) :634–648, 1997.

[141] J.K Wu, C.P. Lam, B.M. Mehtre, Y.J. Gao, and A.D. Narasimhalu. Content-based retrieval for trademark registration. *MTAP*, 3(3) :245–267, 1996.

[142] W. Xiong, B. Qiu, Q. Tian, C. Xu, S.H. Ong, and K. Foong. Content-based medical image retrieval using dynamically optimized regional features. In *Proc. of ICIP'05*, volume 3, pages III : 1232–1235, 2005.

[143] X. Yan, F. Zhu, J. Han, and P.S. Yu. Searching substructures with superimposed distance. In *ICDE '06. Proceedings of the 22nd International Conference on Data Engineering*, pages 88–97, 03-07 April 2006.

[144] W.W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33 :452–473, 1977.

[145] D. Zeilberger. A bijection between ordered trees to binary trees that sends the prunning order to the strahler number. *Discrete Math*, 82 :89–92, 1990.

[146] V.N. Zemlyachenko. Determining tree isomorphism. *Seminar on Combinatorial Mathematics*, pages 54–60, 1971.

[147] D. Zhang and G. Lu. Review of shape representation and description techniques. *Pattern Recognition*, 1 :1–19, January 2004.

[148] G. K. Zipf. Huma behaviour and the principle of least effort. *New York : Hafner*, 1949.