Navigation dans les grands graphes

Nicolas Hanusse

Document de synthèse de l'activité scientifique
Habilitation à diriger les recherches

Navigation dans les grands graphes

Nicolas Hanusse

26 novembre 2009

Avant-propos

Ce document présente un panorama de mes activités de recherche effectuées aprés ma thèse ayant porté sur la *combinatoire* et la *théorie des cartes*. A partir de 1999, j'ai commencé à m'orienter vers l'*algorithmique distribuée*. Début 2003, j'ai considéré que des techniques et résultats d'algorithmique distribuée et d'*algorithmique de graphes* pouvaient être utiles dans la thématique de la *visualisation d'information*. Cette ouverture thématique et mes nouveaux objectifs ont nécessité un apprentissage des domaines tels que la *fouille de données* et l'indexation vidéo, pour lesquels j'ai pu récolter mes premiers fruits.

Mon objectif n'est pas d'être exhaustif mais d'être synthétique. Dans la sélection de résultats que je présente, j'essaye de montrer la variété des différentes techniques utilisées qui vont de la combinatoire (bijective, énumérative) à l'analyse d'algorithmes probabilistes (chaînes de Markov) en passant par la théorie des graphes (décomposition, plongements de graphes).

Table des matières

1 Introduction **4**
- 1.1 Présentation générale . . . 4
- 1.2 Quelques notations . . . 6
- 1.3 Mise en perspective . . . 7

2 Codage compact de graphes et application au routage compact **10**
- 2.1 Quelques mots sur le codage compact et les requêtes locales . . . 11
- 2.2 Graphes planaires . . . 12
  - 2.2.1 Un bref tour d'horizon . . . 12
  - 2.2.2 Principal résultat . . . 14
  - 2.2.3 un résumé de la méthode . . . 14
  - 2.2.4 Un peu de théorie des graphes : Arbres de Schnyder . . . 15
- 2.3 Requêtes rapides dans les chaînes multi-parenthésées compressées . . . 17
- 2.4 Graphes k-pages . . . 17
  - 2.4.1 Perspectives concernant le codage compact . . . 20
- 2.5 Application au routage compact pour les graphes de genre borné . . . 20
  - 2.5.1 Routage, étirement et latence . . . 20
  - 2.5.2 Notre contribution . . . 21
  - 2.5.3 Techniques diverses . . . 21
  - 2.5.4 Évolution du routage compact . . . 22
- 2.6 Cartes planaires extérieures . . . 22

3 Navigabilité des graphes **25**
- 3.1 Graphes navigables . . . 26
  - 3.1.1 Algorithmes gloutons de navigation . . . 26
  - 3.1.2 Graphes navigables . . . 26
  - 3.1.3 Une motivation : réseau logique pour réseaux pair-à-pair . . . 28
  - 3.1.4 Graphes aléatoires navigables . . . 28
  - 3.1.5 Enjeu . . . 29
- 3.2 Petit-monde . . . 29
  - 3.2.1 L'expérience de Milgram . . . 29
  - 3.2.2 Un premier modèle de petit-monde navigable . . . 30
  - 3.2.3 Petit-mondisation de graphes à croissance bornée . . . 31

3.2.4 Petit-mondisation rapide et distribuée . . . 32
3.2.5 Un réseau logique intermédiaire . . . 33
3.2.6 Perspectives : Émergence et fiabilité des petits mondes . . . 35
3.3 Modèles de graphes non navigables . . . 35
3.3.1 Modèles de graphes sans échelle . . . 37
3.3.2 Indistinguabilité des noeuds . . . 38

4 Contribution à la recherche d'information 40
4.1 Un modèle de recherche par agents mobiles pour un réseau non fiable . . . 41
4.1.1 Contexte . . . 41
4.1.2 Un modèle de réseau incertain . . . 41
4.1.3 Résumé des résultats . . . 42
4.1.4 " Expanders " . . . 42
4.1.5 Nouveaux résultats . . . 42
4.1.6 Vers des graphes de navigation tolérant des erreurs . . . 43
4.2 Application à la recherche d'images dans une collection indexée . . . 44
4.2.1 Contexte . . . 44
4.2.2 Limitations des systèmes de recherche existants . . . 45
4.2.3 Recherche d'images par l'approche locale . . . 46
4.2.4 Expérimentations . . . 47
4.2.5 Un algorithme de réduction de dimension pour l'exploration d'une collection d'images . . . 50
4.2.6 Perspective : vers plus d'intéraction. . . . 54
4.3 Vers des requêtes plus complexes ... celles des bases de données . . . 54
4.3.1 Optimisation de requêtes . . . 56
4.3.2 Un algorithme parallèle de calcul d'itemsets fréquents . . . 58
4.3.3 Perspectives en distribué . . . 59

5 Autres contributions à l'algorithmique distribuée 61
5.1 Diffusion d'information épidémique . . . 61
5.1.1 Diffusion épidémique dans le modèle rendez-vous . . . 62
5.1.2 Perspective : Diffusion dans les graphes aléatoires k-out . . . 64
5.2 Exploration perpétuelle . . . 64
5.2.1 Algorithmes d'exclusion mutuelle probabilistes auto-stabilisants . . . 65
5.2.2 Exploration eulérienne . . . 67

Chapitre 1

Introduction

1.1 Présentation générale

Le concept fédérateur de ma recherche est que les requêtes dans divers domaines peuvent s'exprimer comme une navigation dans un graphe. Le graphe étiqueté est une abstraction de réseaux au sens large (informatique, social, ...) pour lequel les étiquettes représentent des informations distribuées sur le réseau. L'objectif est de caractériser les modèles pour lesquels les informations glanées lors de la navigation permettent ou non de répondre efficacement à des requêtes par un agent logiciel ou un humain.

Dans un ordre croissant de complexité, je me suis intéressé aux requêtes suivantes :

- la *déterminisation de caractéristiques locales* du réseau : test d'adjacence entre deux noeuds, degré d'un noeud, ...
- le *routage* : comment aller vers un noeud dont l'adresse (identifiant) est *connue* ;
- la *diffusion* d'une information aux membres d'un réseau ou sous-réseau ;
- la *recherche d'information (localisation de ressources)* ou comment aller vers un noeud dont l'adresse est *inconnue*. Typiquement, comment retrouver dans son arborescence une image dont on a oublié sa localisation et son nom ! Cependant, j'ai aussi considéré des requêtes plus complexes utilisables en bases de données (requêtes de projection et regroupement).

Un bon nombre de mes résultats consiste en l'optimisation d'algorithmes prouvés de manière théorique mais aussi en des expérimentations réelles dans le domaine de la recherche d'information.

Les mesures de performance des algorithmes étudiées portent essentiellement sur **le temps** (temps de recherche, de pré-calcul de structure de données, de mise-à-jour, ...), la **quantité de mémoire** nécessaire pour réaliser une tâche (stockée aux noeuds, d'un agent mobile, taille de messages, ...) et la **qualité** de la réponse obtenue (longueur des routes par rapport à l'optimal pour le routage, mesures rappel/précision en recherche d'information, ...). Les résultats s'expriment essentiellement en des analyses en moyenne ou de pire cas des complexités en donnant des formules asymptotiques ou des majorants/minorants sur les différentes mesures. D'un point de vue qualitatif, mon travail tourne autour de **la navigabilité**, c'est-à-dire la possibilité ou l'impossibilité de trouver une information

avec différentes contraintes. C'est le problème clef de ma recherche.

Dans mon travail, j'ai considéré le cadre, à la fois général et particulier, de l'algorithmique distribuée. Je me suis focalisé sur des modèles ayant des contraintes standards de l'algorithmique distribuée : tout algorithme exécuté à un noeud part d'un état de connaissance *partiel* du réseau. Dans mes travaux, on peut distinguer différentes catégories de contraintes et d'objectifs :

- **Connaissance** : on veut construire de manière efficace une structure de donnée distribuée en partant ou non d'une connaissance totale du réseau ;
- **Adversité** : on analyse le déroulement de requêtes sur une structure de donnée distribuée qui n'est pas nécessairement fiable.

En pratique, dans les grands réseaux, la fiabilité de l'information n'est pas assurée pour différentes raisons : la caractéristique de grande taille implique une existence réelle d'erreurs ou de pannes, la dynamicité empêche la pertinence d'une information, la difficulté d'indexer automatiquement de manière non ambiguë des données complexes (images, documents multi-média), ... Ainsi, les sections 4 et 5 traitent de modèles et d'expériences liés à cette réalité. D'un point de vue formel, la fiabilité peut être mesurée par l'adversité [1]. Pour une requête donnée, la performance d'un algorithme dépend d'un modèle d'adversaire. Le processus de traitement est vu comme un jeu à deux joueurs : d'un côté l'algorithme et de l'autre l'adversaire dont l'objectif est de faire échouer la requête ou de dégrader au maximum les performances de l'algorithme. Par exemple, dans le contexte d'information fiable, l'adversaire est considéré comme inexistant ou nul. À l'autre extrémité, un adversaire fort représente un adversaire omniscient [2] et puissant (pouvant répartir erreurs/imprécisions et informations fiables à sa guise). La définition du modèle d'adversaire dépendra du contexte.

Un des objets récurrent dans ma recherche est la famille des graphes planaires : cette famille a été extrêmement étudiée en théorie des graphes et des cartes. Dans le cadre de requêtes présentées précédemment, l'usage de graphes planaires est très restrictif. Cependant, c'est un excellent point de départ d'un point de vue théorique. Ma première contribution à l'algorithme distribuée traite ainsi du problème de la construction de tables de routage compact dédiées aux graphes planaires et à ses généralisations (graphes de genre borné). Les techniques utilisées sont typiques de la combinatoire et de la théorie des graphes. Concernant les graphes planaires et ses variantes, je me suis concentré sur le codage compressé, les requêtes efficaces sans décodage, la génération aléatoire, ... Tous ces problèmes sont à résoudre également dans un cadre général avec comme finalité la conception distribuée de graphes de navigation légers en mémoire.

Idéalement, nos solutions devraient être performantes dans des modèles de connaissance les plus faibles, les moins fiables et dans un cadre le plus général possible. La figure 1.1 propose une représentation abstraite idéale - et donc imparfaite - du triptyque connaissance, généricité [3] de la solution et adversité et de sa relation avec les perfor-

1. Cette terminologie est personnelle.

2. connaissant à l'avance les éventuels tirages aléatoires et les états passés et futurs du système.

3. Une solution est *universelle* si elle s'applique indépendamment de la topologie. Dans le cas contraire, elle est *dédiée* à une famille de graphes ou réseaux.

mances d'un algorithme. Le plan de ce document suit, de manière détournée et partielle, un cheminement vers cet idéal. La notion de connaissance initiale lors du lancement d'un algorithme est particulièrement pertinente dans le cadre d'un système distribué. Elle est dite *locale* si chaque nœud d'un réseau n'a comme seule connaissance que son voisinage. Un algorithme distribué universel qui fonctionne efficacement avec une connaissance initiale locale et tolérant un degré d'adversité est un très bon candidat pour un usage réel pour un réseau dynamique. L'axe "généricité" peut être a la fois vu comme une caractéristique de la solution (fonctionnant indépendamment ou non de la topologie) mais aussi du contexte. Une solution universelle pourrait ne pas être efficace pour certaines topologies alors qu'une solution dédiée le serait. Cette représentation n'a donc pas de grande ambition mais permet de comparer les contextes des différentes requêtes.

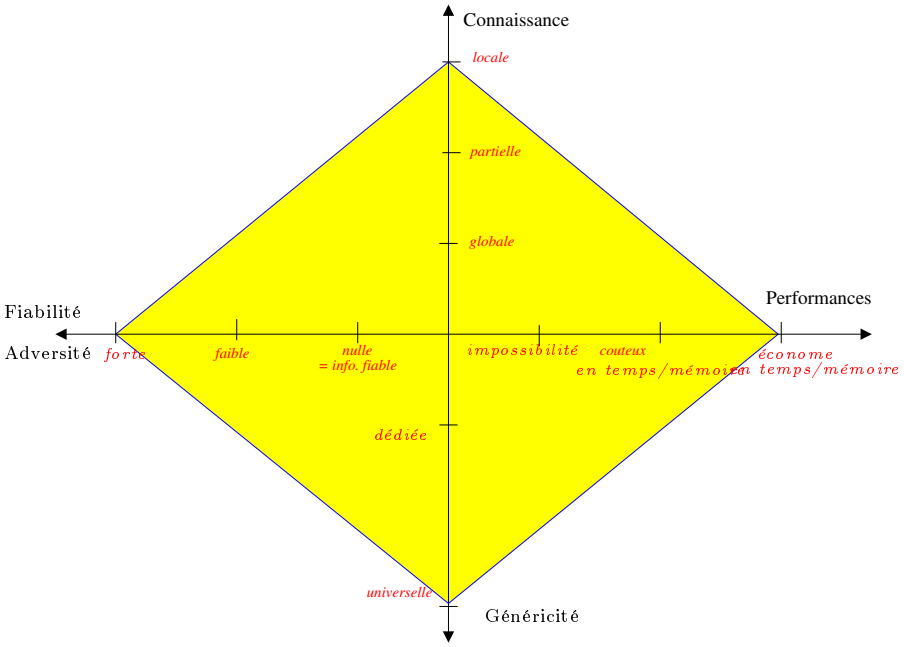


FIGURE 1.1 – Caractérisation des contextes et performances des requêtes.

1.2 Quelques notations

Si cela n'est pas précisé, $G = (V, E)$ désigne un graphe arbitraire décrit par un ensemble de sommets V et d'arêtes E. $n = |V|$ et $m = |E|$ représentent le nombre de sommets et d'arêtes du graphe. Si le graphe est étiqueté, chaque sommet u possède un identifiant $id(u)$. Si les identifiants sont fixés à l'avance sans possibilité de modification, le modèle est dit *indépendant des noms*. L'autre modèle *avec renommage* permet une première opération de renommage mais selon le contexte, ce modèle n'est pas toujours envisageable. Par défaut, nous supposons dans ce qui suit que les sommets sont identifiés et numérotés entre 1 et n.

La boule $B_u(r)$ est constituée de l'ensemble des sommets à distance au plus r du sommet u. On note $b_u(r)$ sa cardinalité. Selon le contexte, les éléments du graphe sont pondérés ou non. De manière générale, $d(u, v)$ désigne la distance entre les sommets u et v. En règle générale, D désigne le diamètre du graphe (pondéré ou non) et Δ le degré maximal.

1.3 Mise en perspective

Voici un scénario un peu artificiel pour justifier ma démarche globale. Bien que le scénario décrit puisse être vu comme une application bien réelle, l'historique de mon travail n'a pas été guidé initialement par cette application et je me suis davantage concentré sur l'étude et la proposition de modèles que sur des expérimentations réelles. Cependant, bon nombre des techniques et résultats théoriques obtenus dans mon travail peuvent servir dans ce cadre.

Imaginons un réseau de nombreux utilisateurs disposant chacun d'un carnet d'adresse d'utilisateurs. Chaque utilisateur stocke une collection de documents vidéos. A tout instant, un utilisateur peut être connecté (ou non) au réseau et rendre accessible ses vidéos à l'ensemble du réseau. Aucune infrastructure centralisée n'étant supposée, cette description est typiquement celle d'un réseau pair-à-pair dédié à des documents vidéos. L'ensemble des vidéos présentes dans le réseau peut être vu comme une gigantesque base de données en perpétuelle évolution. L'utilisateur a la possibilité de modifier sa collection et d'effectuer des requêtes de différents types : accéder à la collection d'un utilisateur (routage), rechercher des vidéos (recherche d'information), effectuer des requêtes complexes (statistiques, fouille de données), explorer des résumés d'ensembles de vidéos avec des interactions simples (visualisation d'information).

La succession des problèmes à résoudre est la suivante :

- **indexation des vidéos** : Ma contribution dans ce domaine est mineure. Á des fins de classification, des attributs nommés *descripteurs* sont calculés pour chaque vidéo. Lors d'expérimentations, nous avons été confrontés à la nécessité de manipuler des descripteurs de haut niveau, sémantiques et facilement interprétables par un humain (nombre de visages, objets, couleurs dominantes, ...). Nous avons donc dû développer des descripteurs ad-hoc interprétables et définir des mesures de similarité entre descripteurs à l'aide de techniques d'étalonnage de mesures psychovisuelles. De manière générale, le fossé sémantique entre mesures de descripteurs et perception humaine implique que l'on peut considérer que les indexations des vidéos ne sont pas totalement fiables. Quelles que soient les structures de données et mécanismes de recherche, il faut en tenir compte.
- **diffusion d'information/mise-à-jour dans le réseau dynamique pour la création/maintenance dynamique d'une structure de données distribuée d'index** : L'avènement récent des réseaux pair-à-pair a permis de proposer de nombreuses solutions pour garantir la connexité du réseau dynamique entre utilisateurs et la réponse à des requêtes simples de type routage (accéder à la collection d'un utilisateur) ou recherche d'information (une image dont on connaît la clef d'in-

dex). Typiquement, les solutions basées sur les tables de hachage distribué (DHT) consistent à affecter des index aux données (vidéos) et des identifiants virtuels aux nœuds (utilisateurs). Pour les deux entités, les valeurs sont issues du même univers de clefs. Chaque nœud est responsable des données dont les clefs d'index sont *proches* de sa propre clef. Dans le cadre de l'application décrite, les solutions existantes ne sont pas adaptées : les requêtes sont très simples, exactes et pré-supposent que les données sont indexées de manière fiable. Un certain nombre de mes travaux portent sur cette partie :

  - *Tolérance aux erreurs* : Pour tenir compte des possibles indexations incomplètes, inexactes, du fossé sémantique, il est nécessaire de considérer des structures de données et des mécanismes de recherche qui tiennent compte du caractère incertain de l'indexation ;
  - *diffusion épidémique de messages* : Pour permettre d'accéder à des requêtes complexes, il peut être nécessaire de diffuser des messages à un sous-réseau de manière efficace. Dans le cadre des réseaux dynamiques, les mécanismes de diffusion les plus pertinents sont des processus épidémiques (en sélectionnant des destinataires de manière aléatoire) ;

- **Résolution des requêtes par un agent mobile** : Des agents mobiles de recherche peuvent être utilisés pour extraire des sous-collections de vidéos ou faire des analyses statistiques sur les caractéristiques des vidéos. Bien entendu, le temps de réponse est un facteur très important mais la quantité de mémoire nécessaire « qu'embarque » un agent mobile est crucial.
  - *Compromis temps/mémoire* : Pour trouver de bons compromis entre mémoire et temps, aussi bien pour des étapes de pré-calcul de structures de données que pour les étapes de requêtes, il est nécessaire de faire une pré-sélection des éléments à calculer de la base de données globale : c'est typiquement le problème de la sélection et matérialisation de vues, déjà étudié dans le domaine des bases de données. Nous avons proposé un éclairage nouveau sur ce problème.
  - *Conception de graphes navigables* : Pour certaines requêtes, ma proposition principale consiste à raisonner sur un réseau logique construit sur les données et non sur les nœuds du réseau. L'objectif est de construire un graphe de navigation dont les sommets sont les données et les arêtes sont définies de manière à garantir la connexité du graphe et des requêtes de type routage. Dans ce cadre, j'ai étudié les aspects algorithmiques des graphes petit-monde (possédant de nombreuses caractéristiques telles que degré moyen faible, petit diamètre, ...). J'ai proposé avec d'autres auteurs des constructions centralisées et distribuées de petit-mondes et de spanners géométriques qui sont de bons candidats pour des graphes de navigation.
- **Représentation des résultats et interaction** : Une fois une sous-collection de vidéos extraites, comment les représenter et comment interagir pour affiner ou modifier sa recherche ? La solution naturelle consiste à lister le résultat de la recherche en alignant des images ou vignettes résumant chaque vidéo. Cependant, dès que la liste est un peu longue (quelques dizaines de vidéos), cette représentation est

fastidieuse. Typiquement, les données sont souvent décrites comme des vecteurs multi-dimensionnels. Si on est capable de définir des mesures de dissimilarité entre vidéos, on peut espérer plonger les données multi-dimensionnelles en deux dimensions de manière à ce que les positions relatives des vignettes résumées dépendent de la dissimilarité. Nous avons proposé une heuristique de réduction de dimension qui a été utilisée pour une exploration aisée de collections d'images indexées. Nous avons également mené des expériences pour une tâche de recherche d'image en utilisant une version multi-dimensionnelle de graphes de navigation basée sur nos descripteurs vidéos. Ce travail fait partie de mes activités les plus appliquées.

Chapitre 2

Codage compact de graphes et application au routage compact

Ce sujet porte sur un aspect algorithmique de la théorie des graphes : comment coder la structure d'un graphe avec un nombre optimal de bits ? Bien entendu, cette question n'a de sens que pour une famille de graphes donnée. S'il existe $f(n, m)$ graphes à n sommets et m arêtes possédant des propriétés topologiques particulières, un codage d'un graphe de cette famille nécessite au moins le logarithme en base 2 de $f(n, m)$ bits.

Les applications sont nombreuses : en synthèse d'images ou en visualisation d'information, la structure et la topologie des objets peuvent comporter des millions de points ou sommets (cf. Figure 2). Par exemple, un maillage d'objet 3D peut être représenté par un nuage de points (48 bits par point sont facilement nécessaires) et un graphe de relations entre les points (100 à 200 bits dans les années 1990). Pour économiser l'espace mémoire, le codage optimal ou compressé s'avère primordial. De manière générale, on peut considérer que les données brutes sont présentées sous forme de matrice d'adjacence de taille $\Theta(n^2)$ ou de listes d'adjacence de taille $\Theta(m \log n)$. Nous montrons dans ce chapitre que dans le cas des graphes planaires et des graphes de nombre de pages borné, nous pouvons faire bien mieux.

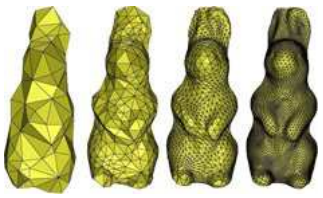


FIGURE 2.1 – Exemple de maillages d'objets 3D pouvant être représentés par un nuage de points et un graphe planaire.

D'un point de vue théorique, l'obtention d'un codage optimal est satisfaisant mais le praticien objectera que la compression et la décompression met tellement de temps qu'il peut devenir totalement inutilisable. Pour cette raison, nous rencontrons dans la

littérature deux familles de codage compact : des codages optimaux (ou quasi) et depuis peu, des codages *efficaces*. L'efficacité concerne deux paramètres :

- l'*espace mémoire* ;
- le *temps* requis pour effectuer des opérations élémentaires sans avoir à décompresser. Dans le cas des graphes, il est intéressant de tester l'adjacence en temps constant, de calculer le degré en temps linéairement proportionnel au nombre de voisins, ... Ce souci d'efficacité dans le domaine du codage compact est relativement récent (voir Munro et Raman [MR01]).

Le codage optimal est évidemment lié à des problèmes d'énumération. Il n'est absolument pas évident de calculer la cardinalité des familles de graphes. Parmi les challenges classiques de l'énumération et du codage compact, bon nombre d'études portent sur les graphes planaires. Un graphe est dit planaire s'il peut être représenté sur le plan sans intersection d'arêtes. Cette famille possède d'autres propriétés particulières (4-coloration des sommets ...) mais à ce jour, le nombre exact de graphes planaires à n sommets demeure inconnu.

Nos résultats portent sur

- deux familles de graphes : les graphes planaires (section 2.2) et les graphes planaires extérieurs - une sous-famille des graphes planaires - (section 2.6),
- Les plongements de graphes (arbitraires) en k pages et un schéma de routage compact associé (sections 2.4 et 2.5).

pour lesquels nous proposons de nouvelles bornes ou formules exactes d'énumération, des algorithmes de codage optimal ou efficace, des algorithmes de génération aléatoire. La maîtrise des plongements de graphes en k pages nous a permis de proposer un schéma de routage compact général pour les graphes de genre borné. Nous présentons dans les sections 2.2.4 et 2.3 les principaux outils originaux utilisés.

2.1 Quelques mots sur le codage compact et les requêtes locales

Soit $\mathcal{G}$ une famille de graphes et $\mathcal{A}$ un algorithme de codage. Le codage proposé par l'algorithme $\mathcal{A}$ pour représenter un graphe $G \in \mathcal{G}$ peut être vu comme un mot $\ell_{\mathcal{A}}(G)$ défini sur un alphabet Σ. Pour un mot w, la notation $|w|$ correspond au nombre de bits requis pour le codage de w. De manière générale, on s'intéresse à la quantité de mémoire pour stocker tous les graphes d'une famille donnée avec un algorithme fixé. La performance (en terme de mémoire) d'un algorithme de codage $\mathcal{A}$ appliqué à une famille $\mathcal{G}$ est donc :

$$|\ell_{\mathcal{A}}(\mathcal{G})| = \max_{G \in \mathcal{G}} |\ell_A(G)|. \tag{2.1}$$

Un (algorithme de) codage est dit *compact* si $|\ell_A(\mathcal{G})| = O(\log |\mathcal{G}|)$ où $|\mathcal{G}|$ désigne la cardinalité de la famille $\mathcal{G}$. En d'autres termes, le codage est optimal à une constante multiplicative près.

Partons d'un codage d'un graphe, nous pouvons nous intéresser à deux objectifs :

- compresser au maximum le codage ;

– permettre d'effectuer des requêtes basiques rapidement sur le codage compressé.

Par exemple, les codages classiques des graphes basés sur les matrices ou listes d'adjacence permettent de répondre en temps $O(\log n)$ à des requêtes d'adjacence mais ne sont pas particulièrement compacts. Une fois un codage compact donné, il n'est pas évident de localiser dans le mot les informations concernant un nœud donné, l'existence d'une arête, ... Rien ne garantit a-priori que les informations sur un nœud donné correspondent à des positions consécutives dans le codage.

Plus précisément, nous considérons comme requêtes :

– le *test d'adjacence* : pour deux nœuds fixés, indiquer s'ils sont reliés par une arête ;

– le *degré* : pour un nœud fixé, calculer le nombre de ses voisins ;

– la *liste des voisins* : pour un nœud fixé, lister les identifiants des nœuds voisins.

Le problème de la résolution rapide de requêtes se pose également dans le contexte de l'algorithmique distribuée. Imaginons que nous désirons répartir la description de la topologie d'un réseau dans les différents nœuds en limitant la redondance de l'information.

L'objectif est finalement le même que dans le codage compact. Une même technique, mais avec des contraintes différentes, peut éventuellement être utilisée dans les deux cadres : il s'agit de l'*étiquetage local*. Plus précisément, on cherche à associer à chaque sommet du graphe (ou nœud du réseau) un mot ou étiquette de plus petite taille de manière à ce que l'union des étiquettes locales permette de retrouver entièrement la description du graphe (voir le survol [GP03]). D'autre part, la plupart des requêtes basiques évoquées peuvent être résolues avec un nombre réduit d'étiquettes.

Plus formellement, nous cherchons, pour tout sommet u d'un graphe, à lui associer une étiquette $\ell_{\mathcal{A}}(G, u)$ qui soit la plus petite possible en taille et qui permette d'effectuer des requêtes rapidement. Pour simplifier la lecture, s'il n'y a pas d'ambigüité sur G et $\mathcal{A}$, nous utiliserons la notation $\ell(u)$.

Prenons par exemple le codage sous la forme d'une liste d'adjacence, l'étiquette d'un nœud correspond donc à la liste de ses voisins qui sont numérotés au moins entre 1 et n. Ceci implique que chaque étiquette a une taille d'au moins $\Omega(\mathtt{deg}(u) \log n)$ où $\mathtt{deg}(u)$ correspond au degré du sommet u. Nous montrerons dans la section 2.4 un algorithme de codage correspondant à un étiquetage local pour lequel le facteur logarithmique peut être enlevé. Il peut donc ainsi être utilisé dans le cadre d'un codage distribué de la topologie d'un réseau aussi bien que dans l'objectif du codage compressé.

2.2 Graphes planaires

2.2.1 Un bref tour d'horizon

Un graphe est *planaire* s'il peut être représenté sur le plan sans intersection d'arêtes. De nombreux problèmes concernant les graphes planaires ont été étudiés. Notamment, avec Cyril Gavoille et Nicolas Bonichon, nous nous sommes intéressés à des questions concernant l'énumération, la génération aléatoire et le codage.

Énumération. Compter le nombre de graphes planaires *non étiquetés* non isomorphes à n sommets est un vieux problème, extrêmement étudié, mais non résolu (cf. [LW87]). Aucune formule close n'est connue, ni même d'asymptotique. Il existe cependant les bornes qui découlent du nombre de graphes planaires *étiquetés* ou de cartes planaires : ainsi Osthus, Prömel et Taraz [OPT03] ont montré qu'il existe au plus $n!2^{5.22n+o(n)}$ graphes planaires étiquetés ; Noy et Gimenez [GN09] ont montré que le nombre de graphes planaires *étiquetés connexes* tend vers $n!2^{4.767n+O(\log n)}$. Liu [Liu88] a prouvé que le nombre de cartes planaires simples[1] tend vers $2^{5.098n+O(\log n)}$ ce qui implique directement une borne supérieure sur le nombre de graphes planaires simples. Pour résumer, il existe des résultats (bornes, asymptotiques) essentiellement pour des variantes de graphes planaires. Les techniques utilisées portent sur la manipulation de fonctions génératrices. Pour cela, la première difficulté est d'arriver à décomposer de manière non ambiguë les objets considérés. Malheureusement, à ce jour, on ne sait pas le faire pour les graphes planaires non étiquetés.

Génération aléatoire. La génération aléatoire d'objets combinatoires est une activité très utile pour l'analyse en moyenne d'algorithmes et pour multiplier les jeux de tests pour certaines expériences. Pour les graphes planaires, l'ajout d'une arête peut détruire la planarité. Du coup, les algorithmes de *génération aléatoire uniforme*[2] connus traitent essentiellement de *cartes planaires*, c'est-à-dire de plongements sur le plan. La nuance est importante car un même graphe peut avoir plusieurs plongements (par exemple un cycle auquel on ajoute deux arêtes incidentes au même sommet). Pour les cartes planaires, Denise *et al.* [DVW96] ont proposé un algorithme de génération dont l'uniformité n'est pas prouvé et dont le temps de génération est empirique. Si on veut se concentrer sur des algorithmes aléatoires uniformes en temps polynomial, Schaeffer [Sch99] puis Banderier *et al.*. [BFSS00] ont montré comment générer des sous-familles (planaires 3-connexes, ...). Il existe malgré tout des algorithmes de génération efficace pour des familles de graphe planaires : les arbres [ARS97], les graphes planaires extérieurs maximaux [BdLP99, ES94] (triangulation d'un polygone), ...

Codage compact. En 1984, Turán [Tur84] est le premier à proposer un codage compact dédié au graphe planaire en $4m$ bits, résultat amélioré par Keeler and Westbrook [KW95] en $3.58m$. Munro et Raman [MR97] descendent le codage à $2m + 8n$ bits. Il est à noter qu'ils utilisent pour cela le plongement en 4 pages des graphes planaires (voir [Yan89]). Dans une série d'articles, Lu et al. [CLL01, CGH$^+$98] ont raffiné le codage à $4m/3 + 5n$ bits. De manière indépendante, des codages en $4n$ bits pour les triangulations ont été obtenus par différents auteurs ([Bon02, CGH$^+$98, Ros99]) mais basés sur différentes techniques. Le codage de Rossignac [KR99], garantissant $3.67n$ bits pour les triangulations fortement connexes, est calculable en temps linéaire. D'après la formule de Tutte [Tut62],

1. un graphe est simple s'il ne contient ni boucle ni arêtes multiples
2. un algorithme de génération est *aléatoire uniforme* si tous les objets d'une même famille peuvent être généré de manière équiprobable.

on sait qu'il faut au moins $3.24n$ bits. En 2006, un codage optimal des triangulations a été proposé par Aleardi, Devillers et Schaeffer [ADS06].

D'un point de vue théorique, le meilleur codage général des graphes planaires est celui de He, Kao et Lu [HKL00, Lu02b] qui approche à un facteur $1 + o(1)$ un codage optimal en espace mémoire. La technique utilisée est basée sur une décomposition récursive des séparateurs du graphe. Cette technique, assez sophistiquée, ne donne cependant aucune borne explicite sur la taille du codage et ne semble pas être adaptée à des fins de génération aléatoire, de requêtes rapides.

Les codages les plus compacts sont basés soit sur l'approche *couverture par des arbres couvrants et codage de parcours* ou sur l'approche *décomposition arborescente*. Nos travaux utilisent la première approche.

2.2.2 Principal résultat

Voici le résultat principal de notre contribution à la connaissance des graphes planaires. Il nous a fallu 3 articles pour y arriver (cf. [BGH03b, PS03, BGH$^+$06]) et un article de synthèse est en soumission.

Théorème 2.2.1. *[BGH$^+$06] Tout graphe planaire connexe avec n nœuds et m arêtes peut être codé et décodé en temps linéaire avec au plus $4.91n + o(n)$ bits ou $2.82m + o(m)$ bits de mémoire.*

Ce résultat implique que le logarithme en base 2 du nombre de graphes planaires est plus petit que $4.91n$. Comme il est connu que le nombre de graphes planaires biconnexes a un logarithme binaire égal à $4.71n$, il s'en suit que notre codage est quasi-optimal. D'autre part, comme notre borne est paramétrable avec le nombre d'arêtes, en utilisant la borne inférieure issue de [GN09], nous sommes capables de montrer que *presque tous* les graphes planaires non étiquetés ont au moins $1.85n$ arêtes et au plus $2.44n$ arêtes. Ces deux bornes sont les meilleures connues à ce jour.

2.2.3 un résumé de la méthode

Notre technique est basée sur la décomposition en 3 arbres des triangulations du plan (appelée en anglais realizer) et des techniques sophistiquées de codage à l'aide de mots de parenthèses. La route pour mener au résultat est longue mais originale. Les sections suivantes décrivent brièvement les outils originaux que nous avons utilisés. La figure 2.2 résume les différentes étapes et techniques utilisées.

- À tout graphe planaire G, on associe un plongement cellulaire sur le plan, c'est-à-dire une carte planaire C ;
- On modifie alors la carte en préservant l'adjacence des sommets de manière à obtenir une carte C' ayant des propriétés particulières que je ne présente pas ici. Cette nouvelle carte est une carte *bien-ordonnée*.
- On construit de manière canonique un sur-graphe de la carte C' qui est une triangulation que nous avons appelée *super-triangulations*(cf. Figure 2.3). Cette super-triangulation a la particularité de pouvoir être décomposée en 3 arbres couvrants

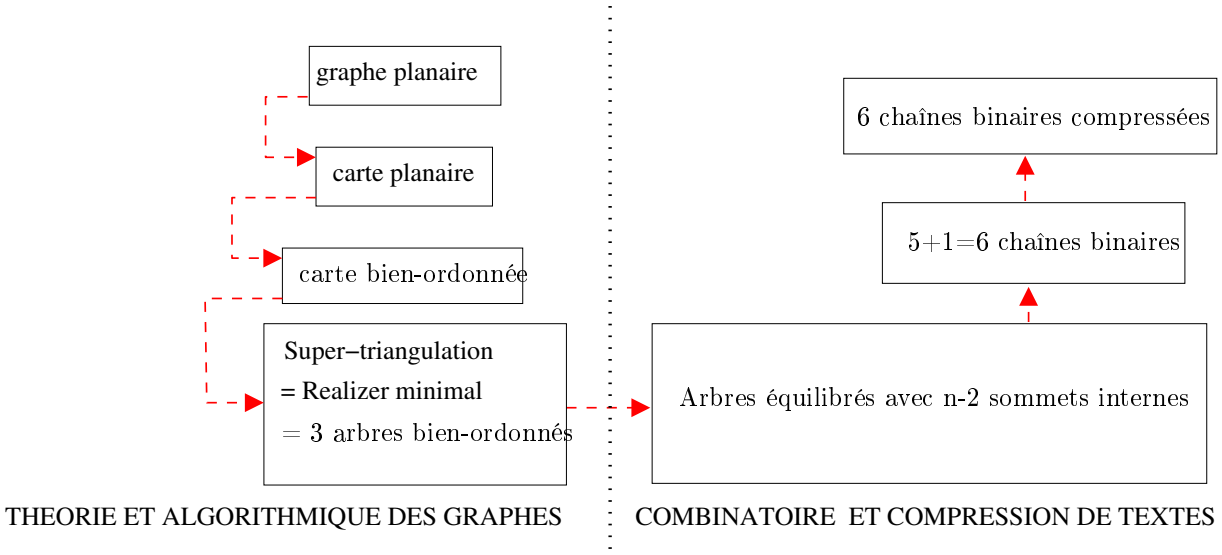


FIGURE 2.2 – Les différentes étapes menant au codage compact d'un graphe planaire.

qui s'entremèlent de manière très précise (cf. section 2.2.4). De plus, un des trois arbres est un sous-graphe du graphe initial. A ce stade, on peut obtenir un codage en $5.24n - 5$ bits. En effet, nous avons vu qu'une triangulation peut-être codée avec $3.24n$ bits et l'existence des arêtes ne se pose que pour les arêtes de deux des trois arbres, c'est-à-dire pour au maximum $2n - 5$ arêtes pour être précis.

- Poulhalon et Schaeffer ont montré une bijection entre super-triangulation et une famille d'arbres équilibrés dans [PS03]. Cette bijection a permis de combiner des propriétés des différents types d'arbres sachant que la connaissance de deux des trois arbres implique une description canonique du troisième.
- Pour finir, le codage est une union de six chaines binaires de différentes densités mais non indépendantes. L'analyse du codage compressé de l'ensemble de ces six chaînes nous donne les différents résultats.

2.2.4 Un peu de théorie des graphes : Arbres de Schnyder

Les arbres de Schnyder sont au cœur de notre approche. Il me paraît donc important de les mettre en lumière sans compter que nous les utilisons aussi dans un cadre assez différent : le résumé d'un espace euclidien de deux dimensions.

Définition 2.2.2 (realizer)**.** *Un* realizer *d'une triangulation est une partition des arêtes intérieures en 3 ensembles T_0, T_1, T_2 d'arêtes orientés tel que pour chaque sommet interne v, on a :*

- *l'ordre des aiguilles d'une montre des arêtes incidentes à v est : sortant dans T_0, entrant dans T_1, sortant dans T_2, entrant dans T_0, sortant dans T_1 et entrant dans T_2 ;*

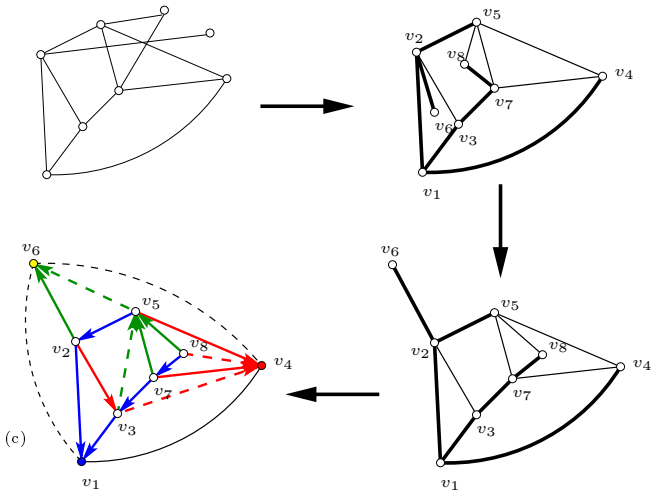


FIGURE 2.3 – D'un graphe planaire à une super-triangulation.

– *il existe exactement une arête sortante incidente à v dans T_0, T_1 et T_2.*

La figure 2.4 montre deux realizers de la même triangulation. Nicolas Bonichon a montré que le nombre de realizers est asymptotiquement égal à $2^{4n+O(\log n)}$ (cf. [Bon02]) alors que le nombre de triangulations est seulement $(256/27)^{n+O(\log n)}$ (cf. [Tut62]).

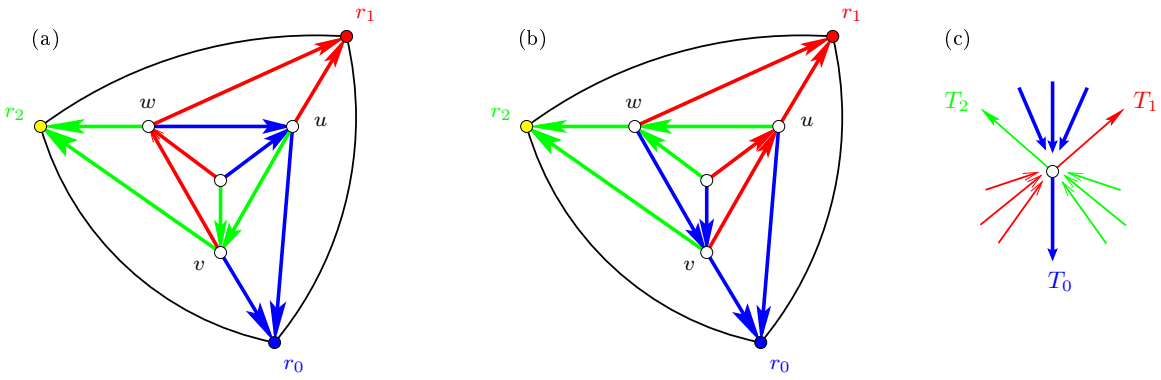


FIGURE 2.4 – (a)-(b) 2 realizers d'une même triangulation. (c) : ordre cyclique des arêtes orientées.

Schnyder a montré dans [Sch90] que chaque ensemble T_i d'un realizer est un arbre enraciné dans l'un des sommets de la face extérieure. Ainsi, les ensembles T_0, T_1 et T_2 sont nommés *arbres de Schnyder*.

Dans notre travail, les super-triangulations sont des realizers particuliers, *les realizers minimaux* : il n'existe pas de triangle orienté (pas nécessairement une face) formé par une arête de chaque arbre pris dans l'ordre cyclique T_2, T_1, T_0. Par exemple, le realizer (b) de la figure 2.4 est minimal mais ce n'est pas le cas pour le realizer (a).

2.3 Requêtes rapides dans les chaînes multi-parenthésées compressées

Nous utilisons comme outils dans diverses circonstances [BGH03a, GH08] des requêtes basiques sur des chaînes (ou mots) définies sur un alphabet Σ quelconque. Dans certains cas, nous manipulons des mots ayant certaines propriétés : les *chaînes multi-parenthésées équilibrées*.

Chaînes multi-parenthésées. Soit $\Sigma = \{(_1,)_1, (_2,)_2, \ldots, (_k,)_k\}$ un alphabet. Pour un entier $p \in [1, k]$, la sous-chaine composée de parenthèses de type $(_p,)_p$ est *équilibrée* si elle contient le même nombre de parenthèses ouvrantes et fermantes et si chaque préfixe contient une majorité (ou égalité) de parenthèses ouvrantes. En combinatoire, une chaîne de parenthèses équilibrées est appelée *mot de Dyck*. Sans perte de généralité, une chaîne de parenthèses multiple est équilibrée si pour tout p, chaque sous-chaîne est équilibrée.

Requêtes élémentaires sur une chaine. Dans nos travaux, pour obtenir rapidement des requêtes locales, nous utilisons les fonctions élémentaires suivantes appliquées sur une chaîne S (pas nécessairement de parenthèses) :

- *select*(i, σ) retourne la position de la i-ème occurrence du symbole $\sigma \in \Sigma$ dans S.
- *rank*(i, σ) retourne le nombre d'occurrences du symbole $\sigma \in \Sigma$ dans S avant (et incluant) la position i.
- Pour les chaînes parenthésées, si la i-ème position de S est une parenthèse ouvrante (resp. fermante) *match*(i) retourne la position dans S de la parenthèse fermante (resp. ouvrante) correspondante.

Par exemple, pour la chaîne S de la figure 2.6, *select*$(3, '[')$ retourne 16 et *match*(16) retourne 18. D'autres requêtes élémentaires existent mais ne sont pas présentées ici.

Les premières descriptions efficaces des chaînes équilibrées de k types de parenthèses pour plusieurs fonctions élémentaires ont été proposées dans [CGH+98, CLL01, MR01] lorsque $k = O(1)$. Pour une chaîne de longueur ℓ et $2k$ symboles, en ajoutant une table auxiliaire de $o(\ell \log k)$ bits, les opérations *select* and *rank* peuvent être exécutées en temps $O(\log \log k)$ [GMR06] ou $O(\log k / \log \log \ell)$ [FMMN07]. Ce temps devient constant dès que $k = O(\text{polylog}(\ell))$. Les plus récents développements permettent même de faire les mêmes requêtes si les chaînes sont compressées optimalement.

2.4 Graphes k-pages

Un *livre* est une surface composée de l'union de demi-plans appelés *pages* et dont l'intersection est une ligne, appelée *reliure*, correspondant à la frontière de chacune des pages. Cette terminologie est personnelle mais consiste en une traduction de la littérature anglaise. Un plongement en k-pages d'un graphe consiste en

- un ordre total sur les sommets (numérotés classiquement de 1 à n) représenté sur la reliure et

– une partition des arêtes en k parts (pages) telle que les arêtes d'une même page ne s'intersectent pas. [3]

Voir la Figure 2.5 pour un exemple. Un tel plongement est appelé *plongement en pages* (cf. [Bil92, DW04] pour un survol). Les graphes ayant un plongement en k pages sont appelés des graphes k-pages. Le *nombre de pages* d'un graphe G est le plus petit entier k tel que G soit un graphe k-pages.

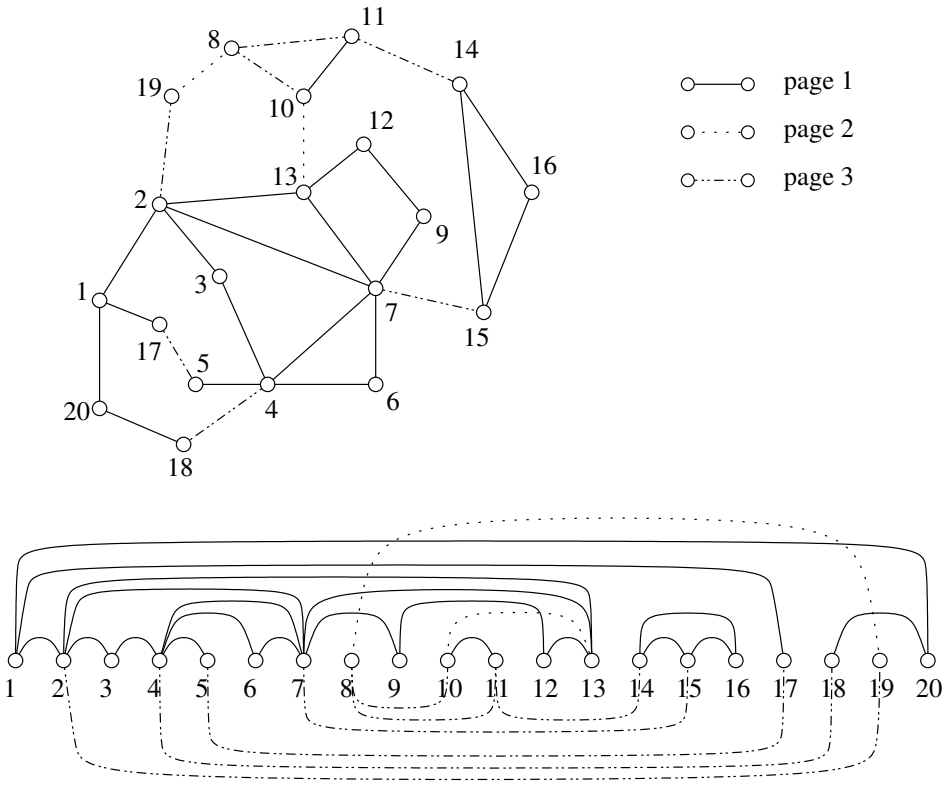


FIGURE 2.5 – Un graphe planaire et son plongement en 3 pages.

Nombre de pages d'un graphe. Pour tout graphe, il existe un nombre minimal de pages. Il est connu que tout graphe planaire peut être plongé en au plus 4 pages [Yan89]. Le nombre minimal de pages pour l'ensemble des graphes planaires est donc 4. Ce nombre peut être linéaire (pour les graphes complets). Les graphes de nombre de pages 1 sont les graphes planaires extérieurs (incluant les arbres) et les graphes de nombre de pages 2 sont les sous-graphes des graphes planaires hamiltoniens (incluant les séries-parallèles). Il existe aussi des graphes non planaires ayant un nombre de pages inférieur à 4 (K_5 est un 3-pages). De manière générale, le calcul du nombre minimal de pages d'un graphe est un problème NP-dur et aucun algorithme d'approximation n'est connu à ce jour.

3. en supposant que $u < v$, $u' < v'$ et $u < u'$, l'arête (u, v) et l'arête (u', v') s'intersectent si et seulement si $v < v'$.

Notre résultat. Nous avons prouvé que le codage des graphes k pages à n sommets et m arêtes nécessite $kn - o(kn)$ bits à partir d'une borne inférieure sur le nombre de graphes de nombre de pages égal à k :

Théorème 2.4.1. *[GH08] Le nombre $\mathcal{U}_{n,k}$ de graphes connexes simples de n sommets et de nombre de pages k, $1 \leq k \leq n/2$, satisfait*

$$\mathcal{U}_{n,k} \;\geq\; \frac{\left(2^k\left(2^k-1\right)\right)^{n/2-k}}{2\,k!}\;.$$

Ce résultat implique que le codage de Munro et Raman de $4kn+2n-k^2+o(n)$ bits est optimal à un facteur 4 près. Nous avons proposé un nouveau codage en $2m\log(k)+4m$ bits qui s'avère plus efficace pour de nombreux graphes k pages (dès que le graphe est peu dense). De plus, notre codage en deux chaines (une multi-parenthésée S avec $2k$ symboles et l'autre B binaire. cf Figure 2.6) est efficace car il supporte des requêtes d'adjacence rapide. En résumé, le principe d'encodage consiste à définir un ordre total entre les extrémités d'arêtes : à chaque sommet pris dans l'ordre total des sommets, on code d'abord les arêtes déjà rencontrées puis les nouvelles. La chaîne binaire sert à localiser les sommets.

Plus formellement, notre encodage implique que :

Théorème 2.4.2. *[GH08] Tout graphe k-pages avec m arêtes (autorisant arêtes multiples et boucles) et i sommets isolés peut être représenté avec $2(m+i)\log_2 k + 4(m+i)$ bits. De plus, avec une table auxiliaire de taille $o((m+i)\log k)$ bits, il est possible de tester l'adjacence en temps $O(\tau \cdot \log k)$, avec $\tau = \min\{\log k/\log\log m, \log\log k\}$). Le calcul du degré peut être fait en temps $O(1)$ et la liste des δ voisins d'un sommet prend un temps $O(\tau\delta)$.*

Le terme multiplicatif τ provient du temps mis pour effectuer une itération des fonctions *rank*, *select*, ... appliquées sur nos deux chaînes. La partie difficile consiste à montrer que les requêtes locales avec peu d'appels (typiquement l'adjacence en temps $O(\log k)$) des fonctions peuvent être realisées. Pour l'adjacence, cela s'explique par le fait absolument remarquable qu'il suffit de tester l'adjacence dans au plus $O(\log k)$ pages.

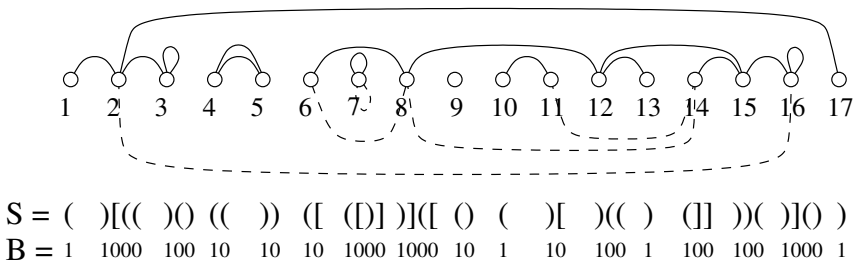


FIGURE 2.6 – Un 2-pages et son codage.

2.4.1 Perspectives concernant le codage compact

On peut notamment penser aux travaux suivants :

- *Codage compact et efficace de graphes planaires.* Immédiatement, nous obtenons un encodage qui peut s'avérer extrêmement performant pour les graphes planaires. En effet, il existe un algorithme - certes complexe - linéaire qui permet de plonger un graphe planaire en un k-pages avec $k \leq 4$. Ainsi, non seulement le codage tient un espace $\Theta(n)$ bits qui est optimal à une constante multiplicative prés (cf. Section 2.2) mais les requêtes locales d'adjacence et de degré s'effectuent en temps $O(1)$. Notre codage est donc théoriquement bien plus rentable que la liste d'adjacence. Une expérimentation serait intéressante.
- *Stockage distribué de k-pages.* Une première application directe est le stockage distribué d'un graphe k-pages. Pour cela, on peut associer facilement à chaque sommet un étiquetage local. Il est composé bien entendu des sous-chaînes brièvement décrites (grosso-modo prenant un espace de l'ordre de $\Theta(\delta \log k)$ bits) mais aussi de tables auxiliaires pour exécuter les fonctions *rank*, *select*, ... Celles-ci prennent un espace $o((m+i)\log k)$. Mais est-ce suffisant ?
- *Obtention d'un plongement en pages.* Pour rendre notre codage pratique en général, il faut partir d'un plongement avec le plus petit nombre de pages. Il n'existe pas à notre connaissance d'algorithme de plongement qui approximerait le nombre de pages d'un graphe. Ce problème est à étudier.

2.5 Application au routage compact pour les graphes de genre borné

2.5.1 Routage, étirement et latence

Modèle de routage. Le *routage* consiste à acheminer un message entre deux nœuds identifiés du réseau. Le réseau est modélisé par un graphe dont les sommets (noeuds) et les extrémités d'arêtes (ports de sortie) sont étiquetés. Classiquement, nous supposons que chaque nœud u possède un identifiant $id(u)$ (souvent un numéro entre 1 et n mais pas nécessairement) et stocke une structure de données. De même, les ports d'entrée (resp. de sortie) sont numérotés entre 1 et $\texttt{deg}^-(u)$ (resp $\texttt{deg}^+(u)$). Selon les circonstances, on peut considérer que le graphe sous-jacent est simple (ou non), pondéré, orienté, ...

Un message est constitué d'un *en-tête* et d'un *corps de message*, c'est-à-dire le message lui-même. L'en-tête comprend l'identifiant du nœud de départ ainsi que l'identifiant du nœud d'arrivée. A l'arrivée à un nœud, un algorithme distribué, appelé *schéma de routage*, est exécuté localement en prenant en compte l'en-tête, le port d'entrée et la structure de donnée locale. Il retourne le port de sortie. Si l'algorithme distribué n'est pas autorisé (resp. est autorisé) à modifier l'en-tête, le schéma de routage est dit *sans mémoire* (avec mémoire).

Performances liées au routage. L'*étirement* est une mesure de qualité de la route prise lors d'un routage. Soit $\mathcal{A}$ un schéma de routage. Soit P le chemin dans G pris par le message lors d'un routage entre u et v. L'*étirement* de P, noté $stretch(u, v)$, est égal au ratio entre la longueur de P et la distance entre u et v dans G. Plus généralement, l'étirement de $\mathcal{A}$ sur G vaut $\max_{u \neq v} stretch(u, v)$. Un routage de plus courts chemins a un étirement de 1.

La *latence* est le temps de calcul local de l'algorithme distribué. En général, nous considérons le pire cas.

L'*espace mémoire* considéré est la taille de la table de routage exprimé en bits. Sans hypothèse particulière, l'espace mémoire pour garantir un routage de plus court chemin est $\Omega(n \log n)$ bits (cf. [GP96]). Notre contribution est un *schéma de routage universel* dont l'espace mémoire dépend d'un paramètre de graphe : le genre ou le nombre de pages.

2.5.2 Notre contribution

Nous nous sommes intéressés dans [GH99] aux schémas de routage par plus courts chemins, sans mémoire, pour les graphes de genre γ et les graphes de nombre de pages k. Le *genre* γ d'un graphe G est le plus petit entier γ tel que G peut être plongé sans intersection d'arêtes sur une surface de genre γ, c'est-à-dire une sphère avec γ trous. En réalité, il existe une relation forte entre genre et nombre de pages, Malitz [Mal88] a montré que $k = O(\sqrt{\gamma})$ sachant que $k = \Omega(\sqrt{\gamma})$ [HI87].

Nous avons montré que pour n'importe quel routage de plus court chemin choisi, on peut créer un schéma de routage *avec indépendance des noms.*

- avec des tables de mémoire $n \log k + O(n)$ (resp. $n \log \gamma + O(n)$) bits de mémoire par noeuds pour les graphes de nombre de pages k (resp. graphes de genre γ).
- un temps de latence de $O(\log^{2+\epsilon} n)$;

Ce résultat très général ne peut être amélioré dans le sens ou nous avons aussi montré une borne inférieure de $\Theta(n \log k)$ bits d'espace mémoire pour au moins un nœud pour une grande famille de graphes k-pages dans le même article.

Pour les graphes planaires, l'espace mémoire devient $8n$ bits. A ce moment-là, il s'agissait du schéma de routage le plus compact existant pour les graphes planaires. En 2002, Lu [Lu02a] a proposé un schéma qui descend l'espace mémoire à $7.18n$ bits.

2.5.3 Techniques diverses

Cet article est intéressant à plusieurs égards. Nous utilisons différentes techniques. Nous partons cependant d'une hypothèse forte : le graphe considéré en entrée est déjà plongé en pages.

Précalcul des chemins. Le principe du schéma de routage consiste à ce que chaque nœud stocke non pas la description complète du graphe mais la description du plongement sur k pages d'un arbre couvrant de poids minimal - heureusement de manière compacte - puis calcule localement la route pour aller à la destination. Pour cela, nous utilisons une nouvelle fois les *fonctions élémentaires sur les chaînes.* Celles décrites dans [MR97]

sont suffisantes. Le problème est que le nombre de "sauts" pour aller d'un sommet à un autre pourrait être linéaire entraînant un temps de latence linéaire car on fera au moins un nombre d'appels linéaire aux fonctions élémentaires.

Domination. L'astuce pour réduire le temps de latence à un nœud u consiste à construire pour un "petit" échantillon de sommets S_u le port de sortie de u qui mène à chaque nœud de S_u. Sans décrire l'usage de S_u, on peut quand même indiquer que S_u est un ensemble r-dominant de l'arbre de plus courts chemins (avec $r = \log^{1+\epsilon}$ dans notre cas) et que le stockage supplémentaire à chaque nœud est négligeable par rapport aux chaînes stockées initialement. Un *ensemble r-dominant* S est un sous-ensemble des sommets d'un graphe tel que tout sommet du graphe est à distance au plus r d'un élément de S. Cet outil nous a été utile dans diverses circonstances.

Complexité de Kolmogorov. Pour obtenir notre borne inférieure, nous avons utilisé des techniques de bornes inférieures basées sur la quantité de mémoire requise pour décrire toutes les combinaisons d'arbres de plus courts chemins possibles.

2.5.4 Évolution du routage compact

De très nombreux travaux ont été effectués depuis notre travail. De manière générale, la communauté s'est concentrée sur des schémas de routage avec un étirement constant et pour de grandes familles de graphes (avec mineur exclu, à croissance bornée ou même arbitraire). Des questions générales subsistent : les bornes changent-elles si on interdit de re-étiqueter les nœuds, si le graphe en entrée est orienté ?

En ce qui concerne les graphes planaires, la borne inférieure pour le routage de plus court chemin est $\Omega(n^{1/3})$ avec des identifiants de taille polylogarithmique. Le majorant $7.18n$ de Lu n'a pas été amélioré. La question reste ouverte. Il est à noter qu'il y a une très grande différence si l'étirement est différent de 1 car Thorup a montré qu'avec un étirement égal à $1 + \epsilon$, l'espace mémoire devient polylogarithmique pour chaque nœud.

2.6 Cartes planaires extérieures

Les graphes planaires extérieurs sont des graphes planaires dont tous les sommets peuvent être dessinés sur la face extérieure (voir l'exemple de la figure 2.7) . Les faces sont les régions du plan délimitées par les arêtes lors d'un dessin sur le plan du graphe planaire. D'un point de vue topologique, nous parlons de cartes plutôt que de graphes pour désigner les dessins ou plongements de graphes sur des surfaces. On peut considérer les graphes ou cartes planaires extérieures comme une sur-famille des arbres. Les techniques élaborées pour le codage de graphes planaires s'appliquent facilement pour cette famille : réalizer minimaux, bijection et codage en chaines compressées. Cependant, nous avons dû montrer des propriétés supplémentaires spécifiques aux cartes planaires extérieures : un des trois arbres de Schnyder est vide, ...

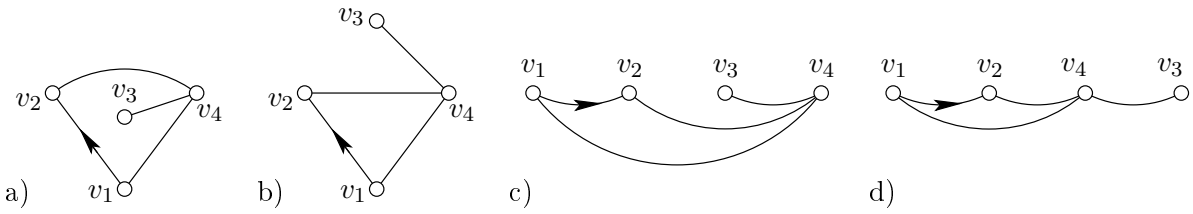


FIGURE 2.7 – Un même graphe mais 4 plongements différents : non-extérieur, extérieur, 2 plongements en pages différents.

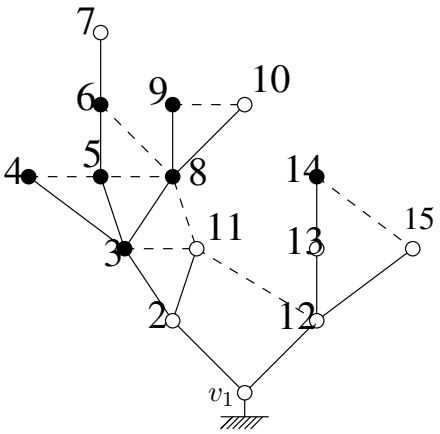


FIGURE 2.8 – Exemple de décomposition d'une carte planaire extérieure en un arbre de Schnyder et au plus une arête supplémentaire par sommet.

Notre contribution. Nous avons défini une décomposition de cartes planaires extérieures enracinées $\mathcal{PE}$ en un arbre et une liste d'arêtes. Cette décomposition, constructible en temps linéaire, implique l'existence d'une bijection entre les cartes planaires extérieures enracinées à n sommets $\mathcal{PE}_n$ et les arbres bicoloriés enracinés à n sommets ayant comme dernière branche des sommets unicoloriés (cf. Figure 2.8). Comme conséquence, nous avons montré dans [BGH03a] :

- une *formule d'énumération nouvelle* : $|\mathcal{PE}_n| = \sum_{d=1}^{n-1} 2^{n-d-1} \frac{d}{2n-2-d}\binom{2n-2-d}{n-1-d} \sim \frac{2^{3n}}{36n\sqrt{\pi n}}$ sommets ;
- un *codage optimal et efficace* en $3n$ bits qui admet sans décompression des requêtes d'adjacence et de degré en temps constant et donne la liste des d voisins en temps $O(d)$;
- un algorithme de *génération aléatoire uniforme* qui fonctionne en temps linéaire.

Concernant l'algorithme de génération, il s'agit d'une méthode à rejet basée sur notre bijection et sur l'algorithme de génération aléatoire uniforme de Arnold et Sleep [AS80]. Des résultats similaires sont également donnés pour les cartes planaires extérieures avec n sommets et m arêtes. Par rapport à notre étude des graphes planaires arbitraires, nous avons utilisé un nouvel outil pour obtenir l'obtention d'un codage efficace : les *requêtes*

rapides dans les chaînes parenthésées présenté dans la section 2.3.

Chapitre 3

Navigabilité des graphes

Dans la section 2.5, l'obtention de routage passe tout d'abord par un pré-calcul qui suppose une *connaissance totale* et donc *exacte* du graphe. Dans le cas d'un réseau de grande taille, cette opération peut s'avérer très coûteuse, voire même impraticable dès lors que le réseau est *dynamique*. Que peut-on faire si on ne possède qu'une *connaissance partielle* ? Il paraît illusoire de chercher à router par les plus courts chemins. De manière informelle, s'il est possible de router sans faire une exploration totale du graphe, nous dirons que le graphe ou réseau est *navigable*. D'autre part, si des chemins courts existent, peut-on les trouver ?

Oracle avec connaissance partielle. Ainsi, ce chapitre traite principalement de modèles de *routage sans mémoire avec une connaissance partielle*. Nous ne faisons aucune hypothèse à priori sur la manière dont cette connaissance partielle s'est constituée. Elle peut être naturelle : les identifiants peuvent correspondre à des coordonnées dans un espace euclidien, la date d'apparition des nœuds pour un réseau dynamique, ... mais aussi plus complexe. De manière générale, nous supposons que chaque nœud du réseau dispose d'un *oracle* [1], c'est-à-dire d'une boîte noire qui répond instantanément à une question posée. Le paramètre de *latence* n'est donc plus considéré. Dans notre cadre, l'oracle prend en entrée l'identifiant du nœud courant, $id(u)$ et celui du nœud cible $id(t)$ et retourne l'identifiant $id(v)$ d'un voisin de u pour mener à la cible. Tel quel, rien ne dit que ce voisin va rapprocher de la cible car l'oracle ne dispose que d'une connaissance partielle.

D'autre part, les résultats de ce chapitre traitent majoritairement de modèles de graphes aléatoires contrairement au chapitre 2. Nous démarrons par un sujet qui a fait couler beaucoup d'encre ces dernières années : l'effet petit-monde.

Modèles de connaissance. La connaissance de l'oracle est directement liée à la connaissance partielle du graphe dans lequel la navigation est effectuée. Le premier modèle présenté est celui de Kleinberg [Kle00]. Il s'agit d'un modèle de *graphe augmenté*. On suppose connue la structure d'un graphe H. Les distances, plus courts chemins, etc ...

1. Étymologiquement, le terme le plus approprié serait *Pythie* car l'oracle n'est que la réponse donnée par une pythie sensée être porte-parole de Dieu, c'est-à-dire de la connaissance.

dans H font partie de la connaissance de l'oracle. Le graphe final G est obtenu en ajoutant un ensemble d'arêtes L de manière aléatoire (mais pas nécessairement uniforme). Cet ensemble d'arêtes supplémentaires est à priori inconnu. Il se peut donc que les plus courts chemins changent complètement et que le diamètre du graphe soit réduit.

Nous avons étudié ensuite des modèles de *graphes aléatoires sans échelle* : la distribution des degrés suit une loi de puissance, c'est-à-dire que le nombre de sommets de degré x est x^{-c} avec $c > 1$ étant un paramètre de la loi. Parmi eux, les modèles de *graphes évolutifs* sont des modèles originaux de connaissance : tout nouveau sommet est horodaté et relié à un nombre constant de sommets (choisis aléatoirement et pas nécessairement uniformément) appartenant déjà au graphe. La section 3.1 traite des graphes navigables pour lesquels des mécanismes de routage glouton fonctionnent efficacement. Dans la section 3.3, nous montrons néanmoins qu'avec un oracle ne disposant que d'une connaissance locale (identifiants des voisins) et du mécanisme de construction des graphes, il n'est pas possible de router de manière performante.

3.1 Graphes navigables

3.1.1 Algorithmes gloutons de navigation

Nous allons décrire des variantes d'algorithmes de routage qui vont dépendre de la connaissance de l'oracle ainsi que des contraintes sur la mémoire autorisée. Nous avons étudié ces algorithmes dans différents contextes : routage et recherche d'informations.

Notations. Nous supposons que la recherche est effectuée par un agent mobile ayant une mémoire M. Si X est un ensemble de sommets, on note $\Gamma(X)$ l'ensemble des voisins de X dans le graphe G. La notation $X \backslash M$ représente l'ensemble $\{u \in X | u \notin M\}$. La fonction $\delta(u, v)$ mesure une distance entre les identifiants des nœuds u et v. La définition de la fonction de distance est dépendante du modèle considéré et ne doit pas être confondue par la distance $d(u, v)$ entre u et v dans le graphe qui est inconnue par définition.

L'algorithme de recherche `generique` *avec mémoire non bornée* (cf. Listing 3.1) garantit un succès dans la recherche, y compris dans un graphe non-navigable : le prochain nœud à atteindre est un voisin du sommet visité dont la distance à la cible est la plus petite. Deux algorithmes *sans mémoire*, `glouton` et `glouton+` garantissent le succès si le graphe est navigable. La nuance est que `glouton+` fait le meilleur choix local : il indique, parmi les voisins, le nœud qui rapproche le plus. Une remarque générale s'impose : il se pourrait qu'il n'existe pas de nœud dont l'identifiant est t. La recherche s'arrête alors. Dans ce chapitre, nous ne considérons que le cas pour lequel l'identifiant de la cible est connu.

3.1.2 Graphes navigables

Voici une définition formelle et personnelle de la navigabilité :

source s, cible t
M $\leftarrow s$
$u \leftarrow s$
$v_{min} \leftarrow v \mid \forall v, v' \in \Gamma(M) \backslash M,\ \delta(v,t) < \delta(v',t)$
TantQue $u \neq t$ **Et** $v_{min} \neq \emptyset$ **Faire**
    $u \leftarrow v_{min}$
    Ajouter(u,M)
    $v_{min} \leftarrow v \mid \forall v, v' \in \Gamma(M) \backslash M,\ \delta(v,t) < \delta(v',t)$
FinTantQue

Listing 3.1 – `generique`

source s, cible t
$u \leftarrow s$
$v_{min} \leftarrow$ un élément $v \in \Gamma(u)$ tel que $\delta(v,t) < \delta(u,t)$
TantQue $u \neq t$ **Et** $v_{min} \neq \emptyset$ **Faire**
    $u \leftarrow v_{min}$
    $v_{min} \leftarrow$ un élément $v \in \Gamma(u)$ tel que $\delta(v,t) < \delta(u,t)$
FinTantQue

Listing 3.2 – `glouton`

Définition 3.1.1 (Graphe navigable). *Un graphe $G = (V, E, id)$ est f-navigable si à partir d'un étiquetage des sommets par la fonction id, l'algorithme* `glouton` *mène à la cible en au plus f sauts.*

D'autre part, il est important de donner une définition de petit-monde car la littérature regorge de versions et d'interprétations :

Définition 3.1.2 (Petit-monde). *Un graphe $G = (V, E, id)$ est un petit-monde s'il est $O(\log^{O(1)} n)$-navigable.*

source s, cible t
$u \leftarrow s$
$v_{min} \leftarrow v \mid \forall v, v' \in \Gamma(u),\ \delta(v,t) < \delta(v',t)$
TantQue $u \neq t$ **Et** $v_{min} \neq \emptyset$ **Faire**
    $u \leftarrow v_{min}$
    $v_{min} \leftarrow v \mid v, v' \in \Gamma(u),\ \delta(v,t) < \delta(v',t)$
FinTantQue

Listing 3.3 – `glouton+`

Informellement, dans un petit-monde, on atteint la cible en ne traversant qu'un sous-ensemble petit du graphe. Suivant l'usage, on peut toujours considérer qu'un graphe $o(n)$-navigable est un petit-monde mais nos résultats ne sont pas basés sur cette définition. Avec Philippe Duchon, Emmanuelle Lebhar et Nicolas Schabanel, nous avons étudié le processus d'augmentation qui consiste à transformer un graphe navigable en petit-monde que nous avons nommé **la petit-mondisation**. Notons $H = (V, E, id)$ le graphe navigable. Soit L un ensemble d'arêtes non contenu dans E. $G = (V, E \cup L, id)$ est le sur-graphe de H *augmenté* par L. Pour tout sommet $u \in V$, l'ensemble $L_u = \{v \in V | (u, v) \in L\}$ désigne les *contacts distants* de u. L est constitué des *liens longs* de G. Dans notre modèle de connaissance, nous supposons que *seule la distance $d(u, v)$ est connue dans H mais pas dans le graphe augmenté G.*

3.1.3 Une motivation : réseau logique pour réseaux pair-à-pair

Dans un réseau pair-à-pair, tous les noeuds du réseau jouent le même rôle. Aucune infrastructure centralisée n'est supposée. Chaque noeud du réseau possède un *carnet d'adresses* de noeuds. Par nature, les réseaux pair-à-pair sont extrêmement dynamiques. Le carnet d'adresses ne cesse d'évoluer. La difficulté première est d'assurer la connexité du réseau. La solution consiste à maintenir un réseau logique entre les noeuds. Un *réseau logique* est un *réseau virtuel* dont les noeuds correspondent aux processeurs et/ou aux ressources (données, blocs mémoire, services, ...) du réseau. La conception des liens virtuels dépend des tâches à accomplir : routage, diffusion, ... De manière générale, les réseaux logiques doivent supporter l'insertion/suppression d'utilisateurs/ressources. De nombreuses solutions existantes garantissent la connexité.

Pour un objectif de recherche (localisation de ressources, accés, ...), plusieurs techniques existent basées sur :

- *des réseaux structurés :* Le graphe sous-jacent est déterministe, possède souvent des propriétés précises (typiquement petit diamètre, degré borné, ...) et est *navigable*. Les requêtes sont traitées de manière efficace : routage et diffusion sont effectués rapidement avec peu de messages. L'inconvénient majeur est que la maintenance dynamique de ces structures précises est très délicate et ne donne aucune garantie lors de perturbation car il devient non navigable.
- *des réseaux non-structurés :* Ici, il n'y a pas contrôle strict de la topologie. Pour la recherche d'information, le seul moyen de limiter le nombre de messages consiste à optimiser l'inondation en tenant compte des statistiques stockées à chaque noeud : nombre de requêtes menant à un item en traversant un lien donné, ...

3.1.4 Graphes aléatoires navigables

Je m'intéresse à un autre type d'approche. Si on relâche la condition de déterminisme d'un réseau logique structuré mais en préservant des propriétés de navigabilité, nous pouvons obtenir des réseaux logiques très performants. C'est l'espoir suscité par les graphes *petit-mondes*(cf. Section 3.2) et de manière plus générale les *graphes aléatoires*.

L'intérêt des graphes aléatoires pour la diffusion (cf. Section 5.1) n'est pas si récente : par exemple, en supposant que chaque noeud a la possibilité de choisir un noeud de manière aléatoire dans le réseau, la mise-à-jour de bases de données répliquées peut être faite avec peu de messages échangés et une faible complexité en temps [DGH+88, KSSV00].

Pour différentes requêtes de recherche, il a été montré depuis peu que le *temps de recherche*, exprimé en nombre de sauts (nombre de pairs visités), devient faible [Kle00, FGP04, DHLS05].

3.1.5 Enjeu

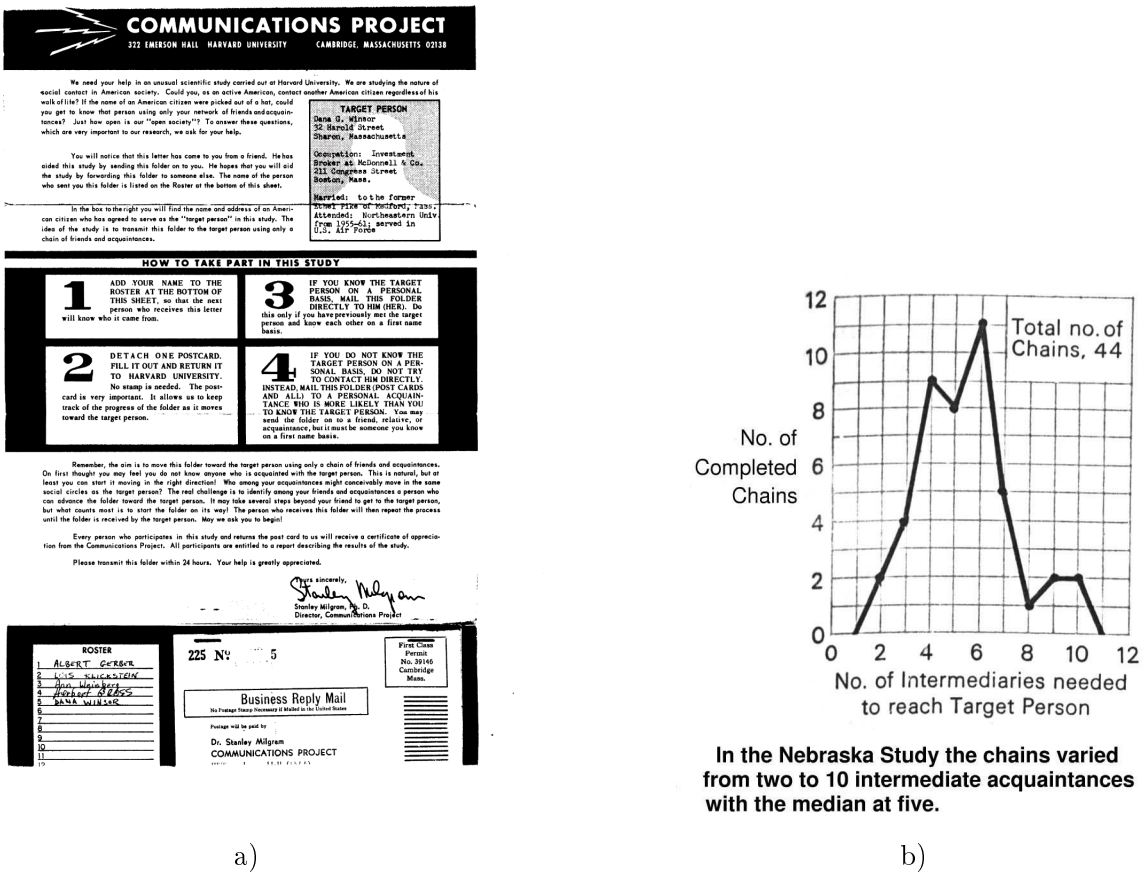
Notre objectif est bien entendu l'étude des performances algorithmiques de ces graphes (temps, mémoire, pertinence, tolérance aux fautes, auto-stabilisation, ...) et de proposer de nouveaux graphes navigables. Le véritable challenge est de se passer de la connaissance totale pour effectuer le pré-calcul de l'oracle. Ce problème correspond à la maintenance économique du réseau logique dans un environnement distribué et dynamique. Notre contribution dans ce domaine est présentée dans [DHLS06]. Nous y proposons un algorithme distribué économique pour transformer un graphe de croissance borné (représentant la connaissance partielle) en petit-monde.

3.2 Petit-monde

Bon nombre de réseaux réels aussi variés que les réseaux internet, réseaux sociaux, réseaux d'interaction protéines-protéines partagent des propriétés structurelles remarquables : petit diamêtre, forte densité locale, densité globale faible, ... De tels réseaux sont communément appelés « petit-mondes » bien qu'il n'existe pas de définition formelle à ce jour. De manière plus surprenante, de nombreux plus courts chemins peuvent être trouvés entre n'importe quelle paire de nœuds (cf. Milgram [Mil67] en 1967) de manière distribuée sans qu'aucun nœud ait une connaissance globale du réseau.

3.2.1 L'expérience de Milgram

L'adage selon lequel le monde est petit est ancien. Dès 1929, l'écrivain Frigyes Karinthy publia un recueil d'histoires en énonçant la propriété de six degrés de séparation : "en au plus 5 intermédiaires, il est possible de contacter toute autre personne dans le monde en supposant que chaque individu n'utilise que ses véritables connaissances pour faire transiter un message". En 1967, Milgram proposa une série d'expériences pour observer et confirmer cette affirmation. En sélectionnant une personne cible basée dans la banlieue de Boston, il demanda à des groupes d'individus choisis aléatoirement dans le Nebraska et à Boston de faire suivre un message en direction de la personne cible - qu'ils ne connaissaient pas - en utilisant uniquement leur connaissance personnelle. Seuls des informations sur le nom, l'adresse, la profession du destinataire étaient indiquées (cf. Figure 3.1).



a) b)

FIGURE 3.1 – a) Message original diffusé dans l'expérience de Milgram. b) Distribution du nombre d'intermédiaires lors des succès.

Sur 217 individus de départ, 64 chaînes sont arrivées à la personne cible avec en moyenne $5,2$ intermédiaires. Ainsi, Milgram put affirmer que le monde semblait réellement petit. Mais le plus surprenant est de voir qu'un mécanisme de routage simple *sans connaissance globale* permet d'acheminer des messages par des courts chemins.

3.2.2 Un premier modèle de petit-monde navigable

Différents modèles ont été proposés pour tenter d'expliquer ce phénomène (cf Watts et Strogatz [WS98]) mais Kleinberg [Kle00] a montré en 2000 que ces modèles ne possédaient pas la propriété essentielle de navigabilité : bien que de tels réseaux aient un diamètre polylogarithmique en la taille du réseau, aucun court chemin ne peut être trouvé sans une connaissance totale du réseau. Plus précisément, avec une vision partielle du réseau, le routage mène à des chemins de longueur polynomiale en la taille du réseau. Kleinberg proposa un modèle de graphe augmenté constitué d'une grille dans laquelle on rajoute à chaque nœud un nombre constant de longs liens distribués selon la distribution

harmonique, c'est-à-dire qu'un nœud v est un contact éloigné du nœud u avec probabilité proportionnelle à $\frac{1}{d(u,v)^s}$. Dans ce modèle, la connaissance locale de chaque nœud est la métrique de la grille ainsi que la position des contacts éloignés. Kleinberg a montré qu'il existe un algorithme décentralisé (n'utilisant que la connaissance locale) qui calcule pour toute paire de nœuds un chemin de longueur polylogarithmique si et seulement si s est égal à la dimension de la grille. L'algorithme est extrêmement simple : chaque nœud transmet le message à son voisin qui est le plus proche (dans la métrique de la grille) à la destination. Il s'agit de l'algorithme `glouton+` . Barriere et al ont ensuite montré que la longueur de routage de ce simple algorithme est au moins proportionnelle à $\log^2 n$ où n désigne le côté de la grille.

a) b)

FIGURE 3.2 – a) Grille augmentée d'un contact distant b) Grille augmentée de plusieurs contacts distants

Il est important de noter qu'il ne suffit pas que la grille augmentée ait un diamètre polylogarithmique (c'est le cas si $s < 2D$ où D est la dimension de la grille) pour que le routage trouve un chemin de longueur polylogarithmique.

3.2.3 Petit-mondisation de graphes à croissance bornée

On pourrait être tenté de croire que la propriété de petit-monde navigable est très spécifique au modèle de grille augmentée. Ce résultat se généralise. De manière générale, nous appelons le processus de tranformation d'un graphe navigable en petit-monde **la petit-mondisation**. Avec Philippe Duchon, Emmanuelle Lebhar et Nicolas Schabanel, nous avons montré dans [DHLS05] qu'en réalité de grandes familles de graphes peuvent être utilisées comme graphe de base (au lieu de la grille) et augmentées d'un lien long par nœud, de tels graphes devenant ainsi des petit-mondes navigables. Notre graphe d'entrée est de *croissance* $\alpha(r)$*-bornée* avec $\alpha(r) = O(\log^{O(1)} r)$. Pour tout sommet u, si $b_u(r)$ est le nombre de sommets à distance au plus r de u, alors $b_u(2r) \leq \alpha(r) b_u(r)$.

Le principe consiste à utiliser une distribution pour les liens longs qui prenne en compte l'expansion des tailles de boules (nombre de noeuds à distance donnée) centrées en chaque noeud. Plus formellement, le processus est décrit dans l'algorithme 3.2.3 :

Notre résultat implique que tout graphe ayant des croissances de boules assez homogènes ou modérées (croissance polynomiale telles les grilles D-dimensionelles avec $D = O(\log n)$ ou croissance sous-exponentielle telles que des subdivisions d'arbres) peut être transformé en petit-monde navigable. Comme corollaire, tous les graphes de Cay-

Pour tout sommet u **Faire**
Dans H, tirer aléatoirement et indépendamment le contact distant L_u de u avec $\Pr(L_u = v) = \frac{1}{Z_u b_u(d(u,v))}$, Z_u étant une fonction de normalisation.

Listing 3.4 – Petit-mondisation

ley (représentatifs de groupes) connus jusqu'à ce jour sont petit-mondisables. D'autre part, nous avons montré que le produit de deux graphes petit-mondisables est un graphe petit-mondisable.

Schéma de preuve. Le principe est de montrer qu'en moyenne, en suivant l'algorithme `glouton+` , la distance dans le graphe H (pas G) est divisée par une constante au bout de $\Theta(\log n)$ sauts. Le graphe G obtenu est donc $O(\log^2 n)$-navigable. Ce schéma suit celui de Kleinberg dans [Kle00] avec une complexité légèrement accrue. Notre véritable contribution est la définition des liens longs pour cette grande famille de graphes.

Autres opérations de petit-mondisation. Depuis, de nouvelles familles de graphes ont été petit-mondisées. On peut citer les graphes à largeur arborescente bornée [Fra05], les graphes de dimension doublante bornée [AGGM06, Sli07], à mineur exclu [AG06],... On pourrait se demander si toute famille pourrait être petit-mondisée ? Malheureusement non, dans [FLL06], il a été montré que certains graphes ne peuvent pas être petit-mondisés car une fois augmentés, ils sont au mieux $n^{1/\sqrt{\log n}}$-navigable. D'un autre côté, tout graphe peut-être augmenté et devenir $O(n^{1/3})$-navigable [FGK+09].

3.2.4 Petit-mondisation rapide et distribuée

Le principal inconvénient pratique des différents schémas de petit-mondisation est qu'ils sont centralisés. Ils nécessitent une connaissance initiale complète du graphe H, le pré-calcul des liens longs est long. Dans [DHLS06], nous avons proposé un processus de petit-mondisation décentralisé extrêmement économique appliqué aux graphes de croissance $O(1)$-bornée.

Modèle distribué. En premier lieu, nous supposons que chaque noeud à la capacité de calculer sa distance à tout autre noeud dans le graphe de base H. Il s'agit de la connaissance de l'oracle. Par exemple, des techniques basées sur des étiquetages de distances permettent de répondre à de telles requêtes de manière distribuée [GPPR01]. En pratique, on peut aussi considérer que la distance peut être mesurée ou approximée grâce à des adresses IP ou des coordonnées géographiques. D'autre part, chaque noeud ne dispose que d'une *connaissance locale*, c'est-à-dire la liste de ses voisins. Vue d'un noeud, une *ronde de communication* correspond au temps écoulé entre deux envois. Il peut recevoir un nombre quelconque de messages dans cet intervalle. Notre algorithme

peut fonctionner dans un environnement *asynchrone* (l'ordonnancement des calculs est relativement arbitraire) mais impose localement des mécanismes de synchronisation.

Plus formellement, notre principal résultat est le suivant :

Théorème 3.2.1. *[DHLS06] Avec probabilité* $1 - 1/n$*, tout graphe de croissance* $O(1)$*-bornée de diamêtre* D *peut être augmenté en petit-monde avec :*

- *une mémoire locale de* $O(\log^2 n \log D)$ *bits ;*
- *un temps de* $O(\log n \log D)$ *rondes de communications ;*
- *Au total, chaque noeud reçoit et émet au plus* $O(\log n \log D)$ *messages de* $\log n$ *bits.*

Dans les sections suivantes, nous décrivons succinctemet les deux principales étapes (construction d'un réseau virtuel intermédiaire et des liens longs) et techniques qui nous ont mené au résultat.

Le réseau virtuel intermédiaire permet entre autre de calculer la cardinalité $b_u(r)$ (de manière exacte ou approximée) des boules quels que soient u et v. Ce genre de requêtes peut être intéressant indépendamment de l'objectif de petit-mondisation.

3.2.5 Un réseau logique intermédiaire

Le principal objectif est d'obtenir rapidement des approximations constantes de cardinalités $b_u(r)$ non pas pour toutes les combinaisons de paires de sommmet et rayon (u, r) mais pour un échantillon bien précis. Au préalable, nous devons faire un petit tour sur les notions qui nous ont été utiles aussi bien dans le processus de petit-mondisation distribué ...que dans l'application de recherche d'image dans une collection indexée 4.2.3. Ainsi, dans [DHLS06] et [DH08], nous avons défini un réseau logique intermédiaire $G_{T,c}$, calculable dans un environnement distribué. Les feuilles de l'arbre T désignent les noeuds du réseau ou des index sur les données et c est une constante supérieure à 1.

Domination, séparation et arbre d'échantillonage. L'ensemble $V' \subset V$ est une *r-domination* de V si $\forall v \in V$, $\exists v' \in V'$ tel que $d(v, v') \leq r$. L'ensemble $V' \subset V$ est une *r-séparation* si $\forall v, v' \in V'$, $d(v, v') > r$.

Nous considérons une *hiérarchie d'échantillons* de sommets : $S_h \subseteq S_{h-1} \subseteq \ldots \subseteq S_0 = V$ ayant comme propriétés :

- *couverture* : S_i est une 2^i-domination de S_{i-1}.
- *séparation* : S_i est une 2^i-séparation.

Un *arbre d'échantillons* T peut être construit à partir de la hiérarchie (en notant que chaque noeud peut apparaître dans au plus $h = \lceil \log_2 D \rceil$ échantillons). Chaque instance de noeud u stocke :

- $P(u) \in \{v \in S_{i+1} | \delta(u, v) \leq 2^{i+1}\}$ est l'*unique parent* de u, choisi arbitrairement parmi ceux à distance au plus 2^{i+1} de u.
- ses *enfants*.

La copie ou instance du noeud u de niveau i est noté $u^{(i)}$. $P_i(u^{(j)})$ est l'*ancêtre* de $u \in S_j, j < i$ au niveau i.

L'exemple de la figure 3.3 schématise l'arbre d'échantillon pour un chemin dont les sommets sont numérotés dans l'ordre $1, 2, 3, 5, 7, 8$ et 14. La distance entre chaque noeud correspond simplement à la différence des identifiants des noeuds.

Ajout d'arêtes de voisinage et cardinalité des boules. Soit c une constante appelée *coefficient de voisinage*. Nous définissons un ensemble *d'arêtes voisines* $E_i(c)$ pour l'ensemble des noeuds $u \in S_i$: $E_i(c) = \{(u, v) \in S_i | d(u, v) \leq c.2^i\}$.

Définition 3.2.2. *[DHLS06, DH08] Soient $T = (V(T), E(T))$ un arbre d'échantillons de G et c un coefficient de voisinage. On considère le graphe $G_{T,c}$ défini par :*

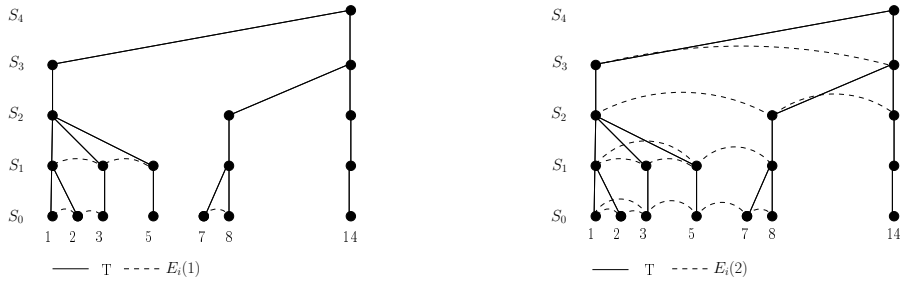
$$V(G_{T,c}) = V(T) \tag{3.1}$$

$$E(G_{T,c}) = E(T) \cup \{E_i(c)\}_{i \in [1,h]}. \tag{3.2}$$

où h est la hauteur de l'arbre.

Accès aux boules. Dans la figure 3.3, nous pouvons voir l'influence du coefficient de voisinage. Pour atteindre les sommets à distance au plus 2^i d'un sommet u, il suffit d'atteindre son ancêtre de niveau i, $u^{(i)}$, et de considérer toutes les feuilles des sous-arbres enracinés dans le voisinage de $u^{(i)}$: nous obtenons un ensemble X qui contient $B_u(2^i)$ mais qui est contenu dans $B_u((2+c)2^i)$. Dans [DHLS06], nous avons montré que $c = 4$ était nécessaire et suffisant. D'autre part, de par la propriété de croissance bornée, nous avons $b_u(2^i) = \Theta(b_u((2+c)2^i)$. Des astuces peuvent être trouvées pour gagner du temps pour les requêtes de cardinalité (stockage à la volée du nombre de feuilles d'un sous-arbre) ou calculer *exactement* la cardinalité d'une boule (parcours des feuilles et calcul de la distance à u - technique utilisée dans les implémentations de la section 4.2).

Dans la littérature, nous pouvons trouver des constructions comparables [GGN04, KL04, MHP06] à notre structure de données intermédiaire mais la difficulté consiste à définir une *version qui puisse être décrite de manière décentralisée*. C'est notre véritable contribution.



a) $G_{T,1}$. b) $G_{T,2}$.

FIGURE 3.3 – Influence du coefficient de voisinage. $G_{T,c}$ avec $id(V) = \{1, 2, 3, 4, 5, 7, 8, 14\}$ and $\forall x, y \in V, d(x, y) = |id(x) - id(y)|$.

Construction des liens longs

Sans rentrer dans les détails, pour chaque noeud u, le principe consiste à choisir de manière aléatoire uniforme une échelle de distance i entre 1 et h, c'est-à-dire une distance 2^i et sélectionner une feuille aléatoire dans l'ensemble de sommets X. Cette opération peut être réalisée rapidement : chaque sommet de l'arbre pouvant sélectionner et stocker de manière anticipée un nombre logarithmique de feuilles dans lequel chaque sommet va se servir.

3.2.6 Perspectives : Émergence et fiabilité des petits mondes

Émergence. Le principal challenge est de proposer des constructions distribuées de petit-monde qui soient les moins coûteuses possibles. Étudier des processus de petit-mondisation décentralisés et rapides a aussi un intérêt d'ordre fondamental :

– comment se construisent les petits mondes réels ?

Il a été observé que des réseaux sociaux, de transports aérien, ... sont des petits-mondes selon diverses définitions. Cependant, les processus qui mènent à cette constatation sont mal compris. Typiquement, nous verrons dans la section 3.3.1 comment expliquer une des caractéristiques fréquemment observées sur la distribution des degrés (ressemblant à une loi de puissance). Cependant, il n'existe pas de modèle totalement convaincant pour expliquer l'apparition de la propriété de navigabilité. Notre algorithme est une première étape. Cependant, sa nature très algorithmique est difficilement transposable dans le cadre d'un réseau social. Le très récent travail de Chaintreau, Fraigniaud et Lebhar [CAFL08] est plus attractif de ce point de vue là mais souffre, á priori, d'une lacune. A notre connaissance, le temps de convergence vers un petit-monde est long et ne suffit pas à expliquer l'émergence rapide des petits-mondes dans un réseau social.

Fiabilité. L'aspect concernant la fiabilité a été peu exploré. Par nature, les algorithmes gloutons sont auto-stabilisants. Si des erreurs transitoires ont lieu concernant la connaissance de l'oracle, il se peut que le routage boucle temporairement mais la cible sera atteinte dès que la connaissance de l'oracle sera redevenue exacte. Si le réseau contient de manière permanente des erreurs, il y a fort à parier que l'algorithme générique ait un bon comportement. L'expérience menée dans la section 4.2 a tendance à le confirmer.

3.3 Modèles de graphes non navigables

On peut considérer que la propriété de navigabilité est avant tout due à la structure des graphes réels : petit diamêtre, existence de nombreux petits cycles, degré moyen faible, distribution des degrés hétérogène suivant une loi de puissance, ...En réalité, la possibilité ou non de choisir/renommer les identifiants est absolument fondamentale. On appele ainsi le modèle *avec indépendance des noms* dans lequel il est impossible[2] d'effectuer un pré-traitement de nommage des noeuds.

2. Quelle que soit la raison : coût prohibitif, impraticable suivant le contexte, ...

Type	Modèle	modèle de connaissance	Borne infèrieure
Aléatoire pur	Chung–Lu	Faible	$\Omega(\frac{n}{\log^{\ell} n})$
	Chung–Lu	Fort	$\Omega\left(\frac{n}{\log^{\frac{\ell(\ell-1)}{\ell-2}}(n)}\right)$
Evolutif	Cooper–Frieze	Faible	$\Omega(n^{1/2})$
	Δ-out Móri	Faible	$\Omega(n^{1/2})$ si $\beta > -1$
	Δ-out Móri	Fort	$\Omega(n^{\frac{\beta}{4+2\beta}-\epsilon})$ si $\beta > 0, \epsilon > 0$
	1-out Móri	Fort	$\Theta(n)$ si $\beta \geq 0$

TABLE 3.1 – Bornes infèrieures sur le temps de routage.

Nous nous intéressons dans cette section à une propriété de *non-navigabilité* de différentes familles de graphes sans échelle dans le modèle avec indépendance des noms. Un graphe est *sans-échelle* si le nombre de sommets de degré δ est proportionnel $n(1/\delta)^c$ avec $c > 1$. Dans la littérature, certains considèrent une équivalence entre graphes sans échelle et petit monde. Cependant, sans vouloir nier l'apparition de graphes sans échelle dans bon nombre de réseaux réels, la distribution de degrés n'a rien à voir avec la propriété de navigabilité [3].

Dans [DEH07] et dans un article de synthèse à venir, nous montrons que différentes familles de graphes sans échelle *ne sont pas des petits-mondes* et plus précisément nécessitent au moins $\Theta(n^{\Theta(1)})$ traversées de noeuds pour atteindre une cible lors d'un routage (rappelons que les noeuds ne disposent pas de table de routage) alors qu'il peut exister des chemins de longueur polylogarithmique. Pour au moins 1 modèle (1-out Mori), le graphe est non navigable. Ceci est vrai *quel que soit l'algorihme de navigation* car nous ne nous focalisons pas sur des algorithmes de routage glouton dans cette section. Ainsi, nous ne donnons aucune contrainte de mémoire sur l'agent mobile qui ferait une recherche, nous l'autorisons à se "téléporter" à tout noeud déja visité. La seule véritable contrainte est la connaissance des noeuds : dans le modèle *faible*, les noeuds ne connaissent pas les identités de leurs voisins mais savent seulement comment les atteindre ; dans le modèle *fort*, l'identité des voisins est connue. Le modèle fort est plus réaliste mais le modèle faible sert de brique de base dans nos preuves. Pour résumer, la notion de navigabilité abordée ici est moins restrictive que celle considérés pour les petits-mondes car l'agent mobile est plus puissant.

Nos résultats sont résumés dans le tableau 3.1. La définition des différents paramètres n'est pas explicitée ici mais sert uniquement pour illustrer précisément le type de résultat obtenu. Dans le cas des modèles de type *aléatoire pur*, les identifiants sont absolument quelconques et ne peuvent aider à la navigation. Cependant l'existence de noeuds de grand degré (quasiment linéaire dans certains cas) a fait penser à certains auteurs que l'algorithme consistant à d'abord chercher les noeuds de plus haut degré était efficace (visite de $O(\log^{O(1)} n)$ noeuds) pour accéder à un noeud quelconque. Nous prouvons ici le

3. Au mieux, si on autorise lors d'une petit-mondisation à ajouter un nombre variable de liens par sommet, cela peut aider à réduire le diamêtre et à accélérer le processus de routage glouton [FG09]

contraire. Le résultat sur les graphes évolutifs est plus surprenant. En effet, les identifiants correspondent cette fois-ci à l'ordre d'apparition des noeuds. Cette information ne s'avère pas assez pertinente pour une recherche efficace.

3.3.1 Modèles de graphes sans échelle

Dans cette section, nous donnons les points clefs des modèles utilisés. Leur formalisation n'est pas donnée ici car elle est trop technique et un peu longue pour être décrite dans le cadre d'une synthèse. On peut distinguer deux types de familles de graphes aléatoires : *les graphes aléatoires purs* et les *graphes aléatoires évolutifs*.

Graphes aléatoires purs. Le graphe aléatoire est construit à partir d'une distribution de degré fixée [MR95] (ou possiblement une séquence de degrés attendue dans [CL02, CL03, CL06]). Dans un tel modèle, il est important de remarquer que les degrés des voisins sont *indépendants*. Aiello *et al.* [ACL00, CL02] se sont concentrés sur les distributions de loi de puissance et ont montré l'émergence d'une composante géante selon certaines conditions, prouvant que la plus grande composante géante peut avoir une taille linéaire et un diamètre logarithmique. Cependant, les modèles de graphes purs ne peuvent pas expliquer l'apparition de loi de puissance dans les graphes réels.

Modèles évolutifs. Les graphes aléatoires évolutifs consistent en une tentative de modélisation de graphes réels et d'explication de l'émergence de la loi en puissance. Au départ de la construction, on se fixe un graphe de petite taille (un simple sommet, arête ou boucle suffisent) et à chaque unité de temps, de nouveaux sommets se connectent aux existants. L'extrêmité d'une nouvelle arête est choisie aléatoirement, soit uniformément (*attachement uniforme*) ou avec une probabilité proportionnelle au degré du sommet extrêmité (*attachement préférentiel*). Dans les modèles les plus simples, on ajoute un seul sommet et une seule arête à chaque unité de temps. Des variantes avec des degrés sortants plus élevés peuvent être obtenues soit en ajoutant plus d'arêtes par unité de temps ou en construisant un graphe avec $n\Delta$ sommets et en fusionnant chaque séquence de Δ sommets consécutifs en un seul ; on obtient ainsi un graphe avec n sommets et $n\Delta$ arêtes, de manière à ce que le degré moyen soit 2Δ. Dans ces modèles, *l'âge et le degré* sont corrélés. En particulier, les degrés des voisins ne sont *pas indépendants* et les techniques d'analyse de champs moyens ne peuvent pas être utilisées. Ceci est une véritable différence avec les modèles aléatoires purs en ce qui concerne l'analyse de processus de recherche.

La plupart des modèles basés sur l'attachement préférentiel ainsi que leur propriétés peuvent être trouvés dans les travaux de Bollobás et Riordan [BR03]. Le concept d'attachement préférentiel est ancien [Yul25], datant de 1925, et a été oublié puis redécouvert, notamment par Barabási et Albert [BA99] qui ont effectivement motivé son usage pour la modélisation de graphes réels. Dans [BR04], Bollobás et Riordan ont donné une description mathématique précise du modèle de Barabási-Albert (BA). Dans [BRST01], il a été montré rigoureusement que la proportion $P(x)$ des sommets de degré x converge asymptotiquement vers une loi de puissance pour $x \leq n^{1/15}$. Puis, dans [BR04], il a été

prouvé que pour $\Delta \geq 2$ le graphe est connexe avec grande probabilité et que le diamètre est asymptotiquement égal à $\log n / \log \log n$ alors que pour $\Delta = 1$ le diamètre est approximativement $\log n$ (voir [BR03]). D'autres variantes du modèle BA ont été proposées [BO04, SDS00, DEM01, YsZwCdL05] et pour chacune d'elles, des graphes sans-échelle ont été obtenus.

Des extensions à des modèles plus généraux combinant attachement préférentiel et uniforme ont été étudiées dans [CF03, LLYD02, Mor03, Mor05]. La figure 3.4 montre un exemple de 2-Mori dans lequel on observe l'hétérogénéïté des degrés. Les sommets les plus anciens sont ceux de plus grands degrés. En résumé, pour chacun de ces articles, chaque sommet nouveau choisit avec probabilité p et $1-p$ s'il utilise un attachement uniforme ou préférentiel. Dépendant de la valeur du paramêtre p, la loi en puissance a été observée et prouvée par Cooper et Frieze [CF03] (dans un modèle très général dans lequel des arêtes supplémentaires entre "vieux" sommets peuvent aussi être ajoutées) ou par Móri [Mor03, Mor05]. Dans ces modèles, les auteurs donnent une limite asymptotique du degré maximal, qui tend à être de la forme n^c pour une constante explicite c dépendant des paramètres du modèle (mais qui est différent de ce qu'on obtiendrait si on extrapolait la loi en puissance jusqu'à $P(d) = 1/n$ car la distribution sans échelle n'est valide que pour les relativement petits degrés).

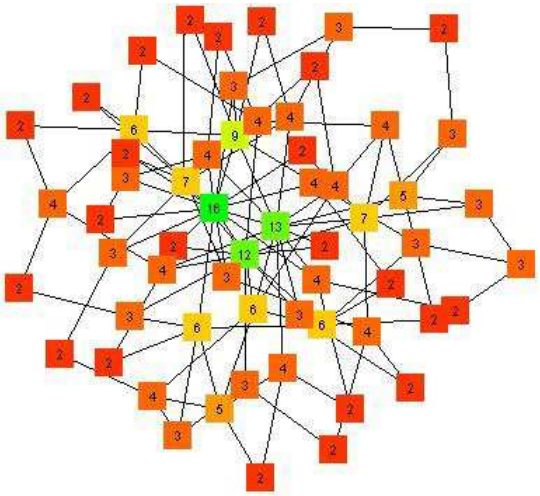


FIGURE 3.4 – Exemple d'un 2-Mori. Les degrés sont indiqués dans chaque sommet.

3.3.2 Indistinguabilité des noeuds

Nos preuves de bornes inférieures sont basées sur une notion probabiliste d'équivalence des noeuds. Intuitivement, les noeuds sont équivalents si leur identités peuvent être échangées sans modifier la distribution de probabilité sur les graphes aléatoires. Ainsi, si on considère un ensemble de noeuds ayant des identités proches, comprises

entre $[t - \Theta(t^c), t + \Theta(t^c)]$, l'identité de ces noeuds ne permet pas de les distinguer d'un point de vue de la recherche. Typiquement, il est fortement improbable qu'il existe une arête entre ces noeuds. Dans le cas du modèle faible, la recherche se comporte comme si elle était effectuée dans un graphe anonyme (noeuds sans identifiants). Il faudrait donc visiter au moins $\Theta(t^c)$ noeuds pour atteindre la cible. La traduction en modèle fort tient compte cette fois de la distribution des degrés.

Chapitre 4

Contribution à la recherche d'information

Dans les deux premières sections de ce chapitre, nous considérons la requête consistant à trouver une cible représentant un item dont l'adresse ou l'identifiant sont inconnus. Nous pouvons à nouveau utiliser le paradigme d'*oracle* comme décrit dans le chapitre 3. Cependant, nous considérons maintenant un *oracle défaillant*. La réponse à une requête n'est pas fiable. Cette incertitude est réaliste à plusieurs points de vue :

- Dans un réseau dynamique, il est impossible que tous les noeuds soient perpétuellement à jour.
- L'imprécision de l'indexation, et celle potentielle de la définition de la cible, ne permet pas de répondre avec exactitude. C'est typiquement le cas pour les documents multimédia. Par exemple, je cherche une image pour illustrer un article (sur les Pyrénées) sans une idée bien définie sur le contenu de l'image.

Un succés à une recherche d'information n'a lieu qu'à l'atteinte exacte du noeud possédant l'item recherché. Accéder à un noeud "voisin" ne suffit plus. Cela correspond au modèle faible énoncé dans la section 3.3.1. La première section traite d'un modèle alors que la deuxième section est bien plus appliquée. Dans le cadre de la thèse d'Anthony Don, nous avons expérimenté des conceptions de graphes navigables dans le domaine de la *visualisation d'informations*. L'objectif est de proposer à un utilisateur, donc humain, une interface de navigation dans des collections d'images indexées qui l'incite à utiliser, consciemment ou non, les algorithmes gloutons de recherche de la section 3.1.1.

La dernière section traite de recherche en cours. Il s'agit de proposer des mécanismes de pré-traitement de systèmes de gestion de bases pour optimiser les temps de requêtes. Les requêtes sont typiquement les requêtes formulées en langage SQL.

4.1 Un modèle de recherche par agents mobiles pour un réseau non fiable

4.1.1 Contexte

Le point de départ de cette problématique est la constatation que l'algorithmique des agents mobiles tient peu compte de plusieurs aspects qui nous semblent importants : la topologie réelle des réseaux, la limitation (ou non) des ressources mémoire, les conséquences de la corruption de la mémoire ou du code, ...

Les agents mobiles font référence à des programmes informatiques identifiables et autonomes qui peuvent se déplacer à l'intérieur d'un réseau et qui peuvent agir au nom d'un usager ou d'une autre entité. La plupart des recherches courantes sur le paradigme des agents mobiles poursuivent deux objectifs : la réduction du trafic de réseau et l'interaction asynchrone. Ces deux objectifs découlent directement du désir de réduire la surcharge d'information et d'utiliser efficacement les ressources du réseau. Il existe un grand nombre de motivations pour utiliser le paradigme d'agent mobile. Cependant, la recherche intelligente d'information, le commerce électronique, la gestion de réseaux, la gestion de la mobilité et les services évolués de télécommunication demeurent actuellement les applications les plus souvent visées dans ce contexte et font l'objet de systèmes d'agents mobiles. Le domaine des télécommunications se prête donc particulièrement à ce genre de technologie autour de laquelle une communauté très dynamique de chercheurs s'active à concevoir des solutions novatrices aux problèmes touchant les domaines mentionnés.

4.1.2 Un modèle de réseau incertain

Ne connaissant pas l'identité, l'adresse ou le chemin menant au nœud cible, l'agent va interroger des nœuds contenant des indications et élaborer au fur et à mesure son cheminement. Dans [HKK04], nous avons proposé un modèle de recherche d'informations dans lequel chaque nœud pointe sur un lien menant à la destination. Le noeud voisin indiqué est sensé appartenir à un plus court chemin vers la cible. Il peut arriver que certains nœuds soient défaillants, pas à jour ou ne sachent pas donner d'indication. Ces nœuds sont désignés sous le terme générique de *menteurs* et pointent sur un voisin qui éloigne de la cible. Malheureusement, l'agent ne connait pas l'état du noeud qu'il interroge. Bien entendu, si le réseau n'est composé que de noeuds menteurs, seule une exploration totale permet de trouver la cible. Nous supposons que le nombre de menteurs est k. Suivant les modèles, l'agent connait ou non une borne sur k. Nous nous intéressons au pire cas : un adversaire connait l'algorithme de l'agent et place les k menteurs à sa guise [1]. La principale mesure de performance est le *temps de recherche*, exprimée en nombre de noeuds visités. La distance entre le noeud source et la cible est notée d.

1. Le premier modèle de recherche en présence de menteurs a été proposé par Kranakis et Krizanc [KK99]. Ils ont analysé des algorithmes de recherche dédiés aux anneaux et au tore. Cependant, chaque noeud est supposé être menteur ou fiable avec probabilité constante.

4.1.3 Résumé des résultats

Ce travail a été présenté dans deux articles [HKK04, HKKK08] et un troisième article est en cours sur ce sujet. Je présente ici une synthèse rapide.

Une première dichotomie s'impose : les algorithmes *universels* donnent une solution quel que soit le graphe considéré alors que les algorihtmes *dédiés* sont spécifiques à une famille de graphes. Notre premier article [HKK04] sur le sujet traite essentiellement d'algorithmes déterministes dédiés aux graphes complets, anneaux, tores, hypercubes et arbres de degré borné. En particulier, pour les arbres de degré borné, nous avons prouvé que le temps de recherche est $\Omega(d+2^{\min\{k,d\}})$ [HKK04] pour *tout algorithme de recherche.*

Nous avons ensuite proposé un premier algorithme *universel et sans mémoire* dans [HKKK08]. Il s'agit d'une marche aléatoire biaisée (BRW pour "biasest random walk") :

- BRW(q) : avec probabilité q, l'agent suit le conseil donné par le noeud. Sinon, un voisin est choisi de manière aléatoire uniforme parmi ceux non conseillés.

Ainsi, pour $q > 1/2$, nous avons montré que l'espérance du temps de recherche est majoré par $O(d + r^k)$ avec $r = \frac{q}{1-q}$ [HKKK08] pour tout graphe de degré borné. De plus, ce résultat est fin car BRW met un temps $\Omega(d + r^k)$ pour le tore [HKKK08] alors qu'il existe un algorithme déterministe dédié mettant un temps $d + O(k)$. Cette borne est un peu décevante et nous nous sommes posé la question sur l'éventuelle conception d'algorithmes universels plus efficaces. Cependant, BRW s'avère plus efficace que n'importe quel algorithme déterministe pour certaines topologies. Par exemple, dans le cas du graphe complet, le temps de recherche devient constant alors que tout algorithme déterministe met un temps $\Omega(k)$. D'autre part, les marches aléatoires biaisées n'ont pas besoin de l'historique des nœuds visités pour fonctionner et n'ont besoin que d'un espace mémoire constant. Les algorithmes probabilistes semblent donc plus intéressants d'un point de vue pratique.

4.1.4 " Expanders "

Le challenge réel est l'obtention d'algorithmes universels pour les graphes *expanders*. Ce sont les graphes à croissance exponentielle rencontrés dans de nombreuses circonstances : petit-monde, graphes sans échelle, graphes aléatoires Δ-réguliers dès que $\Delta > 2$, ... Les graphes expanders sont des graphes creux (degré moyen "petit"), fortement connectés et jouent un rôle clef en informatique fondamentale et dans la théorie des réseaux de communications (voir [HLW06] pour un survol). Formellement, un graphe $G = (V, E)$ est un c-expander si, pour chaque sous-ensemble de sommets $X \subset V$ avec $|X| \leq |V|/2$, on a $|\Gamma(X) \setminus X| \geq c|X|$ où $\Gamma(X)$ est l'ensemble des voisins de X.

A titre indicatif, le réseau $G_{T,c}$ décrit dans la section 3.2.5 est aussi un expander.

4.1.5 Nouveaux résultats

Les résultats de cette section ne sont pas encore parus. Il s'agit d'un travail commun avec David Ilcinkas (LaBRI), Adrian Koswoski (Université de Gdansk) et Nicolas Nisse (Inria Sophia Antipolis). Nous avons proposé différents algorithmes universels de

recherche. Tous les algorithmes sont basés sur des algorithmes canoniques : A consiste à suivre les conseils, R est une marche aléatoire pure[2] et E est une exploration correspondant à un parcours en largeur. On peut imaginer de nombreuses combinaisons d'algorithmes en alternant des phases issues de ces briques de base. Cependant, la durée des phases est trés importante. Par exemple, si la durée des phases de conseils A est bien supérieure à la durée des phases aléatoires R, il se peut que l'agent mobile soit bloqué pour toujours dans la même région du graphe.

Si on se concentre sur l'algorithme R/A : l'agent mobile alterne des phases R de durée t_R et des phases A de durée t_A, nous obtenons notamment que :

- Pour une famille de graphes *expanders*, les *graphes aléatoires* Δ*-régulier*, l'algorithme R/A trouve sa cible en temps $O(k^{O(1)} \log^{O(1)} n)$ avec $t_R = t_A = \Theta(\log n)$;
- Pour les grilles D-dimensionnelles, l'algorithme R/A trouve sa cible en temps $2d + O(k^5) + o(d)$ avec $t_R = t_A = \Omega(k^3)$.

Ce dernier résultat montre l'intérêt de l'algorithme universel R/A par rapport à BRW qui met un temps $\Omega(d + 2^k)$ pour les grilles D-dimensionnelles. La technique de preuve est basée sur l'étude de la vitesse de déplacement vers la cible en fonction de la densité locale de menteurs. Bien que la famille de graphes soit simple, la preuve est relativement complexe. Elle n'est donc pas détaillée ici.

Le résultat sur les graphes aléatoires Δ-réguliers est très surprenant. En effet, il s'agit de graphes *expander* pour lesquels le voisinage de la majorité des sommets est arborescent : plus précisément, pour la majorité des sommets, le sous-graphe induit des $\Theta(n^{1/2})$ plus proches sommets est un arbre. Sachant que $d + 2^k$ est un minorant[3] du temps de recherche pour tous les arbres quel que soit l'algorithme, il n'était pas évident d'aboutir à un temps polynomial. Seulement, dans les graphes aléatoires Δ-régulier, la phase R permet de choisir *quasiment de manière aléatoire uniforme* des sommets de départ intermédiaires extrêmement rapidement. Ainsi, même si un adversaire a la connaissance du point de départ initial et de l'algorithme, il ne peut savoir où se situe l'agent mobile à la fin du phase R. Du coup, le pire placement des menteurs consiste à les localiser tout autour de la cible, c'est-à-dire à une vraie proximité de la cible. La chance aidant, une marche aléatoire qui démarre à proximité de la cible, disons $\log k$, a une probabilité relativement grande, $\Omega(k^{-\Theta(1)})$ d'aboutir à la cible.

De manière générale, l'algorithme universel R/A est une généralisation de l'algorithme `glouton+`. L'ajout des phases aléatoires permet d'éviter de rester bloqué dans des culs-de-sac si le graphe de navigation n'est pas navigable.

4.1.6 Vers des graphes de navigation tolérant des erreurs

Dans le modèle étudié, le pire graphe de navigation semble être l'arbre. C'est malheureusement la structure utilisée par défaut dans de nombreuses circonstances : arborescence de fichiers, ... mais on peut être amené à visiter de grands sous-arbres avant d'être

2. L'agent ne tient pas compte des conseils et choisi le noeud suivant parmi les voisins de manière aléatoire uniforme.

3. pour $k = O(\log n)$

convaincu que la cible ne s'y trouve pas. Cette constatation est valable dans d'autres modèles d'erreurs (menteurs probabilistes par exemple). L'algorithme universel R/A est prometteur aussi bien pour les expanders que pour les grilles D-dimensionelles. On peut parier d'ailleurs que ce résultat est généralisable auw graphes à croissance modérée. D'une certaine manière, tout graphe pour lequel il existe de nombreux courts chemins disjoints semble pertinent.

4.2 Application à la recherche d'images dans une collection indexée

Les analyses théoriques des différents algorithmes de routage glouton nous ont amené à proposer une nouvelle approche pour le problème de la recherche d'image dans une collection indexée.

4.2.1 Contexte

Comment organiser, présenter et rechercher des documents multimédia ?
Cette question se pose dans de nombreux cadres :
- Moteur de recherche d'images ;
- Chapitrage automatique et obtention de résumés de documents vidéo ;
- Organisation et gestion de collections d'images personnelles.

De nombreuses difficultés se dressent devant nous :
- les données sont massives et complexes : les pré-calculs nécessaires à l'indexation sont donc lourds ;
- *le fossé sémantique*[4] complique énormément la pertinence des indexations.
- A ce jour, peu de descripteurs (résumés du document) rapidement calculés sont *interprétables* pour un humain.
- L'*évaluation* de la pertinence d'un système de recherche combinant indexation, représentation et intéraction est délicate.

En résumé, nous sommes face à des contraintes de **temps** et **mémoire** de pré-calculs, d'**imprécision des descripteurs et des index** et d'évaluation de **qualité** d'une solution. Les trois premiers points sont typiquement des mesures évaluées dans nos différents modèles de routage ou de recherche d'informations.

Les descripteurs utilisés sont calculés à partir de contenu visuel (images principalement). Notre cadre de travail est donc *les systèmes de recherche d'images basés sur le contenu*. Souvent, les descripteurs sont des vecteurs et nous manipulons des mesures de dissimilarité.

Requête. Notre objectif principal est avant tout de trouver une image, dite *cible*, dans une collection indexée. Cela peut être une image dont on a un vague souvenir dans une collection personnelle, une image pour illustrer un document, ... Si l'*adresse* (nom,

4. Informellement *le fossé sémantique* représente l'écart entre des informations extraites automatiquement et celles interprétées par un humain.

localisation, index, descripteur) de la cible est connue avec exactitude, il s'agit d'une requête de type *routage*. Cependant, la pratique veut que cela ne soit pas vraiment le cas. C'est la raison pour laquelle, nous considérons qu'il s'agit de *recherche d'information*.

Descripteurs et espace métrique. Dans notre travail, nous avons utilisé le descripteur Color Layout de MPEG7 [BMRS02] qui résume bien la distribution des couleurs dans l'image. Nous avons aussi considéré un descripteur de luminance pour des petites images (niveaux de gris de 64×64 pixels). Nous avons aussi proposé un descripteur *interprétable* par un humain. Ce dernier descripteur, nommé ID, est composé des informations suivantes :

1. Nombre de visages de l'image (annoté de manière supervisée) ;
2. Le ratio de l'image occupé par des visages ;
3. Le nombre de régions de l'image obtenu par une segmentation de quantification de couleur ;
4. La luminance moyenne du descripteur "couleurs dominantes" de MPEG7 [BMRS02].
5. La saturation la plus élevée (Espace HSV) du descripeur "couleurs dominantes" de MPEG7.
6. La teinte (Espace HSV) de la couleur de plus forte saturation du descripeur "couleurs dominantes" de MPEG7.

Les descripteurs considérés peuvent être modélisés par un ensemble métrique (P, δ) défini sur un ensemble $P = \{p_1, \ldots, p_n\}$ avec une fonction de distance $\delta : P \times P \rightarrow \mathbb{R}^+$. Chaque image est représentée par un point de l'espace métrique et les distances représentent la dissimilarité entre images. Il n'y a aucune raison qu'en général des dissimilarités puissent correspondre à un espace métrique mais c'est le cas pour les descripteurs considérés. Nous notons $A = \frac{\max_{p_i,p_j \in P}(\delta(p_i,p_j))}{\min_{p_i,p_j \in P}(\delta(p_i,p_j))}$ le *ratio d'aspect* de l'espace métrique (P, δ).

Un des paramêtres très important est la *dimensionnalité intrinsèque des données* qu'il ne faut pas confondre avec la dimension de représentation ou codage des données. Par exemple, les points du cercle unité peuvent être exprimés avec deux coordonnées cartésiennes dans le plan mais une seule coordonnée suffit en représentation polaire. De manière générale, les descripteurs manipulés peuvent être décrits en très grande dimension mais être aussi plongés dans un espace de petite dimension. Nous avons choisi comme mesure de dimensionnalité intrinsèque la *dimension doublante*. En voici une définition.

Pour chaque point $p \in P$, $c_p(r)$ est le nombre minimum de boules de rayon r requis pour couvrir tous les points de $B_p(2r)$, c'est-à-dire qu'il existe un plus petit ensemble $P' \subseteq P$ de cardinalité $c_p(2r)$ qui est une r-domination de $B_p(2r)$. La *dimension doublante* de (P, δ) est la valeur maximale de $ddim = \log_2 C_p(r)$ pour toutes les valeurs $r \in \{2^i | i = 0 \ldots \log_2 A\}$ et chaque point $p \in P$.

4.2.2 Limitations des systèmes de recherche existants

Requêtes par l'exemple. Le principe consiste soit à spécifier précisément un ou plusieurs descripteurs ou de donner un exemple précis (image de référence, dessin grossier).

Dans les deux cas, cela revient à fixer une valuation au descripteur et de faire des requêtes de type *requêtes des plus proches points* dans un espace de dimension éventuellement grand. Les inconvénients sont multiples : le processus d'interrogation n'est pas agréable, l'intéraction est quasiment inexistante, la pertinence de la réponse est assez aléatoire à cause du fossé sémantique, le temps de réponse peut être très long à cause de la **malédiction de la grande dimension**[5].

Bouclage de pertinence. Cette approche est très intéractive et intuitive. On propose à l'utilisateur un échantillon d'images et l'utilisateur indique quelles sont les images ayant un rapport avec la cible (exemples positifs) et éventuellement celles qui sont très dissimilaires de la cible (exemple négatifs). Un calcul sur le sous-espace (de l'espace métrique des descripteurs) potentiellement proche de la cible est effectué et un nouvel échantillon issu de cet espace est à nouveau proposé. Cette intéraction est itérée jusqu'à l'obtention de la cible. Le principal inconvénient est qu'il faut parfois préciser un grand nombre d'exemples positifs et négatifs.

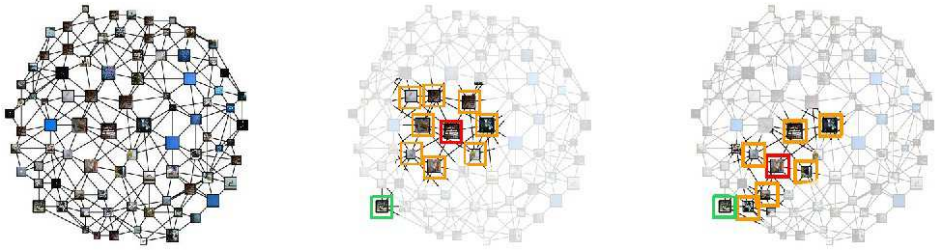
4.2.3 Recherche d'images par l'approche locale

Nous avons proposé dans [DH08] une approche basée sur un graphe de navigation : Chaque nœud du graphe de navigation (cf. Figure 4.1 dans laquelle le noeud rouge représente l'image courante et les noeuds oranges son voisinage) représente une entité visuelle (image ou ensemble d'images similaires). La conception d'un tel graphe nécessite de définir les liens de manière à ce que le graphe résultant soit navigable (cf Chapitre 3). De nombreuses difficultés sont à prévoir : la construction de graphes de taille « raisonnable » est très difficile ; la faible teneur sémantique des descripteurs images et des vidéos (norme MPEG 7) empêche de construire des graphes réellement navigables... et l'utilisateur est un humain. D'un point de vue intéractif, notre système fonctionne similairement aux systèmes de bouclage de pertinence :

- on propose un échantillon d'images et l'utilisateur désigne une image *voisine* qui lui semble la plus similaire à la cible.
- Un nouvel échantillon est proposé.

Cependant, la nouveauté est le calcul du nouvel échantillon : il s'agit des voisins dans le graphe de navigation. L'espoir est donc que *si le graphe est navigable, on se rapproche de la cible à chaque itération.* Cette approche est dite *locale* car le processus de décision est fait à partir du voisinage de la position courante dans le graphe de navigation. Le processus de recherche correspond exactement à l'algorithme `glouton+` (cf. section 3.1.1). Les noeuds *menteurs* modélisent le phénomène de fossé sémantique et les erreurs faites par un utilisateur humain lors de ses choix.

5. En général ce temps est exponentiel par rapport au nombre de dimensions. Plus généralement, la malédiction de la grande dimension implique que les données ne sont pas aisément indexables en grande dimension.



a) b) b)

FIGURE 4.1 – a) Graphe de navigation complet. b) orange/rouge : éléments montrés à l'utilisateur, vert : cible. c) on se rapproche

Limitations des approches locales existantes. Deux solutions existent (SWIM de Androutsos *et al.* [AKP+04] (le graphe de navigation est un anneau augmenté de k plus proches voisins) et NN^k-networks de Heesh et Ruger [HR04] (composé d'un plus proche point pour différentes combinaisons de descripteurs). Les inconvénients de ces graphes de navigation sont les suivants :

- *non-navigable* : il n'y a pas de garantie de succés car il existe des puits et on peut se heurter à des problèmes de *limite de perception*, c'est-à-dire que les voisins sont trop similaires et l'utilisateur devient incapable de choisir.
- *diamètre non borné* : ceci implique que le temps de recherche ne l'est donc pas non plus.

Conception du graphe de navigation

L'étude de la recherche d'information avec menteurs montre que l'arbre n'est pas une topologie appropriée. Par exemple, l'arbre T d'échantillons de la figure 4.2 construit à partir de 20 images n'est pas navigable. En prenant comme source la racine et comme cible une des 4 premières feuilles, on peut être amené à visiter tout le sous-arbre droit pour rien. Par contre, le graphe de navigation $G_{T,c}$ tel que défini dans la section 3.2.5, est navigable. Le descripteur considéré est Colour Layout, un descripteur de MPEG7, qui caractérise la distribution des couleurs dans l'image. La mesure de dissimilarité associée induit un espace métrique. Nous avons montré dans [DH08] la proposition suivante :

Théorème 4.2.1. *En partant du noeud racine (de plus haut niveau) de $G_{T,c}$, l'algorithme* `glouton+` *(resp.* `glouton` *) atteint sa cible en* $\log A$ *sauts (resp.* $O(2^{O(ddim)} \log A)$*) si et seulement si* $c \geq 6$.

4.2.4 Expérimentations

Dans [DH08, Don06], nous avons proposé une interface de navigation basée sur $G_{T,c}$ avec $c = 6$. Il est important de signaler que plus la valeur c augmente, plus le degré maximal du graphe est important (de l'ordre de $2^{O(cddim)}$).

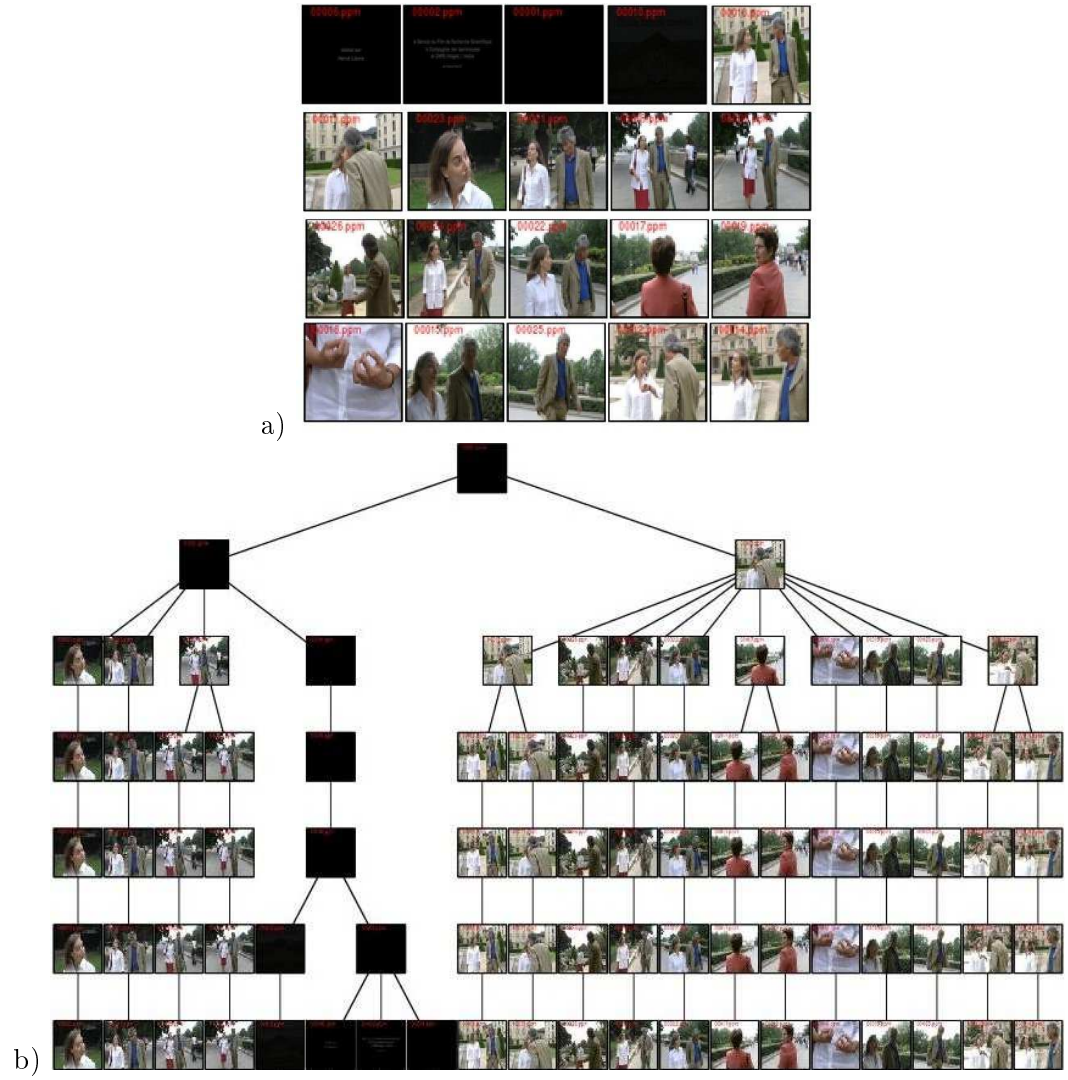


FIGURE 4.2 – a) Données initiales. b)Arbre d'échantillons T pour les images indexées par Colour Layout.

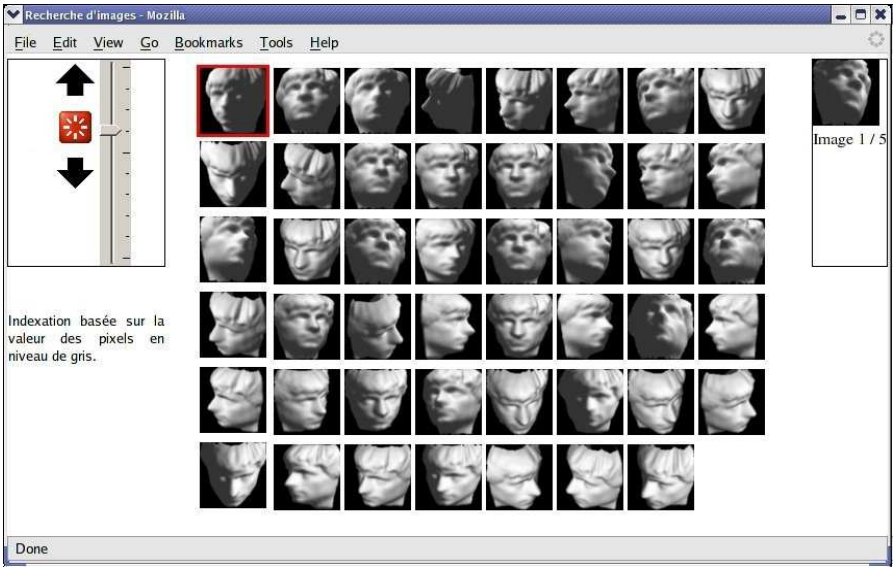


FIGURE 4.3 – Interface de navigation

L'utilisateur peut ainsi se promener dans $G_{T,c}$ en ayant uniquement une vision locale. Dans la figure 4.3, l'image courante est la première (encadrée en rouge), on ne montre que les images voisines du niveau courant et la cible est représentée à droite. La possibilité de monter/descendre dans les niveaux peut être vue comme des opérations de raffinement (zoom) ou de généralisation (dezoom).

Nous avons mené 5 expériences de recherche d'images avec 11 utilisateurs. Chaque expérience consiste à trouver une séquence de 5 images différentes, impliquant un total de 55 tâches de recherche par expérience. Les images cibles ont été choisies de manière aléatoire et sont les mêmes pour tous les utilisateurs. La table 4.2.4 décrit les paramètres de chaque expérience. Pour obtenir une comparaison, chaque expérience est aussi répétée avec un affichage complet de la collection (*recherche linéaire*) ainsi qu'avec le graphe de navigation $G_{T,6}$ (*navgraphe*). La première collection (faces) est un exemple dans lequel toutes les images sont perceptuellement trés similaires et la tâche de recherche est délicate. Dans la seconde collection, l'objectif principal est de montrer l'impact du caractère interprétable du descripteur.

Les mesures de performances sont

- *le temps de recherche :* le nombre de clics (sauts) et le temps de recherche (en secondes) ;
- *le taux de réussite* ;
- *la charge cognitive*, c'est-à-dire la quantité d'informations que l'humain doit traiter. Cela correspond exactement au nombre d'images affichées simultanément. Il s'agit du degré des sommets du graphe de navigation.

En résumé (voir [Don06, DH08] pour l'ensemble des résultats), dans nos expériences, le temps médian de recherche est de l'ordre de 25s pour le graphe de navigation et de 30s

Exp.	Collection d'images	Descripteur et mesure de dissimilarité	Interface
1	265 images d'un même modèle de visages (`faces`) avec 3 degrés de liberté	Luminance $\in [0,1]^{4096}$ et norme $L1$ ($\lceil \log(A) \rceil = 6$).	NAVGRAPH $c = 6$. ($H = 5$, $\bar{\Delta} = 22.95$)
2	500 images-clés du corpus TRECVID 2002 corpora (`trec`)	ID et norme associé ($\lceil \log(A) \rceil = 7$).	NAVGRAPH $c = 6$. ($H = 6$, $\bar{\Delta} = 10.73$)
3	500 images-clés (comme exp. 2)	MPEG7 CLD (12 coefficients) et norme $L2-$ pondéré [BMRS02] ($\lceil \log(A) \rceil = 7$).	NAVGRAPH $c = 6$. ($H = 4$, $\bar{\Delta} = 46.76$)
4	265 images (comme exp. 1).		recherche linéaire.
5	500 images (comme exp. 2).		recherche linéaire.

TABLE 4.1 – Jeux de données (H : médiane du nombre de sauts en moins 120s, $\bar{\Delta}$: charge cognitive moyenne).

pour la recherche linéaire si la cible est trouvée car nous avons limité le temps de recherche à 2 minutes. Le taux de réussite est de l'ordre de 95% pour le graphe de navigation et 85% pour la recherche linéaire. Ceci montre l'intérêt du graphe de navigation par rapport à l'affichage exhaustif de collections de quelques centaines d'images. Nous avons reproduit ces expéricences à des collections de 1000 et 2000 images pour lesquelles l'avantage du graphe de navigation est plus flagrant. Cependant, un passage a une plus grande échelle (plusieurs dizaines de milliers) nécessite une meilleure "qualité" d'indexation car nous sommes de plus en plus confrontés à des problèmes de fossé sémantique ainsi qu'à une limitation de la charge cognitive pour limiter le temps de décision [6] de l'utilisateur.

4.2.5 Un algorithme de réduction de dimension pour l'exploration d'une collection d'images

Après avoir sélectionné les images à afficher, se pose la question de la visualisation : comment disposer les images ? L'objectif de ce travail est de *visualiser les dissimilarités* d'une collection d'images dans le plan afin d'*explorer* la collection. Si les dissimilarités représentent un espace métrique, le principe est donc de plonger l'espace métrique dans le plan avec une *distortion faible* avec une contrainte forte : ce ne sont pas des points que l'on plonge mais des images, c'est-dire des rectangles. Il faut donc aussi tenir compte des problèmes d'occlusion.

6. correspondant au temps de latence en routage

Techniques de réduction de dimension

Le principal objectif est que les similarités et dissimilarités de l'espace métrique initial soient bien reflétées en petite dimension. Ces deux objectifs sont antagonistes : prenons n points pour lesquelles les distances de toute paire sont les mêmes. Essayons de les plonger dans une grille euclidienne de dimension D. La distortion [7] est $\Omega(n^{1/D})$ pour tout plongement. Nous utilisons parfois l'abbréviation MDS ("multivariate dimension scaling") pour parler d'algorithmes de réduction de dimension.

Un algorithme de plongement dans un espace euclidien de dimension D consiste à affecter à chaque point p d'un espace métrique une coordoonnée $\rho(p) \in \mathbb{R}^D$. La distance $d(p,q)$ est la distance euclidienne entre $\rho(p)$ et $\rho(q)$. La distortion d'un plongement peut se mesurer de plusieurs manières :

- *le stress* est une mesure globale : $\sqrt{\sum_p \sum_{q \neq p} \frac{(\delta(p,q)-d(p,q))^2}{\sum_p \sum_{q \neq p} d(p,q)^2}}$;
- *l'étirement* est une mesure individuelle : $\max_{p,q} \frac{\delta(p,q)}{d(p,q)}$.

La présence de *clusters* [8] dans les données et la capacité d'un algorithme de plongement à bien les représenter en $2D$ est un élément important pour l'analyse visuelle de données. Les techniques usuelles ne sont pas dédiées à cette tâche et les espaces métriques sont souvent considérés globalement. L'espoir tacite est que si on plonge un espace métrique en $2D$ avec une faible distorsion, les clusters apparaissent naturellement.

Afin d'effectuer de bonnes comparaisons en tenant compte de cette caractéristique, nous avons eu besoin de jeux de données qui contiennent des clusters. Un corpus bien défini de données *partitionnables* [9] est difficile à trouver. D'autre part, il existe plusieurs manières de définir des clusters.

Les deux principales approches de MDS consistent soit en une analyse des vecteurs propres de la matrice de dissimilarités (c'est l'approche *statistique* de l'analyse en composante principale, l'ACP) où l'usage de modèles physiques : modèles à base de ressorts ou d'attraction/répulsion (voir [FR91, Ead84]. Dans les modèles physiques, le point de départ est un placement aléatoire des points et on laisse agir les forces du système [10] afin de minimiser l'énergie totale du système.

Les algorithmes physiques sont préférés dans le domaine de la visualisation d'information de part leur nature itérative qui permet à l'utilisateur de visualiser le système en évolution et d'intéragir avec lui.

Dans les deux approches, le principal inconvénient est la complexité en temps. Sans hypothèse particulière, la complexité en temps est $\Theta(n^3)$. Pour les modèles physiques, des améliorations peuvent être faites pour réduire le temps à : $O(n^2)$ dans [Cha96], $O(n^{3/2})$ dans [MRC03] et $O(n \log n)$ dans [JM04]) sans dégradation particulière des différentes mesures de distorsion lorsque les points sont distribués aléatoirement dans un espace

7. Dans ce cas, il s'agit du ratio d'aspect de l'espace métrique plongé

8. Terme anglais pour désigner un ensemble de points proches les uns des autres et dissimilaires des autres points

9. avec un découpage naturel des données en clusters.

10. les forces sont calculées de manière à ce que la distance euclidienne entre les points tende vers leur distance initiale dans l'espace d'origine

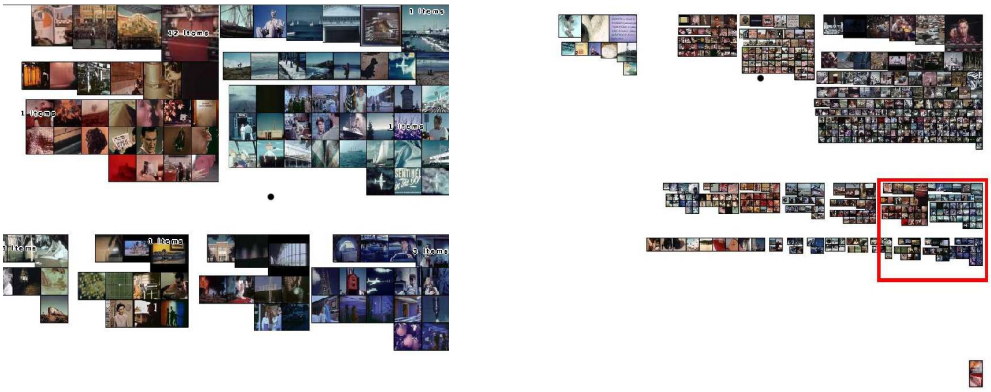
a) b)

FIGURE 4.4 – b) Visualisation de la collection TREC2002CLD.

D-dimensionel. Par contre, les expériences révèlent que la distorsion des algorithmes les plus rapides est bien plus grande dès que les jeux de données sont partitionnables ou que le ratio d'aspect de la métrique est grand. On peut expliquer ce phénomène assez facilement : les différents algorithmes de plongement évoqués sont des heuristiques dans le sens où (1) aucune garantie sur la distorsion n'existe et que (2) le temps d'exécution ne représente qu'un temps de convergence *supposé assez long* pour mener à un état stationnaire. Les algorithmes de plongement les plus rapides sont d'ailleurs basés sur des échantillonnages dans lesquels les petits clusters ont du mal à être pris en considération.

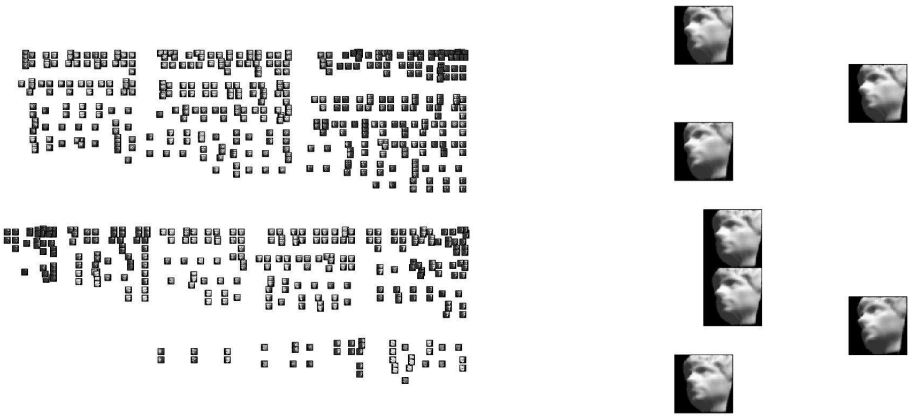
Notre contribution Dans [DH06],nous avons proposé :

- I-Pack (pour Iterative Packing), un algorithme de placement d'images en $2D$ de faible distorsion effective. L'hypothèse de départ est que les dissimilarités entre images peuvent être calculées ou sont données (quelle que soit la forme). La complexité en temps est $O(2^{O(ddim)} n \log A)$. Il est important de signaler que bon nombre de jeux de données réels sont décrits dans des espaces de grande dimension mais possèdent une petite dimension doublante *ddim*. Notre méthode est basée sur deux points :
  1. la construction d'une hiérarchie d'échantillons de plus en plus dense. Dans chaque échantillon, toute paire d'élements est séparée d'une distance minimale dépendant de l'index de l'échantillon ;
  2. le placement en deux dimensions des éléments de l'échantillon, représenté par des rectangles, garantit une séparation minimale entre les rectangles.
- Une méthodologie pour générer des jeux de données aléatoires (cf. Figures 4.7) en fixant simultanément plusieurs paramêtres : cardinalité, dimension doublante, ratio d'aspect, taille des clusters, ...



a) b)

FIGURE 4.5 – b) Visualisation de la collection TREC2002CLD en maximisant les tailles des images. a) Détails de la zone avec des images partageant des composantes de bas niveau.



a) b)

FIGURE 4.6 – Visualisation de la collection faces. a) Vue générale b) Détails d'une zone avec des images partageant un même contenu (pose + illumination).

– Des comparaisons effectives sur des jeux de données réels ou synthétiques (cf. Figure 4.8). Des expériences montrent que le temps d'exécution et les mesures de qualité sont particulièrement intéressantes pour I-Pack dès lors que les données sont partitionnables. Nous avons également proposé une nouvelle mesure de qualité, nommée la *stabilité* du placement afin de mesurer les modifications du placement lors de l'ajout/suppression d'items.
– Nous avons proposé une interface de navigation qui implémente simultanément le placement et l'arbre d'échantillonnage issu de $G_{T,c}$ (cf Figures 4.6 et 4.4).

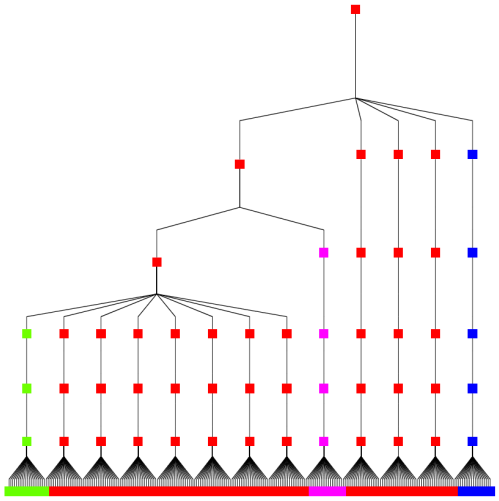


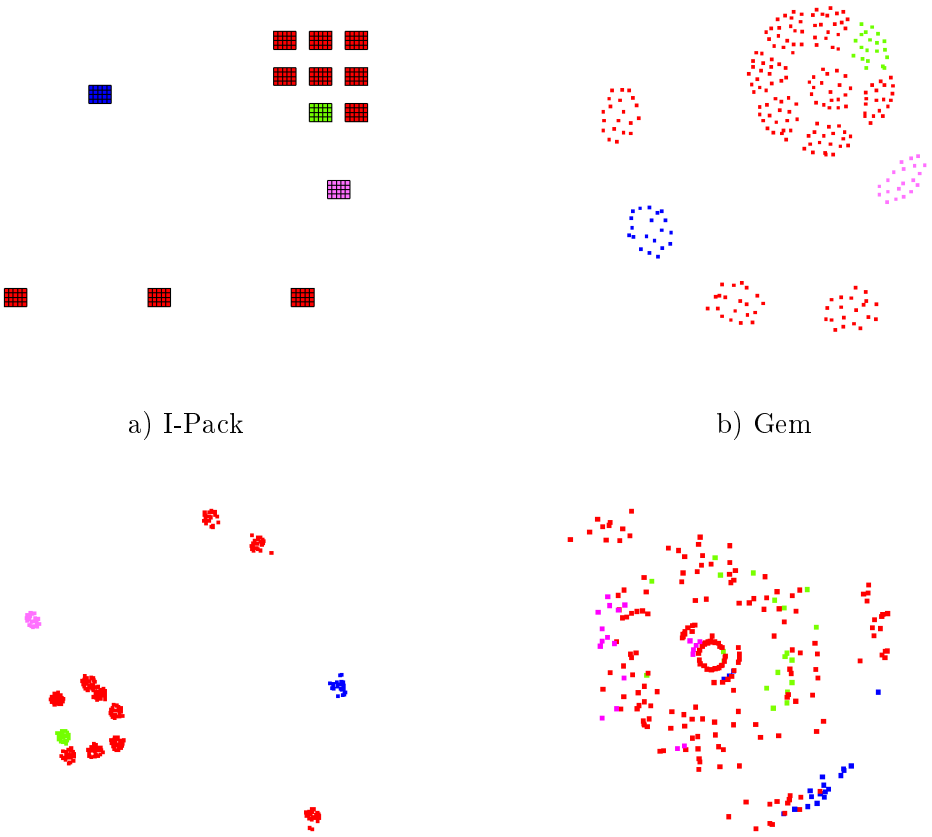
FIGURE 4.7 – Arbre d'échantillon d'un jeu de données partitionnable avec 13 clusters de taille 20

4.2.6 Perspective : vers plus d'intéraction.

En résumé, nos techniques permettent de bien sélectionner et placer les images. Il peut s'avérer impossible d'obtenir un étirement petit pour toutes les paires d'images. Dans ce cas-là, des techniques d'intéractions locales (cf. [MCH+09]) sont particulièrement adaptées pour visionner rapidement le voisinage d'une image dans le graphe de navigation lors d'affichage global d'une collection.

4.3 Vers des requêtes plus complexes ... celles des bases de données

Ce travail peut être considéré comme de la recherche en cours. L'objectif général est de proposer dans un premier temps un éclairage nouveau sur des problèmes classiques de bases de données et de concevoir à plus long terme des algorithmes distribués avec garantie de performances dédiés à la fouille de données. Des travaux préliminaires effectués



a) I-Pack b) Gem

c) Chalmers1 d) Chalmers2

FIGURE 4.8 – Comparaison sur le jeu de données synthétique de la Figure 4.7. Les complexités en temps sont : a) $O(n \log A)$, b) $O(n^3)$, c) $O(n^2)$ et d) $O(n^{3/2})$.

avec Sofian Maabout et Radu Tofan sont présentés dans [HMT09a, HMT09b, HMT09c].

4.3.1 Optimisation de requêtes

Dans les applications OLAP (On-Line Processing Analysis), la rapidité de l'évaluation des requêtes analytiques est primordiale. En effet, les utilisateurs posent des requêtes complexes, difficiles à optimiser par les systèmes de gestion de bases de données classiques. Ainsi, on a souvent recours à un pré-calcul ou d'une manière équivalente à la matérialisation [11] des réponses aux requêtes les plus souvent exprimées par les utilisateurs. Cependant, ces dernières peuvent être trop nombreuses pour être toutes pré-calculées et ceci pour deux raisons : soit à cause du temps qu'il faudrait pour toutes les calculer ou bien en raison de l'espace mémoire nécessaire pour toutes les stocker. Ainsi, on utilise des algorithmes permettant de choisir les *"meilleures"* vues (ou réponses aux requêtes) à matérialiser.

Les données sont représentées par une table T définie sur un ensemble d'attributs D. Une requête sur la table est typiquement formulable grâce à la logique du premier ordre. Dans notre travail, nous ne considérons que les requêtes en langage SQL de la forme : `SELECT c,<Mesures> FROM` T `GROUPBY` c `WHERE <condition>` où $c \subseteq D$ désigne une liste d'attributs. On appele *cuboïde* la sous-table de T définie par un sous-ensemble d'attributs [12]. Le *cube de données* est constitué de l'ensemble des cuboïdes. Que ce soit en présence ou absence de hiérarchies entre certains attributs (ou dimensions), le nombre de cuboïdes existants dans le cube de données $\mathcal{C}$ est supérieur ou égal à 2^D (sans hiérarchie, on suppose ici que $\mathcal{C} = 2^D$).

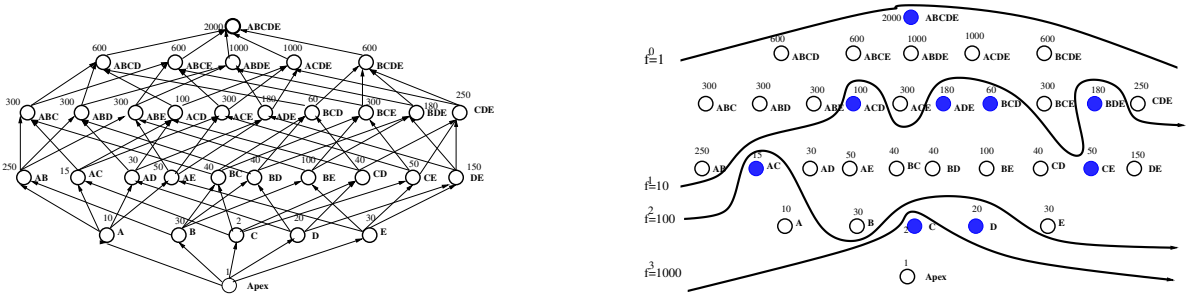
Les performances des algorithmes de sélection sont évaluées selon trois critères :

1. *l'espace mémoire* requis pour stocker les vues sélectionnées ;
2. *le temps d'évaluation* des requêtes ;
3. *le temps d'exécution* de l'algorithme de sélection.

A titre indicatif, sans pré-calcul, c'est à dire sans sélection de vues, le temps d'évaluation d'une requête est proportionnel au nombre d'entrées $|T|$ de la table de faits T. En effet, un parcours complet de T est nécessaire pour répondre. Si on décide de matérialiser à l'avance le cube de données complet, on peut alors accéder directement au cuboïde c correspondant à la requête. Le temps de réponse devient donc optimal [13].

La plupart des solutions proposées jusqu'à présent consistent à choisir un sous-ensemble de vues permettant d'optimiser l'évaluation des requêtes sans dépasser une limite d'espace mémoire fixée par l'utilisateur. Certaines variantes de ce problème ont été étudiées : (1) la nature des données qu'on peut stocker, e.g seulement des vues avec/-sans index (l'index sert à accéder lors de l'existence de la clause de condition), (2) le modèle de coût e.g minimiser non seulement le temps d'évaluation des requêtes mais aussi la coût de mise à jour ou finalement (3) l'ensemble des requêtes susceptibles d'être posées e.g toutes ou bien un sous-ensemble. Dans ce dernier cas, le problème peut être

11. La matérialisation consiste en un stockage sur disque des parties de la base
12. ce qui correspond à la projection de T sur un sous-ensemble d'attributs
13. si on ne considère pas de conditions particulières sur les valeurs d'attributs



a) b)

FIGURE 4.9 – a) Treillis des cuboïdes - la taille des cuboïdes est indiquée ; b) Bordures (en bleu) pour $f = 10^i$.

raffiné en considérant la fréquence $\{p_i\}_{i=1...|\mathcal{C}|}$ ou *charge de travail* des requêtes $\mathcal{Q} \subseteq \mathcal{C}$ que l'on veut optimiser. Nous nous intéressons dans notre étude à un cadre général dans lequel $\mathcal{Q}$ correspond à des *cuboïdes ou requêtes pertinents*. Ils peuvent être définis de manière arbitraire par l'administrateur de la base de données ou sélectionnés parmi les cuboïdes dont la fréquence est au delà d'un seuil fixé, c'est-à-dire les *requêtes fréquentes*.

En général, le problème peut être décrit par un problème d'optimisation sous contrainte : la contrainte étant l'espace disponible qu'il ne faut pas dépasser et la fonction à optimiser est le coût moyen des requêtes qu'il faut minimiser.

Dans [HMT09a, HMT09b], nous proposons que la contrainte soit une contrainte de performance. Dans un monde idéal, le temps de réponse à une requête q est égal à $|q|$, à un facteur de normalisation prés. Pour avoir un tel temps de réponse, il faut stocker toutes les réponses à l'avance. Cette solution est inenvisageable. Si on se permet de dégrader le temps de réponse d'un facteur f avec une petite valeur de $f > 1$, on peut espérer n'avoir à pré-calculer qu'une petite partie du cube de données et ceci en un temps raisonnable. L'objectif dans notre approche consiste donc à minimiser l'espace mémoire requis pour stocker un sous-ensemble de cuboïdes $\mathcal{S} \subset \mathcal{C}$ garantissant un temps de réponse pour une requête $q \in \mathcal{Q}$ d'au plus $f \cdot |q|$.

A notre connaissance, aucune des solutions proposées dans la littérature [GHRU97, TCFS08] n'offre un bon compromis entre complexité de l'algorithme de sélection, minimalité du coût moyen de la solution et garantie de performance.

Algorithmique des treillis. Notre solution est basée sur des algorithmes performants sur les treillis. En effet, on peut naturellement définir un ordre partiel sur les cuboïdes d'une table de faits (cf. Figure 4.9). Informellement, un cuboïde q' est "ancêtre" d'un cuboïde q si toute requête exprimée sur q peut être calculée à partir de q'. C'est le cas si les attributs de q' forme un sur-ensemble des attributs de q. Notre proposition consiste à construire un ensemble de cuboïdes que nous appelons *bordure* $\mathcal{B}$ tel que tout élément de $\mathcal{Q}$ possède un ancêtre dans la bordure qui lui est f-fois plus grand.

Nous avons proposé plusieurs solutions qui sont soit optimales (basées sur une formulation en programmation linéaire en nombre entiers) mais trés coûteuses en temps

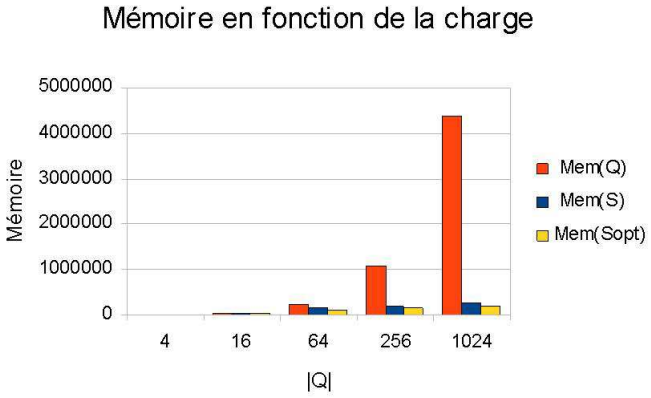


FIGURE 4.10 – Exemple de gain de mémoire avec notre solution approchée (<1 seconde) en comparaison avec la solution optimale (>1 heure avec CPLEX) pour $f = D = 10$.

et relativement peu réalistes (les tailles de cuboïdes doivent être connues) ou des f et $f(1 + \ln|\mathcal{Q}|)$-approximations (basées sur des parcours de graphes et des variantes du problème set cover) de complexité limitée.

4.3.2 Un algorithme parallèle de calcul d'itemsets fréquents

Une variante de notre solution à la matérialisation partielle de vues peut servir à un problème classique de la fouille de données : l'extraction d'itemsets fréquents. Dans [HMT09c], nous revisitons ce problème en proposant un calcul parallèle des itemsets fréquents.

Commençons par définir un peu de terminologie. Soit $\mathbb{I} = \{i_1, \ldots, i_N\}$ un ensemble d'objets (nous parlerons aussi d'items). Soit $\mathcal{T} = \{I_1, \ldots, I_M\}$ un multi-ensemble avec $I_j \subseteq \mathbb{I}$ un ensemble d'objets (nous l'appellerons aussi *itemset*). $\mathcal{T}$ est appelée *base de transactions*. Soit $I \subseteq \mathbb{I}$ un itemset, $|I|_{\mathcal{T}}$ désigne le nombre d'occurrences de I dans $\mathcal{T}$. On notera simplement $|I|$ quand $\mathcal{T}$ est fixé. $t = |\mathcal{T}|$ désigne le nombre d'éléments de $\mathcal{T}$. La *fréquence* $f(I)$ de I dans $\mathcal{T}$ est égale à $\frac{|I|}{|\mathcal{T}|}$. Un item I est *s-fréquent* si $f(I) \geq s$. L'objectif de notre travail est de proposer un calcul distribué des éléments maximaux appelés aussi bordure :

Définition 4.3.1 (Fréquent maximal)**.** *I est un itemset s-fréquent* maximal *si aucun de ses sur-ensembles n'est s-fréquent. La bordure $\mathcal{B}(s)$ contient tous les itemsets s-fréquent maximaux.*

Par exemple, soit $\mathbb{I} = \{A, B, C, D\}$. Soit $\mathcal{T} = \{ABD, AB, ABD, CD, AD\}$. AD est 2/5-fréquent car $f(AD) = 3/5$ alors que CD ne l'est pas puisque $f(CD) = 1/5$. On a que $\mathcal{B}(2/5) = \{ABD\}$.

On rencontre le problème du calcul des itemsets fréquents dans divers domaines : par exemple, on peut être intéressé par les chemins entre routeurs (items) les plus utilisés en

considérant pour tout routage (transaction) la liste des routeurs (itemsets) traversés. De manière générale, l'extraction des itemsets fréquents est extrêmement coûteuse car une bordure peut atteindre 2^N (si $s = 0$ mais pas seulement). Aussi, concevoir un algorithme d'extraction des fréquents nous oblige à tenir compte de deux écueils qui sont le temps pour calculer les fréquences (la table $\mathcal{T}$ peut être parcourue plusieurs fois) ainsi que la quantité de mémoire utilisée à chaque itération.

Les différents algorithmes existants et notre contribution sont résumés dans le tableau 4.2. Les mesures de performance sont :

- la mémoire requise pour chaque processeur (maître et esclaves) ;
- le nombre de rondes exprimant le nombre de parcours de tables par un processeur ;
- le temps $T = T_e + T_m + T_c$ correspondant au : (1) temps T_e de calcul des fréquences fait par les *esclaves* exprimé par le produit le nombre de lignes lues par le coût de calcul par ligne ; (2) temps de calcul du *maître* T_m de composition des résultats partiels en résultat global - principalement calcul de moyennes ; (3) temps de communications T_c exprimé en taille de messages échangés.

Il est à noter que jusqu'a présent, aucun modèle de temps n'avait été proposé dans la littérature. Notre proposition permet de comparer d'un point de vue théorique les différents algorithmes. De ce point de vue, APriori peut sembler aussi compétitif que MAFIA ou MAX-Miner mais ces deux derniers algorithmes s'avèrent bien plus performants en pratique car ils utilisent des heuristiques dont une analyse de pire cas (comme l'analyse présente) ne peut tenir compte.

Algorithme	Mémoire	ronde	T_e	T_m	T_c
APriori [AS94]	$O(2^N/\sqrt{N})$	N	$O(t2^N)$		
MAFIA/GenMAX [BCG01, GZ05]	$O(2^N/\sqrt{N})$	$N/2$	$O(t2^N)$		
PickBorders [HMT09a]	$O(b)$	$O(2^N)$	$O(2^N t)$		
Parallel/MAX-Miner [CL08]	$O(2^N/\sqrt{N})$	$N/2$	$O(\frac{2^N t}{k})$	$O(2^N)$	$O(2^N)$
Notre contribution	$O(b)$	$2N$	$O(\frac{2^N t}{k})$	$O(Nb)$	$O(2^N)$

TABLE 4.2 – Contribution ($b = |\mathcal{B}(s)|$)

Pour indication, les complexités des versions séquentielles sont indiquées en supposant que tout le calcul est fait par un processeur. Les versions parallèles existantes (principalement Parallel MAX-Miner) partent du principe que chaque processeur est responsable d'une proportion $1/k$ de $\mathcal{T}$ et que le maître s'occupe de composer les résultats. Les inconvénients sont multiples : les quantités de calculs effectués et la taille des messages échangés sont indépendantes de la taille de la bordure.

4.3.3 Perspectives en distribué

Vers des algorithmes distribués avec garantie de performances

Notre solution actuelle est essentiellement efficace d'un point de vue de la mémoire requise et du temps de calcul du maître. Il serait bien entendu intéressant de la tester en

pratique dans un environnement de type grille.

Pour l'instant, le temps de calcul est dominé par le nombre d'appels à la fonction "calcul taille d'un cuboïde" (pour la sélection de vues) et "calcul fréquence" (pour les itemsets fréquents). Nous sommes en train d'étudier un algorithme probabiliste qui garantirait l'optimalité du nombre d'appels à un calcul (de l'ordre de $\Theta(bD)$ [GKM+03] basé sur un calcul itératif - et donc non parallélisable - de transversaux de graphes). Ce temps est souvent négligé dans la littérature [14].

Étiquetages Compacts et distribués

Une autre direction est celle du codage compact, distribué, permettant des requêtes rapides. A ce jour, peu de résultats combinant toutes ces caractéristiques existent : si la structure induite est un arbre (XML ou autre), on sait comment coder un arbre de manière distribuée en autorisant des requêtes comme le test d'adjacence ou d'ancêtralité (cf. [AAK+06, FK09]). Malheureusement, les treillis considérés dans notre étude peuvent s'avérer bien plus difficiles à appréhender. Pour les cubes de données, de nombreux algorithmes de compression existent (basés sur les produits de partitions [CNC+09], ...). Mais pour l'instant, le codage compressé de treillis n'a pas été considéré dans le cadre distribué. Toutes ces voies sont prometteuses.

14. En pratique, des estimations grossières sont utilisées sans aucune garantie de précision.

Chapitre 5

Autres contributions à l'algorithmique distribuée

Nous présentons brièvement des contributions à l'algorithmique distribuée qui ont été publiées dans [DHSZ06] (diffusion épidémique) et dans [DHT04, BGH+09] (exploration perpétuelle).

5.1 Diffusion d'information épidémique

Les articles qui traitent de la diffusion ont pour objectif de limiter l'usage de ressources mémoire, le nombre de messages circulant dans le réseau et de borner le temps au bout duquel chaque noeud est informé. La variété de modèles et de contraintes a déja donné lieu à une littérature luxuriante mais la majorité des algorithmes possèdent de nombreux inconvénients qui empêchent leur usage dans de réelles applications : ils sont rarement tolérants aux pannes ! En effet, des chemins de diffusion sont souvent précalculés. Si un noeud intermédiaire tombe en panne, le calcul d'un chemin de secours n'est pas forcément prévu. Ils sont gourmands en mémoire et nécessitent une connaissance préalable et globale du réseau. Ils sont inadaptés aux réseaux dynamiques (par exemple les réseaux mobiles ad-hoc).

Une famille d'algorithmes pour la diffusion semble l'emporter sur les autres en terme d'efficacité et de mise-en-place. Il s'agit des algorithmes de type "commérage" ou de "contamination". Ce sont des algorithmes probabilistes. Ils fonctionnent avec le principe suivant : un noeud informé choisit de manière aléatoire un noeud voisin en fonction d'informations locales et lui envoie le message à diffuser. L'intérêt de ce processus est qu'il n'est pas nécessaire d'avoir des informations globales (topologie inconnue) sur le réseau pour diffuser (donc léger en mémoire), et qu'il supporte dans de nombreux cas les pannes et l'asynchronisme. Cependant, les analyses et comparaisons du temps de diffusion sont exprimées lors d'executions synchrones.

En poussant les rumeurs Une première étude a été menée par Feige et al. [FPRU90] en 1990. Les auteurs donnent des bornes supérieures et inférieures générales valides avec

grande probabilité, $\Omega(\log(n))$ et $O(n\log(n))$, pour le temps de diffusion d'un message dans un graphe inconnu, n étant le nombre de noeuds. Un noeud contaminé choisit uniformément au hasard un voisin et le contamine.

En poussant et tirant les rumeurs Un travail récent de Karp et al. [KSSV00] étudie le modèle de l'appel téléphonique aléatoire. A chaque ronde, chaque noeud choisit uniformément au hasard un voisin (plus ou moins comme avec Feige), la transmission de la rumeur a alors lieu soit de l'appelant vers l'appelé - push transmission algorithm - soit de l'appelé vers l'appelant - pull transmission algorithm -. Dans le cas du graphe complet, ils ont montré que toute rumeur diffusée en $O(\log(n))$ étapes nécessite en moyenne l'envoi d'au moins $\omega(n)$ messages. L'introduction de la notion d'âge d'une rumeur permet d'arrêter la diffusion de la rumeur complètement diffusée (ce qui n'est pas considéré par Feige).

5.1.1 Diffusion épidémique dans le modèle rendez-vous

Dans notre étude, nous proposons un nouveau modèle de communication basé sur les rendez-vous et nous analysons un algorithme multi-saut distribué pour diffuser un message dans un environnement synchrone. Dans le modèle "rendez-vous", deux voisins peuvent communiquer si et seulement s'ils s'appellent mutuellement et simultanément. Le modèle "rendez-vous" - introduit en 2000 par Métivier, Saheb et Zemmari[MSZ00] - dans le cadre de la diffusion est totalement nouveau.

Pourquoi prendre rendez-vous ? Le mécanisme de rendez-vous permet le contrôle de congestion ou de collision afin de limiter l'échange d'informations avec au plus un noeud voisin à la fois. Par exemple, le problème de la maintenance économe des bases de données répliquées a été proposé par Demers *et al.* [DGH$^+$88]. La réplication est le processus de copie et de maintenance d'objets dans un système distribué. Les changements appliqués à un site sont capturés localement avant d'être transmis et appliqués aux autres sites. Reguliérement, chaque base de données essaye de comparer son contenu avec d'autres bases de données. A cause de la taille des bases de données, la quantité d'informations transmises et le temps requis pour la mise-à-jour peuvent être énorme. Nous pouvons être intéressés par interdire qu'une base de donnée échange des données simultanément avec plusieurs bases à cause de la taille des messages échangés. Les rendez-vous garantissent cette propriété et la taille de transmission pour obtenir un rendez-vous peut être trés petite (seulement l'identité d'une base, ...). Plus généralement, le processus de rendez-vous est utile dès que l'on veut minimiser la congestion, la quantité d'information pour des tâches de mise-à-jour dans des réseaux dynamiques : tables de routage dans les réseaux mobiles, réseau de robots (à cause de contraintes physiques) ... Dans [MSZ02], les rendez-vous sont utilisés pour élire de manière aléatoire un noeud dans un graphe anonyme.

Le modèle de rendez-vous. Le graphe sous-jacent est considéré comme non-orienté. Un sommet est *contaminé* s'il a reçu le message envoyé par le *sommet initial* v_0.

Le modèle peut être implémenté de manière totalement distribuée. L'analyse de la complexité est fondée sur la notion de *rondes*. Plus précisément, à chaque ronde :

- pour chaque sommet $v \in V$, choisir en parallèle aléatoirement et uniformément une arête incidente,
- si une arête (v_i, v_j) a été choisie par v_i et v_j, il y a rendez-vous,
- s'il y a un rendez-vous et si v_i est le seul contaminé, v_j devient contaminé.

T_G est le *temps de diffusion* ou le *temps de contamination* ; c'est le nombre de rondes nécessaires pour que tous les sommets du graphe G soient contaminés. T_G est une variable aléatoire ; dans notre étude, on s'intéresse à son espérance mathématique $\mathbf{E}(T_G)$. Il est évident que la diffusion par rendez-vous est plus lente qu'un processus de diffusion en poussant et en tirant puisqu'il nécessite une synchronisation locale. Nous donnons des bornes assez précises sur $\mathbf{E}(T_G)$.

On peut formuler quelques remarques sur notre modèle : notre algorithme est sans mémoire puisqu'on ne prend pas en considération le cas où un voisin v_j d'un sommet contaminé v_i a déjà été contaminé. Plusieurs diffusions peuvent avoir lieu simultanément et nous étudions le temps de diffusion d'une seule rumeur. Dans notre étude, les ennemis sont le degré et le diamètre. Considérons deux étoiles à $n/2$ feuilles, relions les centres de ces deux étoiles par une arête. Dans notre modèle, l'espérance mathématique du temps de diffusion est minorée par $\Omega(n^2)$ alors que dans le modèle de [FPRU90], elle est en $O(n)$ et avec grande probabilité T_G est en $O(n \log n)$.

Nos résultats Le résultat principal de notre étude est l'encadrement suivant pour tout graphe G : $\log_2 n \leq \mathbf{E}(T_G) \leq O(n^2)$. Plus précisément, pour tout graphe G de degré maximum Δ, $\mathbf{E}(T_G) = O(\Delta n)$.

Nous montrons aussi qu'il existe des graphes pour lesquels l'espérance du temps de diffusion atteint soit la borne inférieure soit la borne supérieure à un facteur constant prés. Par exemple, pour les arbres binaires complets, $\mathbf{E}(T_G) = O(\log_2 n)$ alors que $\mathbf{E}(T_G) = \Omega(n^2)$ pour les graphes double étoile. Pour les graphes à degré borné Δ et de diamètre D, nous montrons aussi que $\mathbf{E}(T_G) = O(D\Delta^2 \ln \Delta)$. Cette borne est serrée car pour un arbre Δ-aire complet de diamètre D, $\mathbf{E}(T_G) = \Omega(D\Delta^2 \ln \Delta)$. Dans [MSZ00], il a été prouvé que le graphe complet minimise le nombre moyen de rendez-vous, cependant, le temps moyen de diffusion est $\Theta(n \ln n)$.

Les techniques principales sont basées sur l'étude des arêtes de la coupe entre sommets contaminés et les non contaminés. La principale difficulté est d'arriver à faire une analyse de complexité amortie pour aboutir à des résultats fins.

5.1.2 Perspective : Diffusion dans les graphes aléatoires k-out

Pour $k \geq 2$, l'étude des graphes aléatoires k-out [1] pour une diffusion efficace n'a pas été publiée à ce jour. C'est un travail fait avec Philippe Duchon. Les résultats sont juste présentés sous forme de perspectives.

Pour un usage de diffusion économique dans un réseau pair-à-pair, le graphe k-out est un bon candidat : le carnet d'adresses d'un réseau pair-à-pair est de petite taille ($2k$ en moyenne), une requête de choix aléatoire de noeuds dans un réseau pair-à-pair est une requête envisageable (même si on ne sait pas faire aléatoire et uniforme), ... A notre connaissance, le travail de diffusion épidémique et économique est celui de Karp [KSSV00]. Cependant pour obtenir une diffusion totale en $O(\log n)$ rondes, non seulement leur modèle nécessite $\omega(N)$ messages mais chaque message est horodaté.

En utilisant un k-out aléatoire comme graphe de diffusion, nous montrons que :

- Pour $k \geq 2$, avec grande probabilité, un graphe aléatoire non orienté k-out est connexe et de diamêtre logarithmique. Il contient une grande composante fortement connexe de taille au moins $N/2$. Tout noeud est à distance au plus $O(\log \log N)$ de cette composante.
- Pour $k \geq 2$, toute rumeur peut être propagée en utilisant l'algorithme "en poussant" dans un graphe k-out aléatoire en $O(\log N)$ rondes, avec $O(N)$ messages.

La meilleure performance des graphes k-out provient de la trés faible redondance, de l'ordre de k, des messages reçus par chaque noeud.

5.2 Exploration perpétuelle

L'exploration perpétuelle de graphes et de réseaux consiste à ce qu'un jeton ou message parcourt indéfiniment à l'aide d'un même algorithme distribué stocké dans chaque noeud. Les applications sont multiples :

- *exclusion mutuelle* : On cherche à garantir que l'accès à une ressource du réseau soit exclusif et limité à un noeud. Le principe consiste à faire circuler un jeton dans le réseau et seul le possesseur du jeton a accés à la ressource critique.
- *Mise-à-jour perpétuelle* : des informations stockées à des noeuds peuvent être mises à-jour, des alertes lancées. C'est une forme de diffusion lente.

Le véritable intérêt de nos résultats est qu'ils traitent d'*algorithmes auto-stabilisants*, c'est-à-dire d'algorithmes tolérants des *pannes transitoires*. Dans notre contexte, les pannes ont des significations multiples : corruption de la mémoire d'un noeud mais aussi changement de topologie du graphe sous-jacent (arrivée ou départ de noeuds, ajout ou suppression de liens).

1. Dans un graphe k-out, chaque sommet choisit de manière aléatoire uniforme k sommets auxquels il se relie. Ensuite, on peut désorienter les arêtes. Le degré d'un sommet n'est donc pas nécessairement égal à $2k$.

5.2.1 Algorithmes d'exclusion mutuelle probabilistes auto-stabilisants

Le problème de l'exclusion mutuelle est un problème fondamental dans le domaine de l'informatique répartie. Considérons un système réparti de n processeurs. Tous les processeurs, de temps à autre, peuvent avoir à exécuter une section critique de leur code durant laquelle exactement un processeur est autorisé à utiliser une ressource partagée. Un système réparti qui résoud le problème de l'exclusion mutuelle doit garantir les deux propriétés suivantes : *(i) exclusion mutuelle* : exactement un processeur est autorisé à exécuter sa section critique à un instant donné ; *(ii) équité* : tout processeur doit être en mesure d'exécuter sa section critique infiniment souvent au cours de l'exécution.

Auto-stabilisation. Le concept de l'auto-stabilisation a été introduit pour la première fois par Dijkstra en 1974 [Dij74]. Ce concept est maintenant considéré comme la technique la plus générale pour concevoir des systèmes qui tolèrent des défaillances transitoires arbitraires. Un système auto-stabilisant garantit que, en partant d'un état initial arbitraire, le système converge vers un état correct en un nombre fini d'étapes, et reste correct en l'absence de nouvelles défaillances (cf. [Dol00]).

Intuitivement, un protocole d'exclusion mutuelle auto-stabilisant par passage de jeton garantit que, même si on part d'un état ou la spécification de l'exclusion mutuelle n'est pas respectée (zéro ou plusieurs jetons sont présents dans le système), alors en un nombre fini d'étapes, un seul jeton circule équitablement dans le réseau. En pratique, on se borne à prouver qu'à partir d'une configuration à plusieurs jetons, on aboutit en un temps fini à une configuration à jeton unique. En effet, il a été prouvé dans [KP93] dans le cas où on se trouve dans un système réparti où les processeurs communiquent par passage de messages, il est indispensable de disposer d'un mécanisme de temporisation (*timeout*) pour injecter des jetons spontanément : si un tel mécanisme n'est pas disponible, le système ne peut être auto-stabilisant puisqu'il se retrouverait bloqué en démarrant d'une configuration initiale sans message.

Travaux précédents

Un peu de terminologie est nécessaire. Un réseau est *uniforme* si tout processeur exécute le même code, et il est *anonyme* si les processeurs ne disposent par d'identificateurs pour exécuter des sections de code différentes. Si un protocole fonctionne dans un réseau uniforme et anonyme, alors il fonctionne *a fortiori* dans un réseau non uniforme ou non anonyme. Depuis les trois protocoles d'exclusion mutuelle auto-stabiliante par passage de jeton proposés dans l'article fondateur [Dij74], de nombreux travaux ont traité de ce problème dans différents contextes et par exemple [Her90, BCD, BGJ, DGT, Ros, Joh] dans le cas des anneaux de processeurs unidirectionnels anonymes et uniformes.

Un protocole d'exclusion mutuelle par passage de jeton est *transparent* vis à vis de l'application qui utilise le protocole s'il ne modifie pas le format des jetons qui sont échangés par l'application. Une telle propriété est souhaitable si par exemple le contenu du jeton est utilisé par l'application (c'est le cas dans un réseau de type *Token Ring* ou *FDDI*, où le jeton contient également les informations devant être transmises au destinataire).

En effet, un protocole transparent est plus facile à implanter (il ne modifie pas le format des trames de l'application) et plus facile à intégrer à des réseaux hétérogènes (dont certaines parties utilisent un protocole de passage de jeton différent). En outre, la charge de la vérification de l'intégrité des messages peut être déléguée entièrement à l'application. Parmi les protocoles précités, seuls [Her90, BCD] et le protocole synchrone de [DGT] sont transparents vis à vis de l'application qui utilise le protocole. Les protocoles présentés dans [BGJ, DGT, Ros, Joh] utilisent soit plusieurs types de jetons (et donc ajoutent un champ *type* aux messages de l'application), soit des informations supplémentaires à chaque jeton circulant de manière à assurer la stabilisation.

Un critère important pour évaluer l'efficacité de la stabilisation d'un protocole est de calculer son *temps de stabilisation* ou *temps de convergence*, noté T, c'est-à-dire le temps qu'il faut pour passer d'une configuration contenant un nombre arbitraire de jetons à une configuration à jeton unique. Comme montré dans [BP89], il est impossible de résoudre le problème de l'exclusion mutuelle auto-stabilisante dans un anneau unidirectionnel anonyme et uniforme au moyen d'un protocole déterministe. Aussi les solutions précédentes sont-elles toutes probabilistes. Parmi celles-ci, [Her90, BCD] ne proposent pas de calcul du temps de stabilisation, et les temps de stabilisation moyens attendus de [BGJ, DGT, Joh] sont de l'ordre de $O(n^3)$, où n désigne la taille de l'anneau, et celui de [Ros] est de l'ordre de n^2. Il est évident que le temps de stabilisation est en $\Omega(n)$, puisque si on part d'une configuration à deux jetons diamêtralement opposés, il faut au moins $n/2$ unités de temps pour qu'un jeton rattrape l'autre.

Un autre critère d'évaluation est celui du *temps de service*, c'est à dire le temps, en phase stabilisée, entre deux passages du jeton sur un processeur. Ce temps est important pour évaluer les performances du protocole quand il n'y a pas de défaillances et ainsi évaluer son surcoût par rapport à un protocole non stabilisant. Dans un système à n processeurs, le temps de service est en $\Omega(n)$, puisque si chaque processeur attend le minimum de temps, il attend autant que les autres. Le temps de service n'est pas calculé dans [Her90, BCD, BGJ], et il est dans [DGT, Ros, Joh] de l'ordre de n^3, n^2, et n, respectivement.

Notre contribution

Dans [DHT04], nous proposons plusieurs protocoles auto-stabilisants pour les réseaux synchrones anonymes et uniformes en anneau unidirectionel où les processeurs communiquent par échange de messages.

Analyse fine de protocoles existants. Les deux premiers protocoles sont des transpositions dans un modèle à passage de messages des protocoles de [Her90, BCD] (pour un protocole sans mémoire et sans connaissance de la taille du réseau) et de [DGT] (pour un protocole avec 1 bit de mémoire et sans connaissance de la taille du réseau)) qui utilisent un modèle à mémoire partagée. Par une analyse fine de complexité, nous montrons que l'équivalent de [Her90, BCD] converge en temps $\Theta(n^2)$ en moyenne, et que son temps de service est en $\Theta(n^2)$ en moyenne (ces deux complexités n'étaient pas calculées dans les articles cités). Puis nous montrons que l'équivalent du protocole synchrone de [DGT]

converge en temps moyen $\Theta(n^2)$ et que son temps de service est en $\Theta(n)$ (il était prouvé seulement $O(n^2)$ dans l'article original).

Nouveaux protocoles. Nous avons également proposé plusieurs protocoles originaux basés sur la notion de *ralentisseur*. A tout instant, chaque processeur possède un état : normal ou ralentisseur. La processus de changement d'état est probabiliste. Chaque processeur peut se proclamer ralentisseur et ralentir les jetons qu'il reçoit avec une certaine probabilité, c'est-à-dire empêcher l'envoi de jetons tant qu'il reste dans l'état ralentisseur. Suivant la puissance du ralentisseur considéré (quantité de mémoire disponible), les résultats de complexité obtenus sont différents, mais chacun des protocoles que nous proposons a un temps de convergence moyen et un temps de service moyen en $O(n)$ (de manière certaine pour un des protocoles). Le dernier protocole du tableau 5.1 est optimal au regard de toutes les mesures de performance considérées.

Protocole	Connaissance de n	Temps de stabilisation	Temps de service	Mémoire	Uniforme	Transparent
[BGJ]	oui	$\Theta(n^3)$	$O(n^3)$	$O(\log(n))$	oui	non
[Ros]	non	$\Theta(n^2)$	$O(n^2)$	$O(1)$	oui	non
[Joh]	oui	$O(n^3)$	$O(n)$	$O(\log(n))$	oui	non
SansMémoire	non	$\Theta(n^2)$	$\Theta(n)$	0	oui	oui
Mémoire1bit	non	$\Theta(n^2)$	$\Theta(n)$	$O(1)$	oui	oui
Ralentisseur1	oui	$\Theta(n)$	$\Theta(n)$	$O(\log(n))$	oui	oui
Ralentisseur2	oui	$\Theta(n)$	$\Theta(n)$	$O(1)$	oui	oui

FIGURE 5.1 – Récapitulatif des résultats

Chaînes de Markov.

Les techniques de nos preuves sont basés sur une analogie entre le temps d'atteinte d'un sommet dans un chemin lors d'une marche aléatoire et le temps de stabilisation : les sommets du chemin sont numérotés de 0 à $n/2$ et représentent la distance entre deux jetons consécutifs. Le temps d'atteinte est le temps au bout duquel la distance devient nulle, *i.e.* les deux jetons fusionnent, lorsque les jetons suivent des marches aléatoires avec ou sans mémoire. L'astuce consiste à considérer les deux jetons les plus proches mais la difficulté est que ce n'est pas toujours la même paire de jetons qui se retrouve être les plus proches. Pour remédier à ce problème, notre analyse est faite sur un processus qui est garantit être plus lent que le processus réel, extrêmement difficile à étudier.

5.2.2 Exploration eulérienne

Le modèle *rotor-router*, aussi appelé *Machine de Propp*, a été considéré comme une alternative déterministe à la marche aléatoire. Il est connu que le chemin suivi par un agent mobile dans un graphe non-orienté contrôlé par un mécanisme de rotor-router finit

par former un cycle eulérien [2] en remplaçant chaque arête par deux arcs orientés dans les deux directions opposées. Le déplacement de l'agent mobile/message est régi par le processus suivant :

- Pour tout noeud, un ordre cyclique des arcs sortant est fixé et le noeud stocke le dernier arc sortant utilisé par l'agent ;
- Tout agent mobile arrivant à un noeud est redirigé vers l'arc sortant suivant dans l'ordre cyclique.*

Le *temps de convergence*, exprimé en nombre de traversées de sommets, vers un parcours eulérien dépend de l'état initial du système. Il est important de noter que le mécanisme rotor-router est *auto-stabilisant* : si le réseau change de topologie ou l'ordre cyclique est modifié, on finira à nouveau par trouver un cycle eulérien.

Dans un travail récent [YWB03], Yanovski *et al.* ont montré qu'indépendamment de la configuration initiale du système, en utilisant un mécanisme de rotor-router, un agent mobile finit par suivre un cycle eulérien en temps inférieur à $2mD$ où D est le diamètre du graphe. Cette borne supérieure est atteinte par le graphe "sucette" [3].

Résultat Dans [BGH$^+$09], nous examinons la dépendance du temps de convergence vers un cycle eulérien avec l'état initial du système. Nous développons une étude de cas sous la forme d'un jeu entre un *joueur* dont le but est d'obtenir un temps de convergence minimal et un *adversaire* avec l'objectif inverse. Le joueur ou l'adversaire a le choix de l'initialisation des ordres cycliques et/ou des premiers arcs. L'ordre du jeu, c'est-à-dire si le joueur joue avant ou après l'adversaire, a des conséquences. Par exemple, il existe des cas pour lesquels le temps de convergence est $O(m)$. D'un autre coté, si l'adversaire a la possibilité de fixer initialement l'ordre cyclique des arcs et des permiers arcs sortants, le temps de convergence atteint $\Omega(mD)$ *pour tout* graphe de m arêtes et diamêtre D. De plus, nous montrons que si l'adversaire fixe l'ordre cyclique après que le joueur ait fixé les premiers arcs sortants alors le temps de convergence est majoré par $O(m \min\{\log m, D\})$. Dans ce cas, nous proposons également une classe de graphes pour lesquels le temps de convergence nécessite $\Omega(m \log m)$. Dans les deux derniers cas, nous exhibons des graphes nécessitant $\Omega(mD)$ et des familles non triviales de graphes de grand diamètre nécessitant un temps $O(m)$.

2. Pour rappel, un cycle est *eulérien* s'il passe par toutes les arêtes - pour nous arcs - du graphe une et une seule fois.

3. Ce graphe est constitué d'une clique et d'un long chemin relié à un sommet de la clique

Bibliographie

[AAK+06] Serge Abiteboul, Stephen Alstrup, Haim Kaplan, Tova Milo, and Theis Rauhe. Compact labeling schemes for ancestor queries. *SIAM Journal on Computing*, 35(6) :1295–1309, 2006.

[ACL00] William Aillo, Fan R. K. Chung, and Linyuan Lu. A random graph model for massive graphs. In 32^{nd} *Annual ACM Symposium on Theory of Computing (STOC)*, pages 171–180. ACM Press, May 2000.

[ADS06] Luca Castelli Aleardi, Olivier Devillers, and Gilles Schaeffer. Optimal succinct representations of planar maps. In *SCG '06 : Proceedings of the twenty-second annual symposium on Computational geometry*, pages 309–318, New York, NY, USA, 2006. ACM.

[AG06] Ittai Abraham and Cyril Gavoille. Object location using path separators. In 25^{th} *Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 188–197. ACM Press, July 2006.

[AGGM06] Ittai Abraham, Cyril Gavoille, Andrew V. Goldberg, and Dahlia Malkhi. Routing in networks with low doubling dimension. In 26^{th} *International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society Press, July 2006.

[AKP+04] P. Androutsos, A. Kushki, K.N. Plataniotis, D. Androutsos, and A.N. Venetsanopoulos. Distributed MPEG-7 image indexing using small world user agents. In *Proc. of ICIP 2004*, pages 641–644, 2004.

[ARS97] Laurent Alonso, Jean-Luc Rémy, and René Schott. A linear-time algorithm for the generation of trees. *Algorithmica*, 17(2) :162–182, 1997.

[AS80] D. B. Arnold and M. R. Sleep. Uniform random generation of balanced parenthesis strings. *ACM Transactions on Programming Languages and Systems*, 2(1) :122–128, 1980.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of VLDB'94 conference*, 1994.

[BA99] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439) :509–512, 1999.

[BCD] J. Beauquier, S. Cordier, and S. Delaët. Optimum probabilistic self-stabilization on uniform rings.

[BCG01] Doug Burdick, Manuel Calimlim, and Johannes Gehrke. Mafia : A maximal frequent itemset algorithm for transactional databases. In *Proceedings of ICDE'01 conference*, 2001.

[BdLP99] Elena Barcucci, Alberto del Lungo, and Elisa Pergola. Random generation of trees and other combinatorial objects. *Theoretical Computer Science*, 218(2) :219–232, 1999.

[BFSS00] Cyril Banderier, Philippe Flajolet, Gilles Schaeffer, and Michèle Soria. Planar maps and airy phenomena. In 27^{th} *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1853 of Lecture Notes in Computer Science, pages 388–402. Springer, July 2000.

[BGH03a] Nicolas Bonichon, Cyril Gavoille, and Nicolas Hanusse. Canonical decomposition of outerplanar maps and application to enumeration, coding and generation. Research Report RR-1295-03, LaBRI, University of Bordeaux 1, 351, cours de la Libération, 33405 Talence Cedex, France, March 2003.

[BGH03b] Nicolas Bonichon, Cyril Gavoille, and Nicolas Hanusse. An information-theoretic upper bound of planar graphs using triangulation. In 20^{th} *Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2607 of Lecture Notes in Computer Science, pages 499–510. Springer, February 2003.

[BGH+06] Nicolas Bonichon, Cyril Gavoille, Nicolas Hanusse, Dominique Poulalhon, and Gilles Schaeffer. Planar graphs, via well-orderly maps and trees. *Graphs and Combinatorics*, 22(2) :185–202, 2006.

[BGH+09] Evangelos Bampas, Leszek Gasieniec, Nicolas Hanusse, David Ilcinkas, Ralf Klasing, and Adrian Kosowski. Euler Tour Lock-in Problem in the Rotor-Router Model. In *DISC 2009*, volume 5805, pages 421–433, Espagne, 09 2009. Springer Berlin / Heidelberg.

[BGJ] J. Beauquier, M. Gradinariu, and C. Johnen. Memory space requirements for self-stabilizing leader election protocols.

[Bil92] Tomasz Bilski. Embedding graphs in books : A survey. *IEE Proceedings-E*, 139(2) :134–138, March 1992.

[BMRS02] A.B. Benitez, M.J. Martinez, H. Rising, and P. Salembier. Color descriptors. In *Introduction to MPEG-7 : Multimedia Content Description Language*, chapter 13, pages 187–212. Wiley, 2002.

[BO04] Pierce G. Buckley and Deryk Osthus. Popularity based random graph models leading to a scale-free degree sequence. *Discrete Mathematics*, 282(1-3) :53–68, 2004.

[Bon02] Nicolas Bonichon. A bijection between realizers of maximal plane graphs and pairs of non-crossing Dyck paths. In *Formal Power Series & Algebraic Combinatorics (FPSAC)*, July 2002.

[BP89] Jean-Claude Bermond and Claudine Peyrat. de Bruijn and Kautz networks : a competitor for the hypercube ? In F. André and J-P. Verjus, editors, 1^{st} *European Workshop on Hypercube and Distributed Computers*, pages 279–293. North Holland, 1989.

[BR03] B. Bollobás and O. Riordan. Mathematical results on scale-free random graphs. In S. Bornholdt and H.Schuster, editors, *Handbook of graphs and networks*, pages 1–34. Wiley-VCH, Berlin, 2003.

[BR04] Béla Bollobas and Oliver Riordan. The diameter of a scale-free random graph. *Combinatorica*, 24(1) :5–34, 2004.

[BRST01] Béla Bollobás, Oliver Riordan, Joel Spencer, and Gábor Tusnády. The degree sequence of a scale-free random graph process. *Random Structures and Algorithms*, 18(3) :279–290, 2001.

[CAFL08] Luca Castelli Aleardi, Éric Fusy, and Thomas Lewiner. Schnyder woods for higher genus triangulated surfaces. In 24^{st} *Annual ACM Symposium on Computational Geometry (SoCG)*, pages 311–319, June 2008.

[CF03] Colin Cooper and Alan M. Frieze. A general model of web graphs. *Random Struct. Algorithms*, 22(3) :311–335, 2003.

[CGH+98] Richie Chih-Nan Chuang, Ashim Garg, Xin He, Ming-Yang Kao, and Hsueh-I Lu. Compact encodings of planar graphs via canonical orderings and multiple parentheses. In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, 25^{th} *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1443 of Lecture Notes in Computer Science, pages 118–129. Springer, July 1998.

[Cha96] Matthew Chalmers. A linear iteration time layout algorithm for visualising high-dimensional data. In *IEEE Symposium on Information Visualization*, pages 127–132, 1996.

[CL02] Fan Chung and Linyuan Lu. Connected components in random graphs with given degree sequences. *Annals of Combinatorics*, 6 :125–145, 2002.

[CL03] Fan Chung and Linyuan Lu. The average distance in random graphs with given expected degrees. *Internet Mathematics*, 1(1) :91–114, 2003.

[CL06] Fan Chung and Linyuan Lu. The volume of the giant component of a random graph with given expected degrees. *SIAM J. Discret. Math.*, 20(2) :395–411, 2006.

[CL08] Soon M. Chung and Congnan Luo. Efficient mining of maximal frequent itemsets from databases on a cluster of workstations. *Knowledge Inf. Systems*, 16 :359–391, 2008.

[CLL01] Yi-Ting Chiang, Ching-Chi Lin, and Hsueh-I Lu. Orderly spanning trees with applications to graph encoding and graph drawing. In 12^{th} *Symposium on Discrete Algorithms (SODA)*, pages 506–515. ACM-SIAM, January 2001.

[CNC+09] Alain Casali, Sébastien Nedjar, Rosine Cicchetti, Lotfi Lakhal, and Noel Novelli. Lossless reduction of datacubes using partitions. *IJDWM*, 5(1) :18–35, 2009.

[DEH07] Philippe Duchon, Nicole Eggemann, and Nicolas Hanusse. Non-searchability of random power-law graphs. In *11th International Conference On Principles Of Distributed Systems (OPODIS 2007)*, volume 4878, pages 274–285, France, 2007. Eduardo Tovar, Philippas Tsigas, Hacene Fouchal.

[DEM01] Eleni Drinea, Mihaela Enachescu, and Michael Mitzenmacher. Variations on random graph models for the web. Technical report, Technical Report, Harvard University, Department of Computer Science, 2001.

[DGH+88] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. *Oper. Syst. Rev., vol.*, 22 :8–32, 1988.

[DGT] A. K. Datta, M. Gradinariu, and S. Tixeuil. Self-stabilizing mutual exclusion using unfair distributed scheduler.

[DH06] Anthony Don and Nicolas Hanusse. A deterministic multidimensional scaling algorithm for data visualisation. In *IV*, pages 511–520. IEEE Computer Society, 2006.

[DH08] A. Don and Nicolas Hanusse. Content-based image retrieval using greedy routing. In *Electronic Imaging - Multimedia Content Access Track*, volume 6820, pages 68200I1–6820I11, États-Unis d'Amérique, 2008. Theo Gevers, Ramesh C. Jain and Simone Santini.

[DHLS05] Philippe Duchon, Nicolas Hanusse, Emmanuelle Lebhar, and Nicolas Schabanel. Brief announcement : Could any graph be turned into a small-world ? In 19^{th} *International Symposium on Distributed Computing (DISC)*, volume 3724 of Lecture Notes in Computer Science, pages 511–513. Springer, September 2005.

[DHLS06] Philippe Duchon, Nicolas Hanusse, Emmanuelle Lebhar, and Nicolas Schabanel. Could any graph be turned into a small-world ? *Theoretical Computer Science*, 355(1) :96–103, April 2006.

[DHSZ06] Philippe Duchon, Nicolas Hanusse, Nasser Saheb, and Akka Zemmari. Broadcast in the rendezvous model. *Inf. Comput.*, 204(5) :697–712, 2006.

[DHT04] Philippe Duchon, Nicolas Hanusse, and Sébastien Tixeuil. Optimal randomized self-stabilizing mutual exclusion on synchronous rings. In Rachid Guerraoui, editor, *DISC*, volume 3274 of *Lecture Notes in Computer Science*, pages 216–229. Springer, 2004.

[Dij74] E. W. Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11) :643–644, 1974.

[Dol00] S. Dolev. *Self-stabilization.* The MIT Press, 2000.

[Don06] Anthony Don. *Indexation et navigation dans les contenus visuels : approches basées sur les graphes.* PhD thesis, Université Bordeaux 1, 2006.

[DVW96] Alain Denise, Marcio Vasconcellos, and Dominic J. A. Welsh. The random planar graph. *Congressus Numerantium*, 113 :61–79, 1996.

[DW04] Vida Dujmović and David R. Wood. On linear layouts of graphs. *Discrete Mathematics & Theoretical Computer Science*, 6(2) :339–358, 2004.

[Ead84] Peter Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42 :149–160, 1984.

[ES94] Peter Epstein and Jörg-Rüdiger Sack. Generating triangulations at random. *ACM Transactions on Modeling and Computer Simulation*, 4(3) :267–278, 1994.

[FG09] Pierre Fraigniaud and George Giakkoupis. The effect of power-law degrees on the navigability of small worlds : [extended abstract]. In Srikanta Tirthapura and Lorenzo Alvisi, editors, *PODC*, pages 240–249. ACM, 2009.

[FGK+09] Pierre Fraigniaud, Cyril Gavoille, Adrian Kosowski, Emmanuelle Lebhar, and Zvi Lotker. Universal augmentation schemes for network navigability : Overcoming the $\sqrt{n}$-barrier. *Theoretical Computer Science*, 410(21-23) :1970–1981, May 2009.

[FGP04] Pierre Fraigniaud, Cyril Gavoille, and Christophe Paul. Eclecticism shrinks even small worlds. In 23^{rd} *Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 169–178. ACM Press, July 2004.

[FK09] Pierre Fraigniaud and Amos Korman. On randomized representations of graphs using short labels. In *SPAA '09 : Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, pages 131–137, New York, NY, USA, 2009. ACM.

[FLL06] Pierre Fraigniaud, Emmanuelle Lebhar, and Zvi Lotker. A doubling dimension threshold $\Theta(\log \log n)$ for augmented graph navigability. Research Report 2006-09, LIP, École Normale Supérieure de Lyon, February 2006.

[FMMN07] Paolo Ferragina, Giovanni Manzini, Veli Mäkinen, and Gonzalo Navarro. Compressed representations of sequences and full-text indexes. *ACM Transactions on Algorithms*, 3(2) :A20, May 2007.

[FPRU90] Uriel Feige, David Peleg, Prabhakar Raghavan, and Eli Upfal. Randomized broadcast in networks. *Random Structures and Algorithms*, 1, 1990.

[FR91] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software - Practice and Experience*, 21(11) :1129–1164, 1991.

[Fra05] Pierre Fraigniaud. A new perspective on the small-world phenomenon : Greedy routing in tree-decomposed graphs. In 13^{th} *Annual European Symposium on Algorithms (ESA)*, volume 3669 of Lecture Notes in Computer Science, pages 791–802. Springer, February 2005.

[GGN04] J. Gao, L.J. Guibas, and A. Nguyen. Deformable spanners and applications. In *Proc. of SCG '04*, pages 190–199. ACM Press, 2004.

[GH99] Cyril Gavoille and Nicolas Hanusse. Compact routing tables for graphs of bounded genus. In Jiří Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, 26^{th} *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1644 of Lecture Notes in Computer Science, pages 351–360. Springer, July 1999.

[GH08] Cyril Gavoille and Nicolas Hanusse. On compact encoding of pagenumber k graphs. *Discrete Mathematics & Theoretical Computer Science*, 2008. To appear.

[GHRU97] Himanshu Gupta, Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman. Index selection for olap. In *Proceedings of ICDE'97 conference*, 1997.

[GKM+03] Dimitrios Gunopulos, Roni Khardon, Heikki Mannila, Sanjeev Saluja, Hannu Toivonen, and Ram Sewak Sharma. Discovering all most specific sentences. *ACM Trans. Database Syst.*, 28(2) :140–174, 2003.

[GMR06] Alexander Golynski, J. Ian Munro, and Satti Srinivasa Rao. Rank/select operations on large alphabets : a tool for text indexing. In 17^{th} *Symposium on Discrete Algorithms (SODA)*, pages 368–373. ACM-SIAM, January 2006.

[GN09] Omer Giménez and Marc Noy. Asymptotic enumeration and limit laws of planar graphs. *Journal of the American Mathematical Society*, 22(2) :309–329, 2009.

[GP96] Cyril Gavoille and Stéphane Pérennès. Memory requirement for routing in distributed networks. In 15^{th} *Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 125–133. ACM Press, May 1996.

[GP03] Cyril Gavoille and David Peleg. Compact and localized distributed data structures. *Distributed Computing*, 16(2-3) :111–120, September 2003. PODC 20-Year Special Issue.

[GPPR01] Cyril Gavoille, David Peleg, Stéphane Pérennès, and Ran Raz. Distance labeling in graphs. In 12^{th} *Symposium on Discrete Algorithms (SODA)*, pages 210–219. ACM-SIAM, January 2001.

[GZ05] Karam Gouda and Mohammed J. Zaki. GenMax : An efficient algorithm for mining maximal frequent itemsets. *Data Mining and Knowledge Discovery*, 11(3) :223–242, 2005.

[Her90] T. Herman. Probabilistic self-stabilization. *Information Processing Letters*, 35(2) :63–67, 1990.

[HI87] Lenwood S. Heath and Sorin Istrail. The pagenumber of genus g graphs is $O(g)$. In 19^{th} *Annual ACM Symposium on Theory of Computing (STOC)*, pages 388–397. ACM Press, May 1987.

[HKK04] Nicolas Hanusse, Evangelos Kranakis, and Danny Krizanc. Searching with mobile agents in networks with liars. *Discrete Applied Mathematics*, 137(1) :69–85, 2004.

[HKKK08] Nicolas Hanusse, Dimitris Kavvadias, Evangelos Kranakis, and Danny Krizanc. Memoryless search algorithms in a network with faulty advice. *Theoretical Computer Science (TCS)*, 402(2-3) :190–198, 2008.

[HKL00] Xin He, Ming-Yang Kao, and Hsueh-I Lu. A fast general methodology for information-theoretically optimal encodings of graphs. *SIAM Journal on Computing*, 30(3) :838–846, 2000.

[HLW06] S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4) :439–561, 2006.

[HMT09a] Nicolas Hanusse, Sofian Maabout, and Radu Tofan. A view selection algorithm with performance guarantee. In *International Conference on Extending Database Technology*, page électronique, St Petersburg Russie, 2009.

[HMT09b] Nicolas Hanusse, Sofian Maabout, and Radu Tofan. Algorithme distribué pour l'extraction des fréquents maximaux. In Augustin Chaintreau and Clemence Magnien, editors, *Algotel*, Carry-Le-Rouet France, 2009. H. : Information Systems/H.3 : INFORMATION STORAGE AND RETRIEVAL.

[HMT09c] Nicolas Hanusse, Sofian Maabout, and Radu Tofan. Matérialisation Partielle des Cubes de Données. In *25èmes journées "Bases de Données Avancées" BDA'09*, pages 1–20, Namur Belgique, 10 2009.

[HR04] Daniel Heesch and Stefan M. Rüger. Nn^{k} networks for content-based image retrieval. In *ECIR*, pages 253–266, 2004.

[JM04] Fabien Jourdan and Guy Melançon. Multiscale hybrid mds. In *8th IEEE international conference on Information Visualisation*, pages 388–393, 2004.

[Joh] C. Johnen. Service time optimal self-stabilizing token circulation protocol on anonymous unidirectional rings.

[KK99] Evangelos Kranakis and Danny Krizanc. Searching with uncertainty. In Cyril Gavoille, Jean-Claude Bermond, and André Raspaud, editors, *SIROCCO*, pages 194–203. Carleton Scientific, 1999.

[KL04] Robert Krauthgamer and James R. Lee. Navigating nets : Simple algorithms for proximity search. In 15^{th} *Symposium on Discrete Algorithms (SODA)*, pages 798–807. ACM-SIAM, January 2004.

[Kle00] Jon Kleinberg. The small-world phenomenon : An algorithmic perspective. In 32^{nd} *Annual ACM Symposium on Theory of Computing (STOC)*, pages 163–170. ACM Press, May 2000.

[KP93] Guy Kortsarz and David Peleg. Generating low-degree 2-spanners. Technical Report CS93-07, The Weizmann Institute of Science, 1993.

[KR99] Davis King and Jarek Rossignac. Guaranteed 3.67V bit encoding of planar triangle graphs. In 11^{th} *Canadian Conference on Computational Geometry (CCCG)*, pages 146–149, August 1999.

[KSSV00] Richard M. Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vöcking. Randomized rumor spreading. In *IEEE Symposium on Foundations of Computer Science*, pages 565–574, 2000.

[KW95] Kenneth Keeler and Jeffery Westbrook. Short encodings of planar graphs and maps. *Discrete Applied Mathematics*, 58(3) :239–252, 1995.

[Liu88] Yanpei Liu. Enumeration of simple planar maps. *Utilitas Mathematica*, 34 :97–104, 1988.

[LLYD02] Zonghua Li, Ying-Cheng Lai, Nong Ye, and Partha Dasgupta. Connectivity distribution and attack tolerance of general networks with both preferential and random attachments. *Physics Letters A*, 303 :337–344, 2002.

[Lu02a] Hsueh-I Lu. Improved compact routing tables for planar networks via orderly spanning trees. In 8^{th} *Annual International Computing & Combinatorics Conference (COCOON)*, volume 2387 of Lecture Notes in Computer Science, pages 57–66. Springer, August 2002.

[Lu02b] Hsueh-I Lu. Linear-time compression of bounded-genus graphs into information-theoretically optimal number of bits. In 13^{th} *Symposium on Discrete Algorithms (SODA)*, pages 223–224. ACM-SIAM, January 2002.

[LW87] Valery A. Liskovets and Timothy R. Walsh. Ten steps to counting planar graphs. *Congressus Numerantium*, 60 :269–277, 1987.

[Mal88] Seth M. Malitz. Genus g graphs have pagenumber $O(\sqrt{g})$. In 29^{th} *Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 458–468. IEEE Computer Society Press, October 1988.

[MCH+09] Tomer Moscovich, Fanny Chevalier, Nathalie Henry, Emmanuel Pietriga, and Jean-Daniel Fekete. Topology-aware navigation in large networks. In Dan R. Olsen Jr., Richard B. Arthur, Ken Hinckley, Meredith Ringel Morris, Scott E. Hudson, and Saul Greenberg, editors, *CHI*, pages 2319–2328. ACM, 2009.

[MHP06] Manor Mendel and Sariel Har-Peled. Fast construction of nets in low dimensional metrics, and their applications. *SIAM Journal on Computing*, 35(5) :1148–1184, 2006.

[Mil67] Stanley Milgram. The small-world problem. *Psychology Today*, 2 :60–67, 1967.

[Mor03] Tamas F. Mori. On random trees. *Studia Sci. Math. Hungar.*, 39 :143–155, 2003.

[Mor05] Tamas F. Mori. The maximum degree of the Barabasi-Albert random tree. *Combinatorics, Probability and Computing*, 14 :339–348, 2005.

[MR95] Michael Molloy and Bruce Reed. A critical point for random graphs with a given degree sequence. *Random Structures and Algorithms*, 6(2-3) :161–179, 1995.

[MR97] J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses, static trees and planar graphs. In 38^{th} *Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 118–126. IEEE Computer Society Press, October 1997.

[MR01] J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses, static trees and planar graphs. *SIAM Journal on Computing*, 31(3) :762–776, 2001.

[MRC03] Alistair Morrison, Greg Ross, and Matthew Chalmers. Fast multidimensional scaling through sampling, springs and interpolation. In *IEEE Symposium on Information Visualization*, pages 68–77, 2003.

[MSZ00] Yves Metivier, Nasser Saheb, and Akka Zemmari. Randomized rendezvous. Trends in mathematics, pages 183–194, 2000.

[MSZ02] Yves Metivier, Nasser Saheb, and Akka Zemmari. Randomized local elections. *Information processing letters*, 82 :313–320, 2002.

[OPT03] Deryk Osthus, Hans Jürgen Prömel, and Anusch Taraz. On random planar graphs, the number of planar graphs and their triangulations. *Journal of Combinatorial Theory, Series B*, 88(1) :119–134, 2003.

[PS03] Dominique Poulalhon and Gilles Schaeffer. Optimal coding and sampling of triangulations. In 30^{th} *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2719 of Lecture Notes in Computer Science, pages 1080–1094. Springer, July 2003.

[Ros] L. Rosaz. Self-stabilizing token circulation on asynchronous uniform unidirectional rings.

[Ros99] Jarek Rossignac. Edgebreaker : Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1) :47–61, 1999.

[Sch90] Walter Schnyder. Embedding planar graphs on the grid. In 1^{st} *Symposium on Discrete Algorithms (SODA)*, pages 138–148. ACM-SIAM, January 1990.

[Sch99] Gilles Schaeffer. Random sampling of large planar maps and convex polyhedra. In 31^{st} *Annual ACM Symposium on Theory of Computing (STOC)*, pages 760–769. ACM Press, May 1999.

[SDS00] J.F.F. Mendes S.N. Dorogovtsev and Samukhin. Structure of growing networks with preferential linking. *Physical Review Letters*, 85(21) :4633–4636, 2000.

[Sli07] Aleksandrs Slivkins. Distance estimation and object location via rings of neighbors. *Distributed Computing*, 19(4) :313–333, 2007.

[TCFS08] Zohreh Asgharzadeh Talebi, Rada Chirkova, Yahya Fathi, and Matthias Stallmann. Exact and inexact methods for selecting views and indexes for olap performance improvement. In *Proceedings of EDBT'08*, 2008.

[Tur84] György Turán. Succinct representations of graphs. *Discrete Applied Mathematics*, 8(3) :289–294, 1984.

[Tut62] William T. Tutte. A census of planar triangulations. *Canadian Journal of Mathematics*, 14 :21–38, 1962.

[WS98] D.J. Watts and S.H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(409-410) :409–410, 1998.

[Yan89] Mihalis Yannakakis. Embedding planar graphs in four pages. *Journal of Computer and System Sciences*, 38(1) :36–67, 1989.

[YsZwCdL05] Xin Yao, Chang shui Zhang, Jin wen Chen, and Yan da Li. On the formation of degree and cluster-degree correlations in scale-free networks. *Physica A*, 353 :661–673, 2005.

[Yul25] G. Yule. A mathematical theory of evolution, based on the conclusions of dr. j. c. willis, f.r.s. *Philosophical Transactions of the Royal Society of London*, B-213 :21–87, 1925.

[YWB03] Vladimir Yanovski, Israel A. Wagner, and Alfred M. Bruckstein. A distributed ant algorithm for efficiently patrolling a network. *Algorithmica*, 37(3) :165–186, 2003.