UNIVERSITÉ ASSANE SECK DE ZIGUINCHOR

UFR SCIENCES ET TECHNOLOGIES
DÉPARTEMENT D'INFORMATIQUES

Mémoire de Master 2

DOMAINE : SCIENCES ET TECHNOLOGIES
MENTION : INFORMATIQUE
SPÉCIALITÉ : GÉNIE LOGICIEL
OPTION : THÉORIE DES GRAPHES ET OPTIMISATION

Thème :

Routage compact de plus court chemin dans les graphes de Halin cubiques complets

Présenté par :

Assane GUEYE

Sous la direction de : **Professeur Youssou DIENG**

Soutenu publiquement le Vendredi 22 Juillet 2022 devant le jury composé de :

Prénom(s) et Nom	Grade	Qualité	Université
Salomon SAMBOU	*Professeur*	Président	UASZ
Daouda Niang DIATTA	*Docteur*	Rapporteur	UASZ
Ibrahima DIOP	*Docteur*	Examinateur	UASZ
Youssou DIENG	*Professeur*	Directeur	UASZ

Année universitaire : 2020/2021

REMERCIEMENTS

Je souhaite remercier Dr Youssou DIENG, d'avoir accepté d'être mon encadreur. Je tiens à lui exprimer toute ma gratitude pour sa disponibilité, ses conseils et l'encadrement exceptionnel qu'il m'a apporté durant l'ensemble ce mémoire de master. Il a su me transmettre sa rigueur mathématique et sa passion pour la recherche. Pour toutes ces raisons je lui témoigne ma plus sincère gratitude.

J'aimerai aussi remercier Dr Daouda Niang DIATTA, d'être parmi les personnes qui stimulent mon esprit de recherche et me poussent a vouloir être un chercheur. Je suis très reconnaissant de l'attention qu'il a porté a mon travail et pour m'avoir fait l'honneur d'accepter de rapporter mon mémoire.

J'adresse mes sincères remerciements à Dr Ibrahima DIOP, pour l'intérêt qu'il a porté à mon travail et avoir accepté d'examiner ce travail.

Je souhaite également remercier Monsieur Salomon SAMBOU ; Professeur à l'Université Assane Seck de Ziguinchor, d'avoir accepté de présider le jury de ma soutenance de mémoire de master.

Enfin, je remercie toute ma famille et mes amis qui m'ont toujours soutenu et encouragé. Ils ont su trouver les mots justes et être présentes quand j'en avais besoin. Je remercie du fond du cœur ma mère sans qui je n'aurais jamais pu être ce que je suis devenu.

DÉDICACES

A ma mère Fatma THIAM et ma grand-mère Codou DIOUF,

A mon grand père feu Baye Ndery THIAM,

A mon père Malick GUEYE,

A mes frères et sœur,

A mes maman Aida, Nabou et Ndeye Fary THIAM,

A mes oncles Babacar, Bassirou et Cheikh THIAM,

A toute la famille THIAM,

A tous ceux qui me sont chers,

J'offre ce modeste travail

Schéma de routage compact de plus court chemin dans les graphes de Halin cubiques complets

Résumé :

Savoir comment transmettre une information est fondamental dans un réseau. Il est essentiel que chaque entité du réseau dispose d'une fonction de routage lui permettant de décider localement, avec sa vue du réseau, du chemin par lequel l'information doit passer. Dans le routage compact, on cherche à mettre en place de tels algorithmes tout en optimisant la table de routage, le chemin parcouru par le message et la latence du routeur.

Dans leur travaux, sur le routage compact, Dieng et al., ont posé la question de savoir s'il est possible de router par de plus court chemin dans un graphe de Halin avec des tables de routage et des entêtes de message de taille $O(\log n)$ bits. Les seuls graphes possédant un tel schéma de routage sont les arbres, les graphes planaire extérieurs et les (k,r)-constellations. En ce qui est des graphes de Halin, les résultats apportés sont de Bassène et al. avec la proposition d'un schéma de routage compact avec un facteur d'étirement de 2.

Dans ce mémoire, nous proposons un schéma de routage de plus cours chemin, dans les graphes de Halin cubiques complets ; une sous famille des graphe de Halin.

Discipline : Informatique

Mots clefs : Routage compact, routage par intervalle
graphe planaire, graphe planaire-extérieur,
graphe de Halin

Université Assane Seck de Ziguinchor (UASZ)

Compact shortest path routing scheme in complete cubic Halin graphs

Abstract :

Knowing how to transmit information is fundamental in a network. It is essential that each entity in the network has a routing function that allows it to decide locally, with its view of the network, which path the information should take. In compact routing, we try to implement such algorithms while optimizing the routing table.

In their work on compact routing, Dieng et al. asked the question of whether it is possible to route through the shortest path in a Halin graph with routing tables and message headers of size $O(\log n)$ bits. The only graphs with such a routing scheme are trees, external planar graphs and (k,r)-constellations. As far as Halin graphs are concerned, the results are from Bassène et al. with the proposal of a compat routing scheme with a stretching factor of 2.

In this paper, we propose a shortest path routing scheme in complete cubic Halin graphs ; a sub-family of Halin graphs.

Discipline : Computer science

Keywords : Compact Routing, interval routing
planar graph, outerplanar-graph
Halin graph

Assane Seck University of Ziguinchor (UASZ)

TABLE DES MATIÈRES

TABLE DES FIGURES

CHAPITRE

1

INTRODUCTION GÉNÉRALE

1.1 Réseaux

La notion de réseau est aujourd'hui très répandue, en plusieurs domaines. Depuis longtemps on parle de réseaux de transport, qu'il s'agisse de routes, de chemins de fer, d'égouts, de gaz, d'électricité, de la poste, ou d'autres réseaux physiques permettant la circulation d'objets matériels d'un endroit à l'autre. Plus récemment les réseaux de télécommunication se sont multipliés, et c'est sans doute dans cette multiplication que réside une des explications de la grande popularité de la notion de réseau, que ce soit dans le monde physique ou dans le monde social. Ainsi, *un réseau peut être défini comme un ensemble d'entités interconnectées.* De ce fait, ces entités peuvent représenter des villes, des personnes, des ordinateurs,...etc. Dans ce qui suit, nous allons donné quelques exemples de réseaux.

1.1.1 Réseaux informatique

Un réseau informatique est un ensemble de matériels et de périphériques. Ces derniers sont reliés par une même technique, leur permettant de communiquer entre eux.

Notons qu'un réseau informatique permet la communication de plusieurs ordinateurs ou périphériques entre eux (exemple l'internet est le plus grand réseau informatique), le partage de documents, de périphériques, les jeux à plusieurs participants...etc.

Un réseau informatique a aussi une portée géographique, c'est-à-dire l'étendue de l'espace où sont répartis les équipements du réseau. Ainsi, suivant ce porté géographique, il sera question de parler de :

— **réseau local ou LAN (Local Area Network)**, il désigne les réseaux avec une étendue spéciale limitée. La plupart du temps, les *LAN* sont utilisés dans les domiciles privés ou dans des entreprises pour mettre en place un réseau local ou un réseau d'entreprise. Ils permettent aux différents appareils de communiquer les uns avec les autres. Sans un tel réseau, l'échange de données serait impossible. Un réseau local est composé en général d'au moins deux appareils, mais peut également connecter plusieurs milliers d'appareils. Si des distances importantes doivent être couvertes, on utilise alors plutôt des réseaux métropolitains ou des réseaux étendus. Un Local Area Network peut connecter des ordinateurs, des smartphones, des imprimantes, des scanners, des périphériques de stokage, des serveurs et d'autres appareils en réseau entre eux et avec internet.

— **réseaux métropolitains ou MAN (Metropolitan Area Network)** interconnectent plusieurs *LAN* géographiquement proches (au maximum quelques dizaines de kilomètres) à des débits importants. Ainsi, un MAN permet à deux noeuds distants de communiquer comme si ils faisaient partie d'un même réseau local.

— **réseaux étendus ou WAN (Wide Area Network)**, ces réseaux s'étendent sur de vastes zones géographiques et relient plusieurs réseaux plus petits comme des *LAN* (Local Area Networks) ou des *MAN* (metropolitain Area Networks). C'est la raison pour laquelle ils sont uniquement utilisés dans un contexte professionnel. Les *WAN* publics sont exploités par les fournisseurs d'accès à internet afin de permettre à leurs clients d'accéder au web.

— **réseau basé sur la technique WIFI ou WLAN (Wireless Local Area Network)**, un *WLAN* est un réseau permettant de couvrir l'équivalent d'un réseau local d'entreprise, soit une portée d'environ une centaine de mètres. Il permet de relier entre-eux les terminaux présents dans la zone de couverture.

Les réseaux peuvent aussi être classer suivant leur mode d'utilisation :
— **le réseau intranet**, est un réseau informatique, mis en place au sein d'une entreprise ou de toute autre entité équivalente. Il permet aux collaborateurs de cette entreprise d'échanger des informations et des documents dans un environnement sécurisé, au sein d'un espace dont l'accès est restreint à un groupe défini.
— **le réseau extranet**, désigne un réseau de télécommunications dont la mission principale est de rendre les échanges à distance plus faciles entre deux ou plusieurs parties. C'est dans le monde professionnel et celui des entreprises que l'on

rencontre le plus fréquemment l'extranet.
Un neurone artificiel n'est rien d'autre qu'un processeur élémentaire. Ce dernier reçoit en permanence un nombre variable d'entrées en provenance de neurones situes en amonts.

— **et internet**, le réseau des réseaux.

1.1.2 Réseaux de neurones

Les réseaux de neurones, également connus sous le nom de réseaux de neurones artificiels ou de réseaux de neurones à impulsions constituent un sous-ensemble de l'apprentissage machine et sont au cœur des algorithmes de l'apprentissage en profondeur. Leur nom et leur structure sont inspirés par le cerveau humain. En effet, ces réseaux imitent la façon dont les neurones biologiques s'envoient mutuellement des signaux.

Un neurone est avant tout un opérateur mathématique, dont la valeur numérique se calcule par quelques lignes d'un programme. Un neurone réalise une somme pondérée suivie d'une fonction non linéaire f. Cette fonction f doit être bornée, continue et dérivable. Elle peut aussi avoir la forme d'une fonction seuil si le résultat recherché est de type booléen (soit 0 ou 1). Les fonctions les plus fréquemment utilisées sont les fonctions sigmoïdes. Pour plus de détails sur les réseaux de neurone voir [TOU92], [NSHO08].

1.2 Théorie des graphes

Dans la plupart du temps pour résoudre un problème concret, on est souvent amené à faire des dessins sur des papiers. Ces dessins représentent dans l'espace le problème à résoudre. On constate que souvent ces dessins sont composés de points et de lignes reliant deux à deux certains de ces points. Ces dessins sont appelés des graphes. Ainsi, la théorie des graphes est la discipline mathématique et informatique qui étudie les graphes, permettant de modéliser beaucoup de problèmes concrets en vue de trouver des méthodes de résolution spécifiques. Par exemple :

1.2.1 Graphe d'un réseau routier

le réseau routier d'un pays peut être représenté par un graphe dont les sommets sont les villes. Si l'on considère que toutes les routes sont à double sens, on utilisera un graphe non orienté et on reliera par une arête tout couple de sommets correspondant à deux villes reliées par une route. Ces arêtes pourront être pondérées par la longueur des routes correspondantes. Étant donné un tel graphe, on pourra s'intéresser, par exemple, à la résolution des problèmes suivants :

- Quel est le plus court chemin, en nombre de kilomètres, passant par un certain nombre de villes données ?

- Quel est le chemin traversant le moins de villes pour aller d'une ville à une autre ?
- Est-il possible de passer par toutes les villes sans passer deux fois par une même route ?

1.2.2 Graphe d'un processus à étapes

Certains problèmes peuvent être spécifiés par un état initial, un état final, un certain nombre d'états intermédiaires et des règles de transition précisant comment on peut passer d'un état à l'autre. Résoudre le problème consiste alors à trouver une suite de transitions permettant de passer de l'état final.

Beaucoup de jeux et autres "casse-tête" peuvent être modélisés ainsi. Considérons, par exemple, le problème du chou, de la brebis et du loup : *Un brave homme se trouve au bord d'une rivière qu'il souhaite traverser, en compagnie d'un loup, d'une brebis et d'un chou. Malheureusement, il ne dispose que d'une petite barque, ne pouvant porter en plus de lui-même qu'un seul de ses compagnons (le loup ou la brebis ou le chou). Bien sûr, la brebis refuse de rester seule avec le loup, tandis que le chou refuse de rester seul avec la brebis. Comment peut-il s'y prendre pour traverser la rivière avec ses trois compagnons et continuer son chemin ?*

L'état initial est l'état où tout le monde est sur la rive gauche de la rivière, tandis que l'état final est l'état où tout le monde est sur la rive droite de la rivière. La règle de transition est la suivante : si l'homme est sur une rive avec certains de ses compagnons, alors il peut passer sur l'autre rive, soit seul, soit accompagné par un seul de ses compagnons se trouvant sur la même rive que lui, sous réserve qu'il ne laisse pas le loup seul avec la brebis, ou la brebis seule avec le chou. On peut modéliser ce problème par un graphe non orienté, dont les sommets représentent les états possibles, et les arêtes le fait qu'on peut passer d'un état à l'autre par une transition.

1.2.3 Graphe d'un automates finis

Un automate fini permet de reconnaître un langage régulier et peut être représenté par un graphe orienté et étiqueté.

Étant donné un tel graphe, on peut s'intéresser, par exemple, à la résolution des problèmes suivants :

- Existe-t-il un chemin allant du sommet initial (1) au sommet final (3) ?
- Quel est le plus court chemin entre deux sommets donnés ?
- Existe-t-il des sommets inutiles, par lesquels aucun chemin allant du sommet initial à un sommet final ne peut passer ?

1.3 Routage compact

Le fait d'envoyer un message entre deux pairs de nœuds dans un réseau (ou un graphe) est appelé le routage.

Le routage est le mécanisme par lequel des chemins sont sélectionnés dans un réseau pour acheminer les données d'un expéditeur jusqu'à un ou plusieurs destinataires. Le routage est une tâche exécutée dans de nombreux réseaux, tels que le réseau téléphonique, les réseaux de données électroniques comme Internet, et les réseaux de transports. Sa performance est importante dans les réseaux décentralisés, c'est-à-dire où l'information n'est pas distribuée par une seule source, mais échangée entre des agents indépendants.

Dans un réseau informatique, lorsqu'on veut envoyer (router) des messages entre deux entités d'un réseau, les données vont parcourir un ensemble de routeurs, qui appliquent tous les mêmes règles de base pour que le transfert des paquets soit cohérents. Un routeur est un élément intermédiaire dans un réseau informatique assurant le routage des données.

Chaque routeur doit décider des chemins par lesquels envoyés les données. Par conséquent, il est nécessaire de mettre en disposition d'un routeur un ensemble d'informations de la topologie du réseau en question. Cet ensemble d'informations constitue ce que l'on appelle ici le schéma de routage.

Un schéma de routage est composé :

— d'adresse pour chaque nœud.

— des entêtes ajoutées a tout message, constituées généralement de l'adresse du nœud destinataire.

— de numéro de port (un entier positif) pour chaque lien reliant deux machines, le sens de parcours est pris en compte, c'est-à-dire le numéro de port allant d'une machine u à une machine v correspond à $port(u,v)$ et le numéro de port allant de v à u correspond à $port(v,u)$. Le numéro 0 est souvent attribué au lien faisant connexion entre le routeur et la machine qui lui est associée.

— d'une fonction de routage permettant de calculer une nouvelle entête e' et un port de sortie p', lorsqu'un message arrive par un port p avec une entête e. Et d'envoyer le message avec sa nouvelle entête vers le port p'.

— d'un mémoire local pour chaque routeur, permettant de stocker la fonction de routage et un certain nombre d'informations sur la topologie du réseau.

Cependant, un schéma de routage est dit valide que s'il permet de transmettre des messages entre tout couple de sommets.

1.4 Problématique

Notons que, si la longueur des routes induites par un schéma de routage est un critère de qualité, l'espace requis par les tables de routage, la taille des adresses et des entêtes

de message, ou la latence (c'est-à-dire le temps mis pour un routeur à décider vers quel port envoyé le message) sont des paramètres essentiels qui doivent être optimisés [1].

Bien que les routeurs soient des matériaux incontournables dans les réseaux informatiques, ils disposent cependant d'espace de stockage limité et d'un coût relativement proportionnel à cette capacité.

Ainsi, la politique ayant pour but de réduire au mieux la taille de l'ensemble des informations nécessaires aux routeurs pour accomplir les tâches qui leur sont allouées est appelée routage compact.

Beaucoup d'articles scientifiques dans le domaine de l'optimisation, ont été consacrés à la recherche de schéma de routage compact en étudiant la structure de la topologie sous-jasent (le graphe correspondant au réseau).

Dans le cas d'une structure arbitraire à n nœuds, la taille des tables de routage est de $n^{\Theta(1/s)}$, avec s le facteur d'étirement, c'est-à-dire le ratio maximum sur toutes les paires de nœuds (u,v) entre la longueur de la route et la distance entre u et v. Pour des structures triangulaires de degré borné, la taille des tables ou des adresses est de $\Omega(n^{1/4})$ [AGGM06].

Pour des structures simples des solutions poly-logarithmique existent. Par exemple pour les arbres, P. Fraigniaud et C. Gavoille dans [FG02] et Thorup et Zwick dans [TZ01] ont obtenus indépendamment un schéma de routage de plus court chemin avec des tables de routage de $O(\log n)$ bits.

Y. Dieng et C. Gavoille en se basant sur les travaux de P. Fraigniaud et C. Gavoille ont proposé dans [DG11] un schéma de routage compact de plus court chemin pour les graphes planaires extérieurs. Résultat qu'ils ont étendu dans les graphes (k,r)-constellation.

A. Bassène dans son mémoire de magister [BAS17] a proposé un schéma de routage compact pour les graphes de Halin mais qui est de facteur d'étirement 2.

Nous avions basé notre réflexion sur la détermination d'un schéma de routage de plus court chemin pour les graphes de Halin. Cette tentative n'ayant pas portée ces fruits, nous avons essayé de regarder les sous-familles de ce graphe. Ce qui nous a permis de proposer un schéma de routage compact de plus court chemin pour une sous-famille de graphes de Halin, les graphes de Halin cubique complets.

1.5 Structure du mémoire

Dans cette section nous allons donné les grande lignes développées dans chaque chapitre de ce document.

1. L'optimisation est une branche des mathématiques cherchant à modéliser, à analyser et à résoudre analytiquement ou numériquement les problèmes qui consistent à minimiser ou maximiser une fonction sur un ensemble fini, mais éventuellement très grand, et dont les propriétés mathématiques ne sont pas facilement caractérisables.

Chapitre 1 : Introduction générale

Dans ce chapitre d'introduction générale, nous avons abordé la notion de réseau dans sa généralité. De la notion de graphe en donnant quelques exemples de problèmes modélisés par des graphes. Nous avons aussi introduit la notion de routage compact, cette dernière est très utile dans les réseaux informatique, dans les réseaux de transport et plein d'autres type de réseaux. Avant de donner

Chapitre 2 : Généralité sur les graphes

Dans ce chapitre, nous introduisons les différentes notions utilisées en théorie des graphes de manière courante et dont nous nous servons dans ce document.

Nous y présentons également les notions liées aux mineurs de graphe, et ainsi que la notion de complexité.

Chapitre 3 : État de l'art sur le routage compact

Dans ce chapitre, nous donnons un état de l'art sur le routage compact. Dans lequel nous allons parlé de routage par intervalle, introduit dans le but d'optimisé la taille des informations ; le routage compact de plus court chemin dans les arbres [FG02, TZ01] et dans les graphes planaires extérieurs [DG11] et le schéma de routage compact dans les graphes de Halin.

Chapitre 4 : Contribution : routage compact de plus court chemin dans les graphes de Halin cubique complet

Nous présentons dans ce chapitre notre schéma de routage compact de plus court chemin sur les graphes de Halin cubiques complets, une sous famille de graphes de Halin.

Chapitre 5 : Conclusion générale et perspectives

Dans ce dernier chapitre intitulé conclusion générale et perspectives, nous avons fait un résumé de l'ensemble des résultats dans le chapitre état de l'art sur le routage compact et ainsi que de notre résultat sur les graphes de Halin cubiques complets. Nous avons aussi énoncé un ensemble de perspectives de recherche. Chaque perspective est donné sous forme de question.

CHAPITRE

2

GÉNÉRALITÉ SUR LES GRAPHES

La théorie des graphes, de même qu'une grande partie de l'analyse, de la théorie des probabilités, de la topologie, doivent leur existence aux travaux du mathématicien *Leonhard Euler* [1].

Le concept de graphe est né du problème des sept ponts de Kœnigsberg. Qui consistait, à montrer "s'il est possible de parcourir un graphe en partant d'un sommet tout en empruntant une fois, ni plus, ni moins, chaque arête pour enfin revenir au sommet de départ ?" Depuis lors les graphes sont devenus des outils de modélisation incontournables pour de nombreux problèmes dans beaucoup de domaines, comme par exemple :

la chimie et la biologie (modélisation de molécules, ADN,...)
l'économie (planning de livraisons, gestion de flots, ordonnancement...)
la sociologie ou science sociale (généalogie, phénomènes de masse, conflits,...)
la cartographie (réseau routier, réseau internet,...) ...etc.

Ainsi, ce chapitre présente entre autre quelques notions sur les graphes probablement connus des lecteurs avertis, de la notion de mineur d'un graphe et de la notion de complexité. Néanmoins le lecteur peut se référer au livre de Claude Berge [BER70], les livres de Bondy et Murty [BM76] [BM08], pour plus de détails sur la notion de graphe. [MT01, MOH01] pour plus de détails sur la notion de mineur et de complexité.

1. Leonhard Euler, n'est le 15 avril 1707 à Bâle(Suisse) et mort le 7 septembre 1783 à 76 ans, est un mathématicien et physicien suisse. Euler fit d'importantes découvertes dans des domaines variés que le calcul infinitésimal et la théorie des graphes.

2.1 Graphes et propriétés

2.1.1 Graphe non orienté

Définition 2.1 *Un graphe non orienté G est un triplet (V, E, Γ), où*
- *V est un ensemble non vide de sommets (vertices en anglais),*
- *et Γ est une fonction d'incidence qui à chaque arête de E associe une paire d'éléments de V, non nécessairement distincts.*

Le nombre de sommets et d'arêtes d'un graphe G, sont définis respectivement par $n = |V(G)|$ et $m = |E(G)|$ et sont appelés dans cet ordre, l'*ordre* et la *taille* de G.

On appelle *arête* $e \in E(G)$ un couple de sommets (u, v), et on dit que $u, v \in V(G)$ sont les *extrémités* de l'arête e.

Une *arête* (u, v) est une *boucle* si $u = v$. Deux arêtes sont parallèles si elles ont les mêmes extrémités. Un sommet est dit *isolé* s'il n'est incident à aucune arête et *pendant* s'il est incident à une seule arête.

Les extrémités d'une arête e sont dits *incidents* à e, et vice-versa. Les sommets incidents à une même arête sont *adjacents*, de la même manière deux arêtes incidentes à un même sommet sont adjacentes.

On dit que deux sommet u et v distincts sont *voisins*, s'ils sont incidents à une même arête. On peut aussi dire u est un voisin de v ou v est un voisin de u. Ainsi l'ensemble des voisins d'un sommet u est appelé le *voisinage* de u, noté par $N(u)$.

Si deux sommets sont reliés par p arrête on dit que G est un $p\text{-}graphe$. Ainsi pour $p = 1$ on a un $1\text{-}graphes$, donc tout au long de ce document nous ne considérons que des $1\text{-}graphes$ sauf mention contraire.

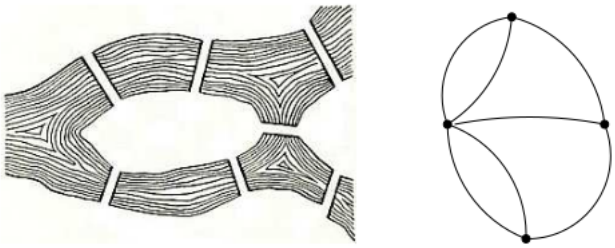

FIGURE 2.1 – Graphe non orienté représentant les pont de Kœnigsberg

Un graphe non orienté est simple s'il n'a ni arête parallèles ni boucle. Notons que, dans la plus part du temps on utilise le mot *arête* pour les graphes non orientés et *arc* pour les graphes orientés. La figure 2.1 ci-dessous donne un exemple de graphe non orienté dont les arêtes représentent les ponts de Kœnigsberg et les sommets les sous régions reliés par ces derniers.

2.1.2 Graphe orienté

Définition 2.2 *Un graphe orienté G est un triplet (V, E,* Γ*), où*

- *V est un ensemble non vide de sommets (vertices en anglais),*
- *et* Γ *est une fonction d'incidence qui à chaque arc (u,v) de E associe une paire d'éléments de V, non nécessairement distincts. u est le sommet initial et v est le sommet terminal.*

Dans un graphe orienté les arcs (u, v) et (v, u) sont distincts. La plus part des propriétés valident pour les graphes non orientés le sont pour les graphes orientés. La figure 2.2 représente un exemple de graphe orienté.

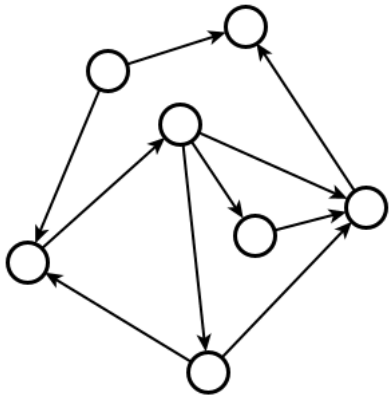

FIGURE 2.2 – Un exemple de graphe orienté à 7 sommets et 10 arcs.

2.1.3 Degré

Définition 2.3 *Le degré deg(u) d'un sommet u de G est le nombre d'arêtes incidentes à u dans G.*

Chaque boucle incidente à un sommet u est compte deux fois

Théorème 2.1 *Pour tout graphe* $G = (V(G), E(G))$, $\sum_{u \in V(G)} deg(u) = 2 \times |E(G)|$.

En effet, pour la somme $\sum_{u \in V(G)} deg(u)$, toute arête est comptée deux fois dans le degré de u si c'est une boucle sur le sommet u. Et une fois dans le degré de u et une fois dans le degré de v, si l'arête est incidente à u et à v.

De cet théorème découle la conséquence suivante.

Corollaire 2.1 *Dans tout graphe* $G = (V(G), E(G))$, *le nombre de sommets de degré impair est pair.*

Définition 2.4 *On dit qu'un graphe non orienté G est k-régulier si tous ses sommets sont de degré k.*

Propriété 2.1 *Soient* δ *le plus petit degré et* Δ *le plus grand degré d'un graphe non orienté G de nombre de sommets n et de nombre d'arêtes m. On a alors* $n \times \delta \leq 2m \leq n \times \Delta$.

Définition 2.5 *Dans un graphe orienté on distingue*

- *le degré entrant* $deg^-(u)$ *qui est égal au nombre d'arcs de sommet terminal u,*
- *le degré sortant* $deg^+(u)$ *qui est égal au nombre d'arcs de sommet initial u,*

- *le degré deg(u) est égal à la somme* $deg^+(u) + deg^-(u)$

Théorème 2.2 *Dans un graphe orienté G,* $\sum_{u \in V(G)}(deg^+(u) + deg^-(u)) = 2 \times |E(G)|$

Dans ce qui suit nous considérons les graphes non orientés sauf mention contraire.

2.1.4 Chaîne, cycle et connexité

Définition 2.6 *Une* chaîne *dans un graphe* $G = (V, E)$ *est une séquence d'arêtes distincts* $\mu = (e_1, ..., e_k)$ *de G telle que chaque arête de* μ *ait une extrémité en commun avec l'arête précédente et l'autre extrémité en commun avec l'arête suivante.*

Le terme chaîne s'applique en pratique pour les graphes non orientés, pour les graphes orientés on parlera de chemin.

Le nombre d'arêtes de la séquence μ est appelé la longueur de μ. Une chaîne sera dite élémentaire si elle ne rencontre pas deux fois le même sommet, et simple si elle n'utilise pas deux fois la même arête.

Notons que dans le cas d'un graphe simple un chemin (une chaîne) est déterminé(e) par la succession des sommets $u_1, u_2, ..., u_k$ qu'il rencontre. Un *plus court chemin* entre deux sommets donnés est le chemin simple dont la longueur est minimum.

Définition 2.7 *Un cycle (*resp. *un circuit) est une chaîne (*resp. *un chemin)* $u_1, u_2, ..., u_k$ *tel que le sommet de départ* u_1 *coincide avec le sommet d'arrivée* u_k.

Définition 2.8 *La* distance *entre deux sommets u et v, noté* $dist_G(u, v)$ *est la longueur de la chaîne ou chemin la plus petite dans G reliant u et v.*

Définition 2.9 *Deux sommets u et v sont* connectés*, s'il existe une chaîne* μ *dans G reliant u et v.*

Définition 2.10 *Un graphe non vide* $G = (V, E)$ *est dit* connexe *si toute paire de sommets est reliée par une chaîne dans G.*

Définition 2.11 *Une* composante connexe *de G est un sous-graphe (voir la sous-section 2.1.7) connexe d'ordre maximale. Un graphe connexe possède une seule composante connexe.*

Un graphe est dit non connexe s'il possède au moins deux composantes connexes.

2.1.5 Graphe complet

Définition 2.12 *Un graphe* $G = (V, E)$ *est dit* complet*, si tout sommets* $u \in V$ *est relié à tous les autres sommets de* V.

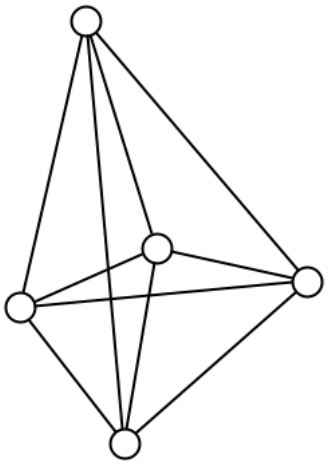

FIGURE 2.3 – Le graphe complet K_5

Dans un graphe complet K_n, tout sommet est de degré exactement $n-1$. Par exemple, le graphe K_5 de la figure 2.3 est un graphe complet, puisque tous les sommets sont de degré 4.

2.1.6 Graphe biparti

Un graphe est biparti s'il existe une partition de l'ensemble de ces sommets en deux sous-ensembles E_1 et E_2 telle que chaque arête du graphe ait une extrémité dans E_1 et l'autre dans E_2, voir figure 2.4.

Autrement dit un graphe est biparti si est seulement si il ne contient pas de cycle de longueur impaire.

Un graphe biparti est dit *biparti complet*, si chaque sommet de E_1 est relié à chaque sommet de E_2. On note le graphe par $K_{|E_1|,|E_2|}$.

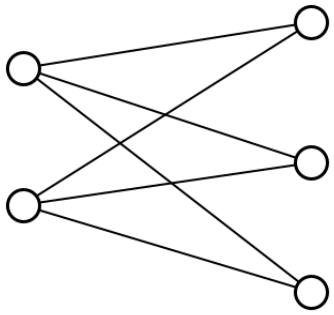

FIGURE 2.4 – Le graphe biparti $K_{2,3}$

2.1.7 Sous-graphes et graphes partiels

Définition 2.13 *Un sous-graphe d'un graphe $G = (V, E)$ est un graphe obtenu en supprimant certains sommets et toutes les arêtes qui lui sont incidentes.*

Si $G = (V, E)$ est un graphe alors $G' = (V', E')$ tel que $V' \subseteq V$ et $E' \subseteq E$, est un sous-graphe de G. Le graphe G est appelé un super-graphe de G'.

On appelle *sous-graphe d'un graphe $G = (V, E)$ engendré par $A \subset V$*, le graphe G_A dont les sommets sont ceux de A et les arêtes sont les arêtes de G ayant leurs deux extrémités dans A.

Définition 2.14 Un graphe partiel engendré par $B \subset E$ *est le graphe $G_B = (V, B)$ dont les sommets sont ceux de G et les arêtes sont de B.*

Un sous-graphe partiel de G est un sous-graphe d'un graphe partiel de G.

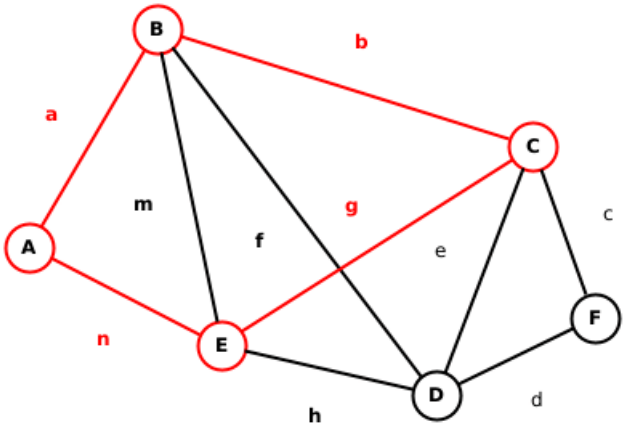

FIGURE 2.5 – Soit le graphe G défini par $G = (V, E)$, où $E = a, b, c, d, e, g, f, g, h, m, n$ et $V = A, B, C, D, E, F$.Le graphe $G_1 = (V_1, E_1)$, tel que $V_1 = A, B, C, E$ et $E_1 = a, b, g, n$, est un sous-graphe de G.

2.1.8 Graphes planaires

On dit qu'un graphe $G = (V, E)$ est *planaire* s'il peut être représenté dans le plan en utilisant des points pour les sommets et des arcs pour les arêtes, sans que deux arêtes ne se coupent, autrement dit s'il est plongeable dans le plan. Ainsi, dans le cas contraire on dit que le graphe est *non planaire*.

Les graphes K_4 et $K_{2,3}$ sont planaires, par contre K_5 et $K_{3,3}$ sont non planaires.

Lemme 2.1 *Soit G un graphe, si $|V(G)| = n \geq 3$ alors $3f \leq 2m$ avec f le nombre de face du graphe et $m = |E(G)|$.*

Preuve : Soit $\mathcal{F}$ l'ensemble de faces et $|F|$ le nombre D'arêtes qui délimitent la face F.

Chaque face F est délimitée par au moins 3 arêtes. $|F| \geq 3 \Rightarrow$ donc $\sum_{i \in \mathcal{F}} |F| \geq 3f$ Chaque arête est une frontière de deux faces et donc on peut en déduire que $\sum_{i \in \mathcal{F}} |F| = 2m$

Donc en combinant les deux equations, on obtient $3f \leq 2m$.

CQFD

Lemme 2.2 *Tout graphe planaire contient au moins un sommet de degré au plus 5.*

Preuve : Soit un graphe G. Si G possède des sommets de degré 1, alors il existe bien un sommet de degré au plus 5 dans G.

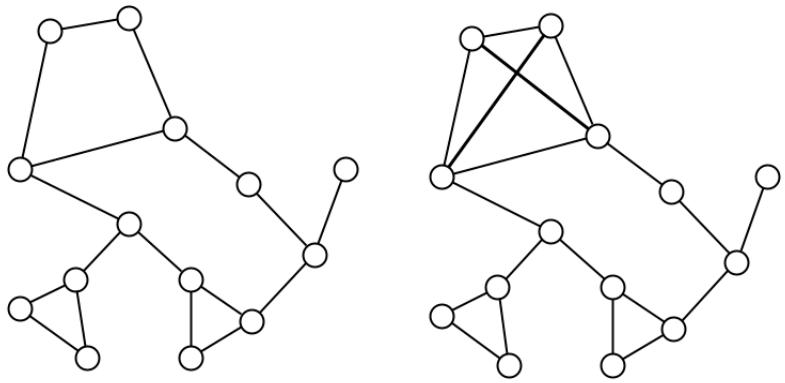

FIGURE 2.6 – Le graphe à gauche est planaire et celui à droite est non planaire

Supposons que G ne possède pas des sommets de degré 1. Raisonnons par l'absurde et supposons que pour tout sommet v de G, $deg(v) \geq 6$. On a $6n \leq 2m$ car la somme de tous les sommets est égale a 2 fois le nombre de arêtes.

En appliquant la formule d'Euler $(n - m - f = 2)$, on a $f = 2 + m - n$.

$3f \leq 2m \Rightarrow 3(2 + m - n) \leq 2m \Rightarrow m \leq 3n - 6 \Rightarrow 2m \leq 6n - 12 \Rightarrow$ Ce qui est en contradiction avec $6n \leq 2m$. Donc il existe un sommet de degré au plus 5 dans G.

CQFD

Si le graphe est planaire, alors toute représentation planaire partitionne le plan en régions connexes (au sens topologique du terme). Ces régions sont appelées des faces, notons f le nombre de faces (y compris la face extérieure). Euler a démontré le lemme suivant.

Théorème 2.3 *Tout graphe planaire connexe vérifie* $n - m + f = 2$.

Preuve : En utilisant la démonstration par récurrence sur le nombre de faces $f(G)$ de G.

Si $f(G) = 1$, toute arête de G est une arête séparatrice. Donc puisque G est connexe alors c'est un arbre. D'où $m = n - 1 = n + 1 - 2$, et le résultat est vrai.

Supposons qu'il soit vrai pour tous les graphes plans connexes avec moins de $f \geq 2$ faces, et soit G un graphe connexe avec f faces. Choisissons une arête e de G qui n'est pas une arête séparatrice. Alors $G \setminus e$ est un graphe plan connexe avec $f - 1$ faces, parce que les deux faces de G séparées par e se regroupent pour former une face de $G\setminus e$. $G\setminus e$ a $n^{'}$ sommets et $m^{'}$ arêtes.Par hypothèse de récurrence, $n^{'} - m' + f - 1 = n - (m-1) + f - 1 = 2$.

CQFD

Ce théorème a permis de démontré de nombreux résultats dont on peut citer les théorèmes 2.5 et 2.6 ci-dessous.

Théorème 2.4 *Tout graphe planaire topologique connexe avec n sommets et dont la plus petit face comporte P arête contient au plus* $\frac{P}{P-2}(n-2)$ *arêtes.*

Preuve : Soit G un graphe planaire topologique connexe avec n sommets, m arêtes, f faces et tel que le pourtour de chaque face comporte au moins P arêtes. D'après la propriété d'Euler, on a $f = m - n + 2$.

En faisant la somme des nombres d'arêtes autour de chaque face on obtient exactement le double du nombre d'arêtes. En effet, chaque arête est comptée à double puisqu'elle apparaît sur le pourtour de deux faces exactement. Comme le pourtour de chaque face contient au moins P arêtes, on a $2m \geq Pf$. En utilisant la propriété d'Euler, cette dernière inégalité se réécrit comme suit : $2m \geq P(m - n + 2)$ c'est-à-dire, en regroupant les m à droite : $\frac{P}{P-2}(n-2) \geq m$

CQFD

Grace a ce lemme, on peut montrer qu'un graphe planaire connexe avec n sommets ne peut pas contenir plus de $3n - 6$ arêtes.

Théorème 2.5 *Tout graphe planaire connexe avec n sommets contient au plus 3(n-2) arêtes.*

Preuve : Soit G un graphe planaire connexe avec n sommets et m arêtes. Considérons une représentation R de G dans laquelle les arêtes ne se croisent pas. R est donc planaire topologique et a le même nombre de sommets et d'arêtes que G. Soit P le plus petit nombre d'arêtes sur le pourtour d'une face de R. Comme chaque face contient au moins 3 arêtes sur son pourtour, on a $P \geq 3$ et le lemme 2.4 nous montre donc que $3(n-2) \geq \frac{P}{P-2}(n-2) \geq m$.

CQFD

Théorème 2.6 *Tout graphe planaire connexe avec n sommets et sans triangle contient au plus 2(n-2) arêtes.*

Preuve : La preuve est similaire à la précédente. Étant donné un graphe planaire G connexe avec n sommets, m arêtes et sans triangle, on peut considérer une représentation planaire topologique R de G dans laquelle les arêtes ne se croisent pas. Le graphe R a le même nombre de sommets et d'arêtes que G et ne comporte pas de triangle. Le plus petit nombre P d'arêtes sur le pourtour d'une face de R vaut donc au moins 4. le lemme 2.4 valable pour tout P nous montre donc que $2(n-2) \geq \frac{P}{P-2}(n-2) \geq m$.

CQFD

Un triangle dans un graphe est un cycle de longueur 3.

Corollaire 2.2 *Tout graphe planaire connexe ayant au moins 3 sommets vérifie l'inégalité* $m \leq 3n - 6$.

Preuve : Montrons l'inégalité $m \leq 3n-6$. Considérons un graphe G simple planaire et connexe avec $n \geq 3$.

Puisque G est simple planaire et connexe et au moins 3 sommets,la distance de chaque face f du plongement de G est au moins 3, i.e $d(f) \geq 3$. Par conséquent d'après la formule d'Euler, on $2m = \sum d(f) \geq 3F(G) = 3(m - n + 2)$ avec F le nombre de faces du plongement de G. Ainsi, $m \leq 3n - 6$.

CQFD

2.2 Homomorphisme de graphes

Un homomorphisme d'un graphe G vers un graphe H est une application h de $V(G)$ vers $V(H)$ qui préserve les arêtes. En d'autres termes, pour toute arête (u, v) de G, il existe une arête $(h(u), h(v))$ dans H avec $h(u)$, l'image par h de u et $h(v)$, l'image par h de v.

Exemple : Ci-dessus pour les deux graphes $G = (V, E)$ avec $V = \{A, B, C, D, E\}$ et $E = \{a, b, c, d, e\}$ et $G^{'} = (E^{'}, V^{'})$ avec $V^{'} = \{X, Y, Z\}$ et $E^{'} = \{x, y, z\}$, il existe une application $f : G \to G^{'}$ telle que $a = (u, v) \in E \Leftrightarrow a^{'} = (f(u), f(v)) \in E^{'}$.

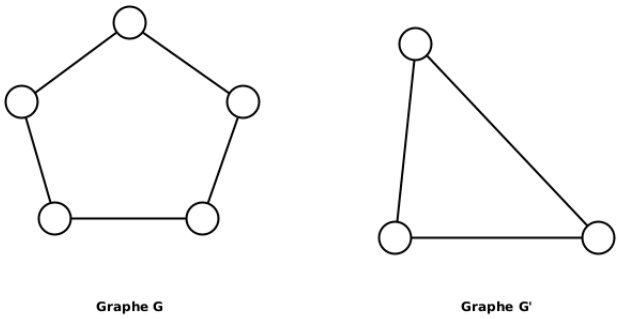

FIGURE 2.7 – Les deux graphes ci-dessus sont homéomorphes

2.3 Arbres

Définition 2.15 *Un arbre est un graphe connexe et* acyclique

Un graphe acyclique est un graphe sans cycle. Les sommets d'un arbre sont généralement appelés les noeuds de l'arbre.

Un arbre peut aussi être défini comme un graphe connexe tel que pour tout couple de sommets (u,v), il existe une unique chaîne entre u et v.

Définition 2.16 *Un* arbre enraciné *est un arbre admettant un sommet distingué appelé racine.*

On parle le plus souvent d'*arborescence* au lieu d'arbre enraciné. De manière générale on représente un arborescence de telle sorte que la racine soit le sommet le plus haut.

Notons qu'on peut orienter les arêtes d'un arbre à partir de la racine.

Définition 2.17 *Une* fôret *est un graphe G dont chaque composante connexe est un arbre, autrement dit G est un graphe acyclique.*

Lemme 2.3 *La condition nécessaire et suffisante pour qu'un graphe $G = (V,E)$ admette une racine est qu'il soit quasi-fortement connexe.*

Un graphe est dit fortement connexe si pour tout couple de sommets u,v, il existe un sommet w d'où partent à la fois un chemin allant en u et un chemin allant en v.

Théorème 2.7 *Soit G = (V, E) un graphe d'ordre $|V| = n \geq 2$; les propriétés suivantes sont équivalentes pour caractériser un arbre :*

1) G est connexe et sans cycles.

2) G est sans cycles et admet n-1 arcs.

3) G est connexe et admet n-1 arcs.
4) G est sans cycles et en ajoutant un arc, on crée un cycle (et un seul).
5) G est connexe, et si on supprime un arc quelconque, il n'est plus connexe.
6) Tout couple de sommets est relié par une chaîne et une seule.

Preuve :
(1) $\Rightarrow$ (2) , car d'après (1) G est connexe et sans cycles, donc c'est un arbre, donc il ad,et $n-1$ arcs.
(2) $\Rightarrow$ (3) , car d'après (2), G est connexe et admet $n-1$ arcs, donc c'est un arbre ; d'après le lemme 2.3, G admet un centre a.
(3) $\Rightarrow$ (4) , car la racine a de l'arbre G possède bien la propriété désirée.
(4) $\Rightarrow$ (5) , en effet, supposons que la quasi-forte connexité ne disparaisse pas lorsque l'on supprime un arc (u,v), il existe alors deux chemins élémentaires $[x,c_1,c_2,...,u]$ et $[x,d_1,d_2,...,v]$ n'utilisant pas l'arc (u,v), mais alors dans le graphe G, il existe deux chemins allant de x en v, donc il existe deux chemins allant de aa en v, ce qui contredit (4).
(5) $\Rightarrow$ (6) , le grahe G étant quasi-fortement connexe, il admet une racine a (d'après le lemme 2.3 ; on a donc $deg_G^-(u) \geq 1$ $(u \neq a)$.
D'autre part, si $deg_G^-(u) > 1$, il existerait deux arcs distincts x,y appartient à l'ensemble des voisinage entrant de u, donc deux chemins distincts allant de a à u, si on supprime l'arc x, le graphe admet toujours a comme racine, donc est toujours quasi-fortement connexe, ce qui contredit (5). On a donc $deg_G^-(u) = 1$ $(u \neq a)$.
Enfin il ne peut exister un arc allant en a, car le graphe obtenu en le supprimant admet toujours a comme racine et est quasi-fortement connexe
(1) $\Rightarrow$ (7) , le nombre d'arcs de G est $\sum_{j=1}^{n} deg_G^-(u_j) = n-1$.
G étant connexe avec $n-1$ arcs, c'est un arbre, donc il n'admet pas de cycles.
(7) $\Rightarrow$ (1) partant d'un sommet b quelconque, avec $b \neq a$, je vais parcourir le graphe G en suivant des arcs dans le sens inverse de celui de leur orientation. Je ne rencontrerai jamais deux fois le même sommet, car G est sans cycles, arrivé en un sommet u avec $u \neq a$, je peux toujours repartir, car $deg_G^- = 1$, on est donc assuré de terminer notre parcours au sommet a. Le sommet a étant alors une racine du graphe G, celui-ci est quasi-fortement connexe.

CQFD

Un graphe simple orienté $G = (X,U)$, est une arborescence s'il admet une racine (c'est-à-dire un sommet r tel qu'il existe un chemin de r à tout autre sommet) et si le graphe non orienté correspondant (on dit aussi le graphe non orienté sous-jacent) est un arbre.

Dans un graphe un *sommet pendant* (ou *feuille*) est un sommet qui n'est adjacent qu'à un seul sommet. Ainsi dans un arbre ou une arborescence un sommet de degré 1 est un sommet pendant ou une feuille.

Théorème 2.8 *Un arbre d'ordre $n \geq 2$ admet au moins deux sommets pendants.*

Pour parcourir un graphe (*resp*. un arbre) on utilise dans la plus part du temps des algorithmes de parcourt comme le *parcours en profondeur* ou le *parcours en largeur*

Parcours en profondeur : La stratégie de parcours en profondeur consiste à descendre le plus profondément dans le graphe. Dans le processus de parcours en profondeur, les arcs sont explorés à partir du sommet v découvert le plus récemment et dont on n'a pas encore exploré tous les arcs qui en partent. Dans le cas où tous les arcs de v sont explorés, l'algorithme de parcours en profondeur revient en arrière pour pouvoir explorer les arcs qui partent du sommet à partir duquel v a été découvert. Ce processus est récursif et est complet lorsque tous les sommets aient été découverts.

Parcours en largeur : Un parcours en largeur (en anglais Breadth First Search, BFS) d'un graphe $G = (V, E)$ consiste à traiter les nœuds de ce dernier par niveau de profondeur : tous les nœuds de niveau i seront traités avant tout nœud de niveau $i + 1$. Faire un parcours en largeur d'un graphe consiste à choisir un sommet, traiter ce sommet ensuite traiter les sommets à distance 1, ensuite les sommets à distance 2 et ainsi de suite. Cet algorithme de parcours construit également une *arborescence de parcours en largeur*.

Pour garder une trace de progression l'algorithme colorie chaque sommet en blanc, gris, ou noir. Les sommets sont coloriés en blanc au départ et peuvent devenir gris, puis noir lors de la progression de l'algorithme.

Un sommet est découvert pour la première fois qu'il est rencontré au cours de l'exécution de l'algorithme ; il prend alors la couleur blanche. Les sommets gris ou noir on donc été découverts, et la distinction entre gris et noir permet de garantir que le parcours se poursuit en largeur d'abord. Si par exemple $(u, v) \in E$ et si le sommet u est noir, alors le sommet v est gris ou noir ; autrement dit, tout sommet adjacent à un sommet noir a déjà été découvert.

Arbre couvrant : Un sous-graphe couvrant G' de G est un arbre couvrant G si G' est un arbre. Un arbre G' couvrant G est dit de plus courts chemins s'il existe un sommet r tel que pour tout sommet v, $dist_G(r, v) = dist_{G'}(r, v)$. Un tel arbre peut être obtenu à partir de tout graphe G en faisant un parcours (en largeur ou profondeur) sur G.

2.4 Mineur de graphe

La notion de mineur est un concept défini par Robertson et Seymour dans une série d'articles ([RS83, RS86a, RSa, RSb, RSc, RS86b, RS88, RS90b, RS90a, RS91, RS95a, RS95b, RS86c, RS95d, RS95e, RS95c, RS99, RS03, RS04b, RS04a]) en théorie des graphes. Ces articles ont été publiés dans la période du 1983-2011.

Soit un graphe G, (u, v) une arête de G. La contraction de l'arête (u, v) dans G signifie le remplacement des sommets u, v et l'arête (u, v) par un seul sommet z tel que toute arête (u, x) ou (v, x), avec x un sommet dans G, est remplacée par (z, x).

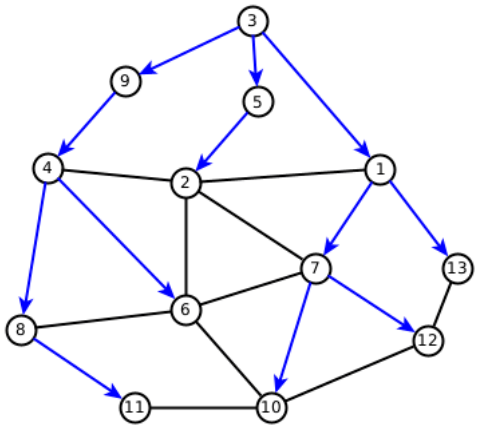

FIGURE 2.8 – Le graphe en bleu est un arbre couvrant de plus court chemin enraciné au nœud 3 du graphe rouge-noir

Un graphe H est dit mineur d'un graphe G, et on note $G > H$, si H peut être obtenu en contractant des arêtes d'un sous-graphe de G. Cela signifie que H est obtenu à partir de G en effectuant un nombre quelconque d'opérations parmi les suivantes :

1.) Suppression d'une arête (u, v) ;
2.) Suppression d'un sommet isolé u ;
3.) Contraction d'une arête (u, v).

Notons que l'une des bénéfiques de la notion de mineur est la caractérisation des classes de graphes. Par exemple dans la classe des graphes planaires, tout graphe de cette famille ne contient comme mineur ni K_5, ni $K_{3,3}$.

Théorème 2.9 *[WAG37] Un graphe G est planaire si et seulement si G ne contient ni K_5 (le graphe complet de 5 sommet) ni $K_{3,3}$ (le graphe biparti complet de 3 sommets) comme mineur.*

2.5 Algorithmique et la complexité algorithmique

2.5.1 Notion d'algorithme

Définition 2.18 *Un algorithme est une procédure de calcul bien définie qui prend en entrée un ensemble de valeurs et qui délivre en sortie un ensemble de valeurs.*

Les algorithmes peuvent être spécifiés en langage humain ou tout langage informatique. Dans ce document nous utilisons un langage proche du langage naturel.

Un algorithme est dit correct si, pour chaque instance en entrée, il se termine en produisant la bonne sortie. On dit aussi qu'un algorithme correct résout le problème donné. Par conséquent, un algorithme incorrect risque de ne pas se terminer ou se termine sur une réponse autre que celle désirée, pour certaines instances d'un problème.

Ci-dessus, on présente deux exemples d'algorithme, le parcours en largeur et le parcours en profondeur.

Parcours en largeur

L'algorithme de parcours en largeur est à la base de beaucoup d'algorithmes importants sur les graphes. Il permet d'explorer un graphe ou d'un arbre en largeur, i.e en un sommet, on visite ces voisins directs avant de passer à un autre sommet. Il permet aussi de calculer les distances de tous les nœuds depuis un nœud source. De plus, le parcours en largeur permet de déterminer si un graphe donné est connexe ou non.

```
ParcoursEnLargeur ( graphe G, sommet s) :
    f ← CreerFile()
    f.enfiler(s)
    marquer(s)
    Tant que( la file est non vide ) faire
        f ← f.defiler()
        f ← afficher(s)
            Pour tout ( voisin t de s dans G ) faire
                Si ( non marqué ) alors
                    f.enfiler(t)
                    marquer(t)
```

Parcours en profondeur

Le parcours en profondeur est un algorithme permettant d'explorer en profondeur un graphe.

L'exploration du parcours en profondeur à partir d'un sommet s donné d'un graphe se fait comme suit : on commence le parcours de s en utilisant un chemin dans le graphe jusqu'à un cul-de-sac ou qu'on atteint un sommet déjà visité. Et on revient sur nos pas jusqu'à un sommet à partir duquel on peut poursuivre le parcours. L'exploration s'arrête lorsque tous les sommets sont visités à partir de s.

Durant l'exploration, on marque les sommets afin d'éviter de reparcourir des sommets parcourus. Initialement, aucun sommet n'est marqué.

```
ParcoursEnProfondeur ( graphe G, sommet s) :
    marquer le sommet s
    afficher(s)
    Pour tout ( sommet u voisin du sommet s ) faire
        Si ( u n'est pas marqué ) alors
            ParcoursEnProfondeur(G, u)
```

2.5.2 Notion de complexité

Nous allons dans cette partie introduire la notion de complexité algorithmique, sorte de quantification de la performance d'un algorithme.

L'objectif premier d'un calcul de complexité algorithmique est de pouvoir comparer l'efficacité d'algorithmes résolvant le même problème. Dans une situation donnée, cela permet donc d'établir lequel des algorithmes disponibles est le plus optimal.

Ce type de question est primordial, car pour des données volumineuses la différence entre les durées d'exécution de deux algorithmes ayant la même finalité peut être de l'ordre de plusieurs jours.

On cherche à mesurer la complexité d'un algorithme, indépendamment de la machine, du langage de programmation, ...etc, c'est-à-dire en fonction de la taille des données que l'algorithme doit traiter.

Définition 2.19 (Complexité) *Le coût (en temps) d'un algorithme est l'ordre de grandeur du nombre d'opérations arithmétiques ou logiques que doit effectuer un algorithme pour résoudre le problème auquel il est destiné.*

Cet ordre de grandeur dépend évidemment de la taille n des données en entrées. On parle de :

- *coût constant s'il ne dépend pas de la taille de n,*
- *coût linéaire s'il est "d'ordre" n,*
- *coût quadratique s'il est "d'ordre" n^2,*
- *coût exponentielle si "l'ordre" est de la forme d'une puissance où n apparaît en exposant (e^n, par exemple).*

Il existe aussi un coût en mémoire d'un algorithme : c'est l'ordre de grandeur de la place qu'il faut réserver pour la bonne exécution de cet algorithme. Cet ordre de grandeur dépend évidemment lui aussi de la taille n des données en entrée.

On distingue trois types de complexité : la complexité au meilleur, en moyenne et au pire des cas.

Définition 2.20 (Complexité au meilleur des cas) *C'est le plus petit nombre d'opération qu'aura à exécuter l'algorithme sur un jeu de données de taille n.* $T_{min}(n) = min_{d \in D_n} T(d)$.

Définition 2.21 (Complexité en moyenne) *C'est la moyenne des complexités de l'algorithme sur des jeux de données de taille n.* $T_{moy}(n) = \sum_{d \in D_n} T(d)/|D_n|$.

Définition 2.22 (Complexité au pire cas) *C'est le plus grand nombre d'opérations qu'aura à exécuter l'algorithme sur un jeu de données de taille n.* $T_{max}(n) = max_{d \in D_n} T(d)$.

2.5.3 Notation asymptotique

Pour plus de détailles sur les notations asymptotiques développées dans cette section, nous redirigeons le lecteur vers le livre [CLRS01].

Notation Θ

La notation Θ permet de borner une fonction donnée. Pour une fonction donnée $g(n)$, on note $\Theta(g(n))$ l'ensemble des fonctions suivantes :
$\Theta(g(n)) = \{f(n) : \exists c_1, c_2 et n_0 \in \mathbb{N}_+^* \ 0 \leq f(n) \leq cg(n) \}$

Notation O

La notation O borne asymptotiquement une fonction par excès et en même temps par défaut. Pour une fonction $g(n)$, on note $O(g(n))$ (se prononce par "$grand O$ de g de n") l'ensemble des fonctions suivantes :
$O(g(n)) = \{f(n) : \exists c, n_0 \in \mathbb{N}_+^* \ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \}$
La notation O sert à majorer une fonction, à un facteur constant près. Ainsi, pour dire que une fonction f est dans l'ensemble $O(g(n))$, on écrit $f(n) = O(g(n))$

Notation Ω

Comme la notation O donne une borne supérieure asymptotique pour une fonction, la notation Ω donne une borne inférieure asymptotique. Pour une fonction $g(n)$, on note $\Omega(g(n))$ (se prononce par "$grand \Omega$ de g de n") l'ensemble des fonctions suivantes :
$\Omega(g(n)) = \{f(n) : \exists c, n_0 \in \mathbb{N}_+^* \ 0 \leq cg(n) \leq f(n) \}, \forall n \geq n_0$

Théorème 2.10 ([CLRS01]) *Pour deux fonctions quelconques* $f(n)$ *et* $g(n)$*, on a* $f(n) = \Theta(g(n))$ *si et seulement si* $f(n) = O(g(n))$ *et* $f(n) = \Omega(g(n))$.

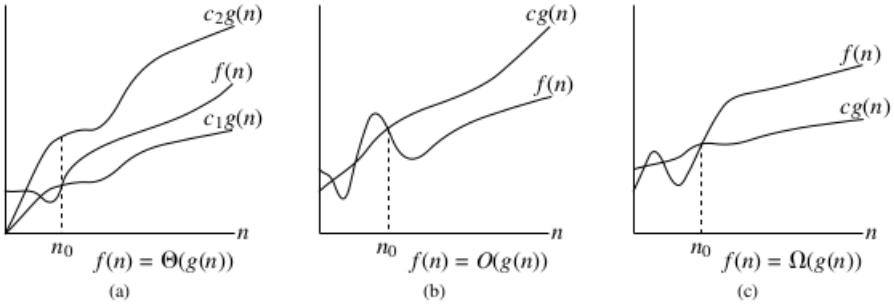

FIGURE 2.9 – Exemples de notations Θ, O et Ω. Dans chaque partie, la valeur de n_0 est la valeur minimale possible ; n'importe quelle valeur supérieure ferait aussi l'affaire. (a) La notation Θ borne une fonction entre des facteurs constants. On écrit $f(n) = \Theta(g(n))$ s'il existe des constantes positives n_0, c_1 et c_2 telles que, à droite de n_0, la valeur de $f(n)$ soit toujours comprise entre $c_1g(n)$ et $c_2g(n)$ inclus. (b) La notation O donne une borne supérieure pour une fonction à un facteur constant près. On écrit $f(n) = O(g(n))$ s'il existe des constantes positives n_0 et c telles que, à droite de n_0, la valeur de $f(n)$ soit toujours inférieure ou égale à $cg(n)$. (c) La notation V donne une borne inférieure pour une fonction à un facteur constant près. On écrit $f(n) = V(g(n))$ s'il existe des constantes positives n 0 et c telles que, à droite de n_0, la valeur de $f(n)$ soit toujours supérieure ou égale à $cg(n)$.

CHAPITRE

3

ÉTAT DE L'ART SUR LE ROUTAGE COMPACT

Dans les réseaux, l'envoi de messages entre deux nœuds est une notion basique. De ce fait un mécanisme permettant d'envoyer ces données est requit. Pour ce faire, une fonction est souvent exécutée au niveau de chaque nœud du réseau, permettant d'envoyer les données, en utilisant un certain nombre d'informations stockées dans une mémoire locale du nœud en question.

Puisque, le prix d'un routeur (i.e, d'un nœud) est proportionnel à son espace mémoire, l'idée d'optimiser la quantité d'informations devient une nécessité, d'où le routage compact.

Nous nous somme intéressés dans ce chapitre à l'état de l'art sur le routage compact. Plus précisément, le routage par intervalle et le routage compact de plus court chemin dans les arbres et les graphes planaire-extérieurs et le routage compact dans les graphes de Halin avec un facteur d'étirement.

3.1 Le routage par intervalle

Le routage par intervalle est un type de routage parmi tant d'autres. Il est utilisé pour optimiser la taille des informations de routage.

Ce type de routage est basé sur une représentation compacte de la table de routage

stockée sur chaque noeud du réseau, en regroupant sous forme d'intervalle d'adresses consécutives, c'est-à-dire l'ensemble des adresses de destination utilisant le même port.

Le principe du routage par intervalle

Nous allons illustrer le principe du routage par intervalle avec le graphe de la figure 3.1, de la même façon que Bassene dans son mémoire de magister [BAS17].

Pour un message arrivant au nœud 1, on constate que les nœuds $2, 3, 4, 5, 6, 7, 8, 9, 10$ et 11 sont atteignables à partir du nœud 1 en utilisant le port p_1. Ainsi, on peut les représentés sous forme d'intervalle d'entiers consécutifs, noté $[2;11]$.

De ce processus, on représente sous forme d'intervalle les nœuds atteignables à partir de 2 en utilisant le port p_1 par $[2;11]$. Du nœud 3, le port p_1 correspond à l'intervalle $[2;1]$ (cet intervalle est appelé un intervalle cyclique que nous allons définir ci-dessous). Du nœud 4, le port p_1 correspond à l'intervalle $[5]$ (le singleton), le port p_2 correspond à $[6;7]$, le port p_3 correspond à $[8;11]$ et le port p_4 lui correspond à l'intervalle $[1,3]$. Du nœud 5, le port p_1 correspond à l'intervalle $[4;1]$. Du nœud 6, le port p_1 correspond à l'intervalle $[1;4]$, le port p_2 correspond à $[7]$, le port p_3 correspond à $[8;10]$ et le port p_4 lui correspond à l'intervalle $[11]$. Du nœud 7, le port p_1 correspond à l'intervalle $[6;11]$. Du nœud 8, le port p_1 correspond à l'intervalle $[1;7]$, le port p_2 correspond à $[9]$, le port p_3 correspond à $[10]$ et le port p_4 lui correspond à l'intervalle $[11]$. Du nœud 9, le port p_1 correspond à l'intervalle $[8;1]$, de même que du nœud 10 le port p_1 correspond à l'intervalle $[8;1]$. Et du nœud 11, le port p_1 correspond à l'intervalle $[8;10]$, le port p_2 correspond à $[1;3]$, le port p_3 correspond à $[4;5]$ et le port p_4 lui correspond à l'intervalle $[6;11]$.

On a ci-après, la définition formelle d'un intervalle.

Définition 3.1 *Un intervalle $[a;b]$ d'entiers, tel que $a,b \in \{0,...,n\}$, est l'ensemble des entiers i tels que :*

— *$a \leq i \leq b$ si $a \leq b$, l'intervalle est alors dit* intervalle linéaire.

— *$a \leq i \leq n$ ou $1 \leq i \leq b$ si $a \geq b$, l'intervalle est alors dit* intervalle cyclique.

Si $a = b$, alors on dénote par $[a]$ l'intervalle $[a,a]$. $]a,b]$ (resp, $[a,b[$) représente l'intervalle $[a,b]-\{a\}$ (resp, $[a,b]-\{b\}$.

Pour un nœud x, l'ensemble des destinations sont regroupées selon les ports de x et, pour chaque port i l'ensemble des destinations utilisant ce port sont regroupées sous forme d'intervalle d'entiers. Alors, la table de routage de x est telle que la case numéro i contient tous les intervalles d'entiers identifiant les destinations passant par le port i. Chaque case de cette table peut alors contenir plusieurs intervalles.

Il est important de rappeler que pour stocker un intervalle, on a besoin juste de stocker les deux entiers représentant les bornes. Nous avons alors besoin de $O(\log n)$ si les bornes sont $\leq n$. Il est alors clair que plus on a d'intervalles, pour chaque case de la table, plus la taille de la table de routage sera grande.

L'enjeu maintenant est de trouver un moyen de faire de telle sorte que le nombre d'intervalles dans chaque case du tableau soit le plus petit possible.Les intervalles sont, cependant, définis à partir de la numérotation des sommets du graphe. Par conséquent, minimiser le nombre d'intervalle reviendra, inévitablement, à trouver un mécanisme pour avoir une bonne numérotation.

La fonction de numérotation permettant de numéroter les sommets est appelée un schéma de numérotage (ou d'étiquetage). Ainsi, on peut dire que l'enjeu est donc de trouver un schéma d'étiquetage permettant de rendre compact le nombre d'intervalles dans chaque case du tableau.

Si dans un graphe donné on trouve une numérotation qui permet de faire un schéma de routage de telle sorte que chaque case de notre tableau contienne au plus k intervalles, on dira qu'on a un routage par $k-$*intervalle* ($k-$*interval* routing). Dans ce cas, le nombre total d'intervalle dans un nœud u va être inférieur ou égale à $deg(u)\times k$, avec k le nombre maximum d'intervalles dans chaque case et $deg(u)$ sera le nombre de ports (ou de liens) liés au nœud u. Par conséquent, un schéma de routage sera de taille $O(deg(u) \times k)$.

Le principe du routage par intervalle sur un graphe $G = (V, E)$ est tel que, en chaque nœud $u \in V(G)$, la fonction de routage vérifie :

— si l'adresse de destination v de l'en-tête du message correspond à u, le message est arrivé et la fonction s'arrête.

— sinon le message est relayé suivant une arrête étiquetée par l'intervalle $I = [a, b]$ tel que $v \in I$.

Notons que vérifier si $v \in [a, b]$ revient à tester si v est compris entre a et b. C'est une opération en temps contant, c'est à dire $O(1)$.

Sur la figure 3.1, le processus de routage d'un message arrivant au nœud 1 avec une entête contenant l'adresse du nœud 7 est le suivant. Lorsque le nœud 1 reçoit le message, il récupère d abord l'entête du message et teste si le message lui est destiné, si c'est le cas le message est à destination et le processus s'arrête ; et puisque dans notre exemple le message est destiné au nœud 7, le nœud 1 cherche dans sa table de routage le port de sortie correspondant à l'intervalle contenant le nœud 7, ce port est p_1. Le message sera envoyé de ce port et arrive au nœud 2 par le port p_1 lié à 2. Le nœud va faire le même teste et va envoyé le message au nœud 4, puis 6 et enfin au nœud 7, arrivé au nœud 7, ce dernier va tester si le message lui est destiné et puisque le message l'est, ce dernier sera envoyé par le port p_0 qui correspond au port lié à la machine locale de 7.

Le schéma de routage par intervalle a été introduite pour la première fois par N. Santoro et R. Khatib [SK85], ils précisent qu'un schéma de routage par intervalle nécessite que chaque ensemble de destination soit représenté en un seul intervalle. Une étude plus générale sur ce type de routage a été faite par Cyril Gavoille [GAV00], qui donna une définition formelle du routage par intervalle et la présentation de quelques notions de base. Il est nécessaire de spécifier que le routage par intervalle ne marche pas souvent pour tous les types de graphes. Dans [GAV00] C. Gavoille donna un ensemble de familles de graphes qui n'admettent pas ce type de routage.

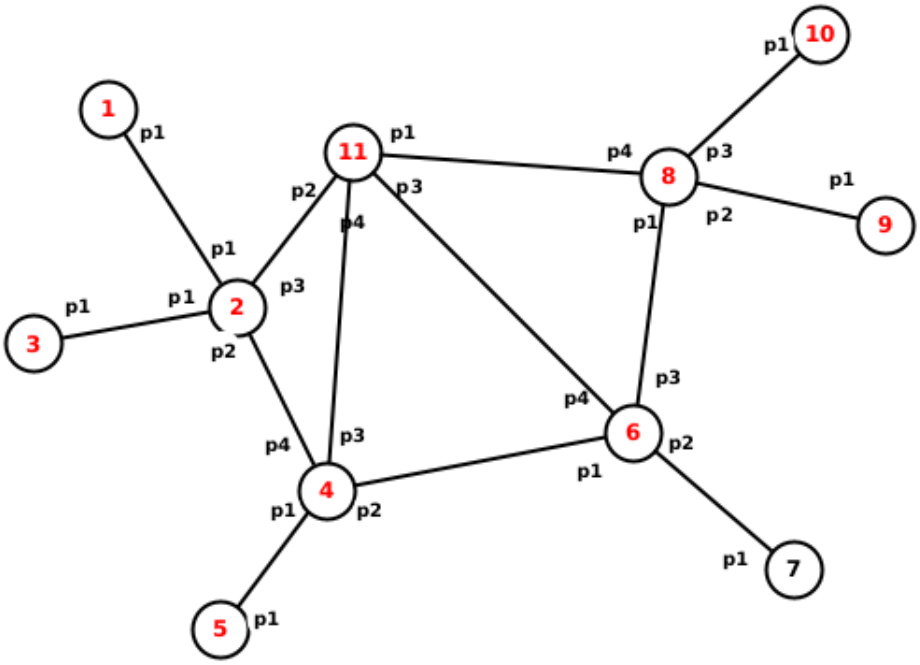

FIGURE 3.1 – exemple de schéma de numérotation pour illustrer le routage par intervalle

3.2 Le routage compact de plus court chemin

De nombreux problèmes de recherche opérationnel reviennent à déterminer l'existence d'un plus court chemin. Par exemple dans les problèmes de routage, on cherche souvent à acheminer des paquets (messages) en utilisant le chemin le plus court possible.

Il est aussi important de router des messages par de plus court chemin en utilisant le moins d'espace mémoire possible et en temps raisonnable.

De nombreux et intéressants résultats dans la recherche de schéma de routage compact de plus court chemin ont été faites, par exemple, Fraigniaud et Gavoille dans leurs article [FG02] ont développés un schéma de routage compact de plus court chemin dans les arbres. Dieng et Gavoille ont démontrés que les graphes planaires extérieurs non valués admettent un schéma de routage compact de plus court chemin [DG11].

Dans cette section nous avons fait un état de l'art sur le routage compact de plus court chemin.

3.2.1 Routage compact dans les arbres

Dans cette section nous allons parler d'un résultat de Fraigniaud et Gavoille [FG02] portant sur le routage compact dans les arbres. Les auteurs, dans leur papier ont développé un schéma de routage compact de plus court chemin dans les arbres.

La problématique consiste à router des messages par de plus courts chemins en utilisant des tables de routage compact. Les auteurs ont prouvé qu'un réseau en arbre de n nœuds, supporte un schéma de routage compact de plus court chemin avec des adresses, des en-têtes, des tables de routage de $O(\log n)$ bits. De plus, la fonction de routage proposée est en un temps de calcul constant (i.e, $O(1)$) [FG02].

Pour construire leur schéma de routage, les auteurs considèrent un arbre T de n nœuds, enraciné en un nœud $r \in T$. Pour tout nœuds u, ils ont noté T_u le sous-arbre de T enraciné en u induit par u et tous ces descendants dans T. Le nombre de nœuds de T_u, noté $w(u)$, est appelé le poids de u.

Concernant l'étiquetage des nœuds, ils ont dans leur papier utilisés un *algorithme de parcours en profondeur* (ou DFS pour Depth First Search, en anglais) modifié permettant d'affecter à chaque nœud un entier dans [1, n].

Un algorithme de parcours en profondeur (DFS) consiste à descendre le plus profondément possible dans le graphe, comme son nom l'indique, pour exploiter tous les sommets afin de déterminer un arbre couvrant le graphe d'entrée. Lors de son exécution les arcs (ou arêtes) sont explorés à partir d'un sommet u découvert le plus récemment et qu'on a pas encore exploré tous les arcs (arêtes) qui en partent. Lorsqu'on parvient

à explorer tous les arcs de u, l'algorithme revient en arrière sur le dernier sommet où on peut explorer ces arcs. Ce processus se répète jusqu'à ce que tous les sommets atteignable à partir du sommet initial aient été découverts. S'il reste des sommet qui ne sont pas atteignable à partir du sommet initial, on choisit un sommet parmi ces derniers comme nouveau sommet initial et appliqué le même processus jusqu'à ce que tous les sommets du graphe global aient été découverts.

Ainsi, si le graphe initial est connexe alors le nouveau graphe constitué par les arcs qui ont été explorés forme un arbre, si le graphe initial n'est pas connexe alors on obtient une *forêt* d'arbres.

Ci-dessous une implémentation récursive de parcours en profondeur d'un graphe G :

EXPLORER((G, s)) **:**
marquer le sommet s
afficher(s)
PourTout (sommet u voisin du sommet s) ***Faire***
Si (u n'est pas marqué) ***Alors***
EXPLORER(G, u)
FinSi
FinPour

DFS((G)) **:**
PourTout (sommet s du graphe G ***Faire***
Si (s n'est pas marqué) ***Alors***
EXPLORER(G, s)
FinSi
FinPour

On peut dire que **EXPLORER** est un DFS qui retourne un unique arbre alors que **DFS** retourne plusieurs arbre c'est-à-dire chaque sommet du graphe d'entrée est une racine d'un arbre.

Ainsi, le DFS modifié des auteurs est décrit comme suit :

Soit T un arbre enraciné en un nœud r. On affecte au nœud r l'entier $id(r) = 1$ l'identifiant de r, et avec un parcourt en DFS on visite le sous-arbre le plus large (i.e le sous-arbre qui a le plus de nœuds), soit T_{u_1}, et on affecte à u_1 le numéro précédent plus 1, à partir de u_1, on applique le même procédé. Puis le second sous-arbre le plus lourd et ainsi de suite.

Ce procédé donne un étiquetage dans lequel $id(r) = 1$, et pour tout nœud intermédiaire $z \in T$ et deux enfants u, v de z, si $id(u) \leq id(v)$ alors $w(u) \geq w(v)$.

Voir la figure 3.2, pour un exemple de numérotation des nœuds d'un arbre à partir de ce procédé.

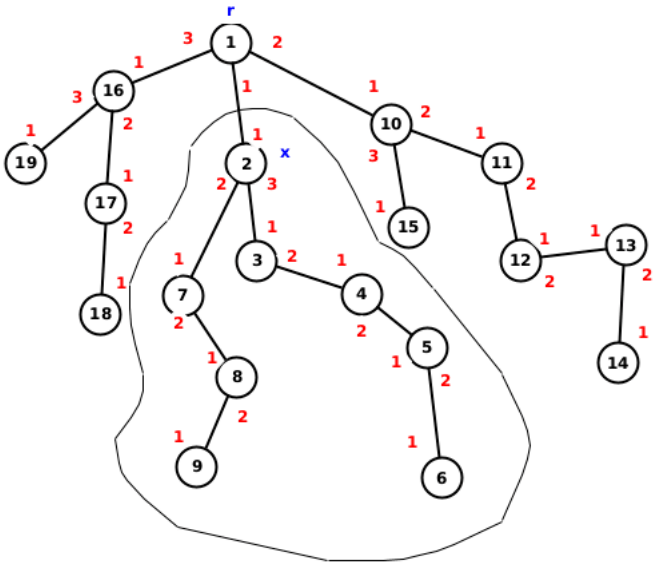

FIGURE 3.2 – Arbre T enraciner en r. En noir numérotation suivant le DFS, en rouge numérotation des ports sur chaque sommet de T

Pour l'assignation des numéros de *port* ou la numérotation des arêtes, les auteurs définissent d'abord la notion de $rank$ d'un nœud, qui, pour z un enfant de v dans l'arbre, $rank$ de z, noté $rank(v,z)$ est le nombre d'enfants de v qui ont des numéros d'identifiant au plus celui de z.

L'étiquetage des arêtes ou la numérotation des ports se fait comme suit :

port(u) **:**
    Si v *est parent de* u ***alors***
        retourner 1
    FinSi
    Sinon
        retourner $1 + rank(u,v)$
    FinSinon

L'étiquetage des ports se termine avec assignation de numéro de port au localhost (ou l'hôte local). Le localhost est le nom utilisé pour se référer à l'ordinateur local. Ainsi pour tout nœud u, on donne le numéro 0 au lien le connectant à son localhost. Voir encore la figure 3.2, pour la numérotation des ports.

Pour définir la structure des adresses et des tables de routage, les auteurs définissent dans leur papier la notion de $path(u)$, pour un sommet u, comme étant la séquence de

$ranks$ pour tous les nœud rencontrés sur le chemin de r à u tels que $path(r) = ()$ la séquence vide.

De manière beaucoup plus précise, si le chemin de r à u est la séquence de nœuds $r = u_0, u_1, ..., u_t = u$ alors $path(u) = (rank(u_0, u_1), ..., rank(u_{t-1}, u_t))$.

Et le $clean - path$ de u, $cpath(u)$, est défini comme étant la séquence obtenue de $path(u)$ en supprimant tous les ranks qui sont égaux à 1. Les auteurs dénotent $|cpath(u)|$ la longueur de la séquence $cpath(u)$. Dans le cas où $path(u)$ contient que des 1, alors $|cpath(u)| = 0$.

Ainsi pour tout nœud u, son adresse est :

$$\boxed{l(u) =< id(u), cpath(u) >}$$

et la structure de donnée stockée en tout nœud u, c'est-à-dire la table de routage est :

$$\boxed{table(u) =< id(u), w(u), w_1(u), |cpath(u)| >}$$

Où $w_1(u)$ est le poids de l'enfant de u le plus lourd (i.e, le poids $w(z)$ tels que $rank(u, z) = 1$). Et si u est une feuille alors $w_1(u) = 0$.

Dans leur papier ils ont aussi montrés le lemme 3.1 ci-dessous qui rend commode le calcule de la complexité en terme d'espace de stockage des tables de routage au niveau de chaque nœud.

Lemme 3.1 *[FG02]*

1) Si u est parent de v, alors rank(u,v) ≤ w(u) / w(v).

2) Si cpath(v) = $(p_1, ..., p_k)$, alors $\prod_{i=1}^{k} p_i \leq n/w(v)$, et $k \leq \log(n/w(v))$.

Pour router un message dans le réseaux, ils ont donnés une fonction de routage direct, c'est-à-dire une fonction qui ne change pas l'en-tête du message tout au long de la transmission et que la seule information stockée dans l'en-tête du message est l'adresse de destination. Cette propriété est aussi un facteur de performance très important. En effet, recalculer l'entête de destination peut avoir une influence sur le temps de calcul de la fonction de routage.

Donc si un nœud u reçoit un message d'en-tête $h(v) = l(v)$ la fonction de routage **ROUTE**$(u, l(u))$ décide vers quel port le message doit-il être retransmit si u n'est pas le destinataire. Cette fonction est définie comme suite :

ROUTE$(u, l(v))$ **:**
  Si $(id(v) = id(u)$ **alors**
    retourner 0
  Sinon Si $(id(v) < id(u)$ ou $id(v) \geq id(u) + w(u))$ **alors**
    retourner 1
  Sinon Si $(id(u) < id(v) \leq id(u) + w_1(u))$ **alors**
    retourner $1 + b$
  Sinon
    retourner $p + b$ où $p = (|cpath(u)| + 1)^{ieme}$ élément de $cpath(v)$
  FinSi

Notons que $b = 0$ si u est racine de l'arbre et $b = 1$ dans le cas contraire.

Dans le soucis de montrer que la fonction de routage route par de plus court chemin, ils ont montrés le lemme suivant.

Lemme 3.2 *[FG02] Pour toute paire de sommets u, v, la fonction de routage décrit par* ***ROUTE*** *route tout message de u à v suivant un plus court chemin.*

En effet lorsqu'un nœud u reçoit un message d'adresse $l(v)$ la fonction **ROUTE** consulte l'en-tête du message pour décider lequel des ports le message doit-il être retransmit, et pour ce faire elle dispose de trois cas de figure suivant le contenu de l'en-tête du message (soit le message est envoyé au localhost de u, soit au parent de u, soit aux descendants de u).

Après vérification de l'en-tête du message.

¤) Si $id(u) = id(v)$, alors la fonction retourne la réponse attendue, puisque le port 0 correspond au port qui relie u à son localhost.

¤) Si $id(v) < id(u)$ ou $id(v) \geq id(u) + w(u)$, alors v n'est pas un descendant de u, donc le message est retransmit au parent de u, d'où le port 1.

¤) Si $id(u) < id(v) \leq id(u) + w_1(u)$, alors v est un descendant de u et puisque $id(v) \leq id(u) + w_1(u)$, ainsi le message est retransmit sur le port de l'enfant de u le plus lourd puisque la numérotation des nœuds commence par numéroté tous les nœuds du sous-arbre le plus lourd, donc la fonction route par $1 + b$ (i.e, 1 si $id(u) = 1$ ou 2 dans le cas contraire).

¤) Si $id(u) + w_1(u) < id(v) < id(u) + w(u)$ alors v est un descendant de u et la séquence $path(u)$ est un préfixe de $path(v)$ ce qui implique que $cpath(u)$ est aussi un préfixe de $cpath(v)$.
Le message est retransmit à un enfant z de u tel que $path(z) = (path(u), rank(u, z))$, et puisque z n'est pas l'enfant le plus lourd de u donc $rank(u, z) \neq 1$. Puisque $rank(u, z) \neq 1$ et $cpath(u)$ est un préfixe de $cpath(v)$ dans ce cas $rank(u, z)$ est contenu dans la séquence de $cpath(v)$. Il est évident que $rank(u, z)$ est le premier élément de $cpath(v)$ après tous les éléments en commun avec $cpath(u)$.

Cependant $(i+1)-eme$ élément de $cpath(v)$ correspond à z, où i est le nombre d'éléments de $cpath(u)$. Ainsi la fonction route par de plus court chemin.

Pour l'implémentation de l'espace mémoire nécessaire au stockage des informations de routage à chaque nœud, les auteurs ont montré le lemme suivant.

Lemme 3.3 *Pour tout sommet u, l'adresse de* $l(u)$ *est de taille plus petit que* $5\log n+1$ *bits, et sa structure de donnée locale* $table(u)$ *est de taille plus petit que* $3\log+\log\log n+4$ *bits.*

En effet, on représente l'identifiant d'un nœud u, $id(u)$ par une chaîne binaire de longueur $\lceil\log n\rceil < (\log n)+1$ bits. Pour le codage de la séquence $cpath(u) = (p_1,...,p_k)$, ils considère l_i la longueur de la décomposition binaire de $q_i = p_i - 2$ $(q_i \geq 0)$; donc $l_i = max(\{\lceil\log(p_i-1)\rceil, 1\}$, en fait, ils représentent la séquence $cpath(u)$ par deux chaînes binaire, appelé A et B, chaque chaîne est de longueur $L = \sum_{i=1}^{k} l_i$.

La chaîne A est composée de la concaténation des représentations binaire des q_i sur l_i bits. Et la chaîne B est un tableau de bits qui indique le début de chaque champs représentant un q_i dans A, ce qui est faisable puisque $l_i \geq 1$.

Puisque $p_i \geq 2$ alors $l_i < (\log p_i)+1 \Rightarrow L < \log(\prod_{i=1}^{k} p_i) + k$. En utilisant le lemme 3.1, $L < \log(n/w(u)) + \log(n/w(u) \leq 2\log n$; donc l'identifiant d'un nœud u, $id(u)$ peut être représenté par au plus $5\log(n)+1$ bits.

Et la table de routage d'un nœud u, $table(u) =< id(u), w(u), w_1(u),$

$|cpath(u)| >$ est représentée par une chaîne binaire de longueur au plus $3\lceil\log n\rceil + \lceil\log k\rceil < 3\log n + \log\log n + 4$ bits (avec $k = |cpath(u)|$ et $k \leq \log n$ par le lemme 3.1.

Avec l'utilisation du lemme 3.4 montré par Kalmar en 1930, qui peut être retrouver dans les travaux de Warlimont et Tan [WT93], les auteurs ont améliorer le codage de $cpath(u)$.

Lemme 3.4 *Soit Z(n) le nombre de séquences d'entiers ordonnés* $p_1, p_2, ...$ *tels que* $p_i \geq 2$ *et* $\prod_{i\geq 1} p_i \leq n$. *Alors Z(n)* $\sim n^{\rho}/\kappa$, *avec* $\kappa = -\rho.\zeta^{1}(\rho)$ *ou* ρ *est l'unique solution réel de l'équation* $\zeta(\rho) = 2$ *et* ζ *est la fonction Zéta de Riemann (*$\rho \approx 1,7286$ *et* $\kappa \approx 3,1429$

A partir du lemme 3.4, tout clean-path peut être codé en utilisant au plus $\lceil\log Z(n)\rceil \leq \rho\log n < 1,8\log n$.

3.2.2 Routage compact dans les graphes planaire-extérieurs

A partir des travaux de Fraigniaud et Gavoille [FG02], Dieng et Gavoille [DG11] ont développés un schéma de routage compact de plus courts chemins pour les graphes planaires extérieurs non valué. Ces résultat est une extension du résultat de Fraigniaud et al. [FG02] à une famille de graphe plus large qui est celle des graphes graphes planaires extérieurs. Les auteurs ont donnés une généralisation de ce résultat à une autre famille plus grande, celle des graphes (k,r)-$constellations$.

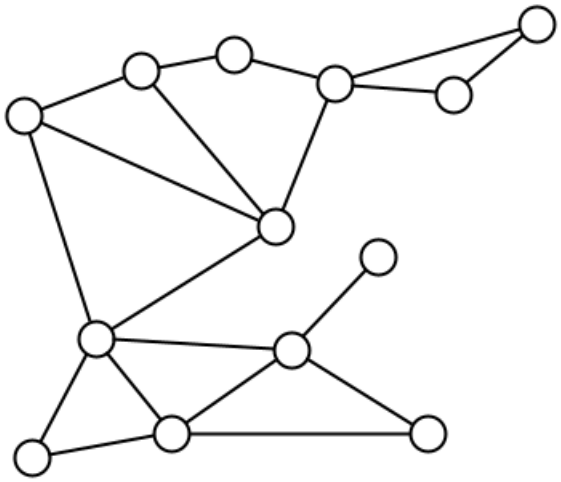

FIGURE 3.3 – Exemple de graphe planaires extérieurs

Définition 3.2 *Un graphe $G = (V, E)$ est planaire extérieur s'il est planaire et que tout sommets $u \in V$ est sur la face extérieur.*

Pour construire le schéma de routage des graphes planaires extérieurs non valués, qui donne le théorème 3.1, les auteurs considèrent un graphe planaire extérieur G a n sommets. Ainsi pour un plongement de G dans un plan de façon à ce que les sommets bordent la face extérieure, ils numérote en utilisant un parcourt bordant cette même face, les sommets du graphe G par des entier dans $[1, n]$.

Ensuite ils posent $\mathfrak{T}$ un arbre couvrant G de plus courts chemins, enraciné en un sommet r_0 choisi arbitrairement.

Ainsi, pour un sommet u, ils construisent :

¤ Une table de routage **Table**-$\mathfrak{T}_u$ pour les sommets de $\mathfrak{T}_u$, construit à partir de $\mathfrak{T}$, où $\mathfrak{T}_u$ est un sous-arbre de $\mathfrak{T}$ enraciné en u. C'est-à-dire pour toute destination $v \in \mathfrak{T}_u$, **Table**-$\mathfrak{T}_u[v]$ donne le numéro de port de l'arête menant vers v.

¤ Une table de routage **Table**-$\mathfrak{H}_u$ pour les sommets $\mathfrak{H}_u = G/\mathfrak{T}_u$, construite à partir de T^u en compactant sous forme de segments l'ensemble des destinations utilisant le même port, où T^u est un arbre couvrant G enraciné en u. C'est-à-dire pour toute destination $v \in \mathfrak{H}_u$, **Table**-$\mathfrak{H}_u[v]$ donne le numéro de port de l'arête menant vers v.

Pour router un message d'un sommet à un autre, les auteurs proposent la fonction de routage **Route** ci-dessous, qui termine leur schéma de routage pour les graphes pla-

naires extérieurs non valués.

Route(u, v) **:**

Si $(v \in \mathfrak{T}_u)$ **alors**
retourner Table-$\mathfrak{T}_u[v]$
Sinon
retourner Table-$\mathfrak{H}_u[v]$

Les auteurs de [DG11] ont montrés le théorème ci-dessous, qui valide que le schéma proposé est compact et de plus courts chemins.

Théorème 3.1 *[DG11] Tout graphe planaire-extérieur connexe non valué G à n sommets admet un schéma de routage de plus courts chemins avec des adresses, des en-têtes et des tables de routage de $O(\log n)$ bits, la latence est constante.*

En effet, pour chaque sommet u, la table de routage de u est constituée des tables **Table-**$\mathfrak{T}_u$ et **Table-**$\mathfrak{H}_u$. La table de routage de u, **Table-**$\mathfrak{T}_u$, construite à partir d'un schéma de routage compact dans $\mathfrak{T}$. La table $\mathfrak{T}$ peut être implémentée par une structure de données de taille $O(\log n)$ bits.

La table de routage **Table-**$\mathfrak{H}_u$ est construite en compactant sous forme de segments l'ensemble des destinations de $\mathfrak{H}_u$ passant par le même port. C'est-à-dire, pour chaque port de sortie p de u, il existe une entrée unique dans **Table-**$\mathfrak{H}_u$ qui contient un ou plusieurs segments de destinations utilisant le port p.

Soit $\Omega(u)$ l'ensemble tel que $\forall v \in H_u, \exists w \in \Omega(u)$ tel que w est un sommet sur un plus court chemin entre u et v dans G. Dieng et Gavoille ont montrés le théorème suivant.

Lemme 3.5 *Pour tout sommet $u \in G$, on a $|\Omega(u)| \leq 6$.*

Ce lemme montre que le nombre d'entrées de la table **Table-**$\mathfrak{H}_u$, i.e $\Omega(u)$ est au plus 6. Et puisque, chaque segment peut être implémenté par une structure de taille $2\log n$ bits. Ainsi, la table **Table-**$\mathfrak{H}_u$ est de taille au plus $12\log n$ bits.

En effet, il existe un résultat de (Frederickson et al., 1988) qui dit que si G est un graphe planaire-extérieur à n sommets numérotés consécutivement suivant un parcours bordant de la face extérieure, alors pour tout arbre couvrant $\mathcal{T}$ de racine u (en particulier d'arbre de plus courts chemins), et tout voisin v de u, l'ensemble des sommets du sous-arbre $\mathcal{T}_v$ peut être compacté en un seul segment.

Dieng et Gavoille ont étendus ce résultat sur des graphes beaucoup plus large, les graphes *(k,r)-constellations*.

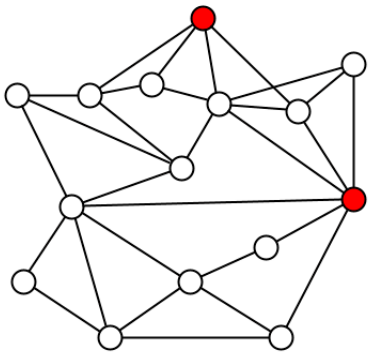

FIGURE 3.4 – Une (2,6)-constellation

Définition 3.3 *Un graphe (k,r)-constellations est un graphe sans mineur $K_{2,r}$, dont chacune de ces composantes bi-connexes peut être rendue planaire-extérieur en supprimant au plus k sommets.*

Cependant le schéma de routage développé pour les graphes *(k,r)-constellations* à partir du schéma de routage pour les graphes planaires extérieurs vérifie le théorème ci-dessous, démontré par Dieng et Gavoille.

Théorème 3.2 *[DG11] Toute (k, r)-constellation connexe non valué G à n sommets admet un schéma de routage de plus courts chemins avec des adresses et des en-têtes de $O((k+1)\log n)$ bits et des tables de routage de $O((kr+1)\log n)$ bits. La latence est de $O(\log k + \log r)$.*

3.3 Routage compact dans les graphes de Halin avec facteur d'étirement

Il arrive souvent que nous ne puissions pas en même temps avoir un routage compact de plus court chemin. Il a un compromis entre la longueur du chemin produit par l'algorithme et la quantité de mémoire à stocker dans chaque nœud. En effet, si on prend une table de routage de taille n telle que l'entrée numéro i contient le numéro de port par lequel il faut passer pour aller en i, alors on aurait un routage de plus court chemin et c'est possible pour n'importe qu'elle graphe. Cependant, la table de routage sera de taille $O(n\log n)$ et le temps de latence de la fonction de routage sera en $O(n)$. C'est donc, la raison qui fait que dès fois, proposer un routage avec un facteur d'étirement raisonnable avec des table de routage compact est intéressant.

Dans cette partie, nous allons parler de schéma de routage pour les graphes de Halin avec facteur d'étirement d'au plus 2. Ce travail proposé par Avéwé Basséne dans le

cadre de son mémoire de magister [BAS17], présente un schéma de routage compact pour les graphes de Halin avec facteur d'étirement d'au plus 2, en se basant sur les travaux de P. Fraigniaud et C. Gavoille [FG02]. La méthode utilisée par Avéwé Basséne est expliquée ci-dessous.

L'auteur considère un graphe de Halin $H = (V, E)$, X le graphe constitué par l'ensemble des sommets de la face extérieur de H et T_c l'arbre couvrant de H enraciné au centre c du graphe $H \setminus X$. Il nota r_c le rayon du graphe.

Pour définir une étiquetage des sommets l'auteur dans [BAS17], numérote les sommets de X par des entiers de 1 à $|X|$. Et les sommets de T_c sont numérotés par des entiers de $|X| + 1$ à n, en utilisant le parcours en profondeur (DFS) modifié de P. Fraigniaud et C. Gavoille dans [FG02].

L'adresse de chaque nœud u du graphe contient un identifiant $id(u)$, le $cpath(u)$, une valeur booléen permettant de dire si *oui* ou *non* le nœud u est voisin direct d'un sommet de X. De plus l'adresse de u contient un ensemble d'informations sur ces voisins directs et d'un tableau à deux entrées $valInt(u)$ stockant le minimum et le maximum de ces descendants dans X.

L'algorithme de routage **ROUTEab** définit par Avéwé Basséne [BAS17] est décrit comme suit :

Lorsqu'un nœud u reçoit un message, l'algorithme de routage **ROUTEab** récupère dans l'entête du message, l'adresse du nœud destinataire, soit v le nœud destinataire et l'adresse du nœud courant c'est-à-dire le nœud u. Ainsi que les informations de ces voisins et des valeurs $u.valInt[i]$ et $v.valInt[i]$ avec $i \in \{0, 1\}$.

ROUTEab(u, v) **:**

Si $(u, v \in X)$ ***alors***
    execute(***Route1(u,v)***)
SinonSi $(u \in X et v \notin X)$ ***alors***
    execute(***Route2(u,v)***)
SinonSi $(u \notin X v \in X)$ ***alors***
    execute(***Route3(u,v)***)
SinonSi $(u, v \notin X)$ ***alors***
    execute(***Route4(u,v)***)

L'algorithme de routage ci-dessus décrit par Avéwé Basséne [BAS17] est élémentaire son originalité vient de son explication. En effet, pour le routage il existe trois cas de figure suivant la position de u et de v dans le graphe.

Notons que **ROUTEab** fait appelle à la fonction *execute* pour exécuter soit ***Route1***, soit ***Route2***, soit ***Route3*** ou soit ***Route4*** suivant la position des deux nœuds (le nœud

source et le nœud destinataire). Et la fonction *estVoisin* qui prend en entrée un nœud $u \in H$ et l'ensemble X puis retourne vrai si $u \in X$ est faux dans le cas contraire.

Dans ce qui suit nous allons présenté les quatre cas de figure énumérés par Bassené [BAS17].

3.3.1 Cas où les nœuds source et destinataire sont dans X ($u, v \in X$)

ROUTE1(u, v) **:**

Si $(id(u) > id(v))$ ***alors***
$v \leftarrow min(|X| - u + v, u - v)$
Sinon
$v \leftarrow min(|X| + u - v, |u - v|)$ ***alors***
Si $(2r_c \geq v)$ ***alors***
$RouteX(v)$
Sinon
$RouteT_c(v)$

Lorsque $u, v \in X$, l'algorithme **ROUTEab** au niveau du nœud source u route un message de u à v en utilisant la fonction *execute* qui exécute **Route1**. La fonction **Route1** en exécutant récupérer l'en-tête du message. Ainsi, elle dispose des informations sur les valeurs de $v.ValInt[i]$, $i \in [0, 1]$, qui constituent le minimum et le maximum des descendants de v dans X.

L'idée d'envoyer un message de u à v est de trouver la longueur minimale ν de part et d'autre des arêtes dans X incidentes à u. De comparer cette longueur avec celle dans T_c reliant u et v. Notons que la valeur ν est donnée par $min(|X| - u + v, u - v)$ si $id(u) > id(v)$ ou $min(|X| + u - v, |u - v|)$ dans le cas contraire. Ainsi, si ν est plus petit ou égal à la longueur dans T_c alors le message est envoyé dans X sinon il est envoyé dans l'arbre T_c suivant le chemin reliant u et v qui est au plus $2r_c$.

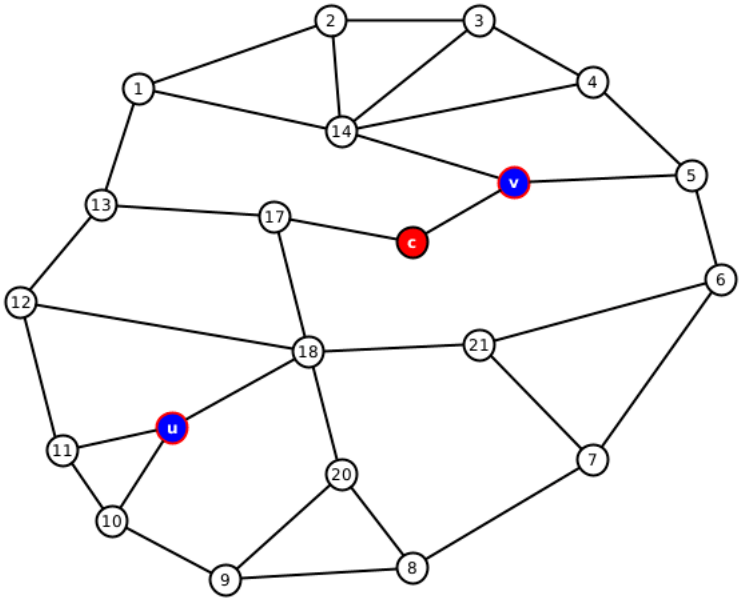

FIGURE 3.5 – Une disposition des noeuds u et v pour le cas **Route1**

3.3.2 Cas où le nœud source est dans X et non le nœud destinataire ($u \in X$ **et** $v \notin X$)

ROUTE2(u,v) **:**

Si $(estVoisin(v,X))$ ***alors***
  Si $(v.valInt[0] > u)$ ***alors***
    $v \leftarrow min(v.valInt[0]-u, |X|+u-v.valInt[1])$
  Sinon
    $v \leftarrow min(|v.valInt[1]-u|, |X|+u-v.valInt[0])$
  Si $(v > 2r_c - 1)$ ***alors***
    $RouteT_c(v)$
  Sinon
    $RouteX(v)$
Si $(\neg estVoisin(v,X))$ ***alors***
  Si $(v.valInt[0] > u)$ ***alors***
    $v \leftarrow min(v.valInt[0]-u, |X|+u-v.valInt[1])$

Sinon Si $(v.valInt[0] < u < v.valInt[1])$ ***alors***
    $\nu \leftarrow min(v.valInt[1] - u, u - v.valInt[0])$
Sinon
    $\nu \leftarrow min(v.valInt[0] - u, |X| + u - v.valInt[1])$
Si $(\nu > 2r_c - 1)$ ***alors***
    $RouteT_c(v)$
Sinon
    $RouteX(v)$

Lorsque le nœud source u est dans X et le nœud destinataire dans T_c, au niveau du nœud source u, la fonction *execute* de **ROUTEab** exécute la fonction **Route2**. Cette dernière, de la même manière que **Route1** récupère les informations contenues dans l'entête du message. Puis détermine le minimum des longueurs des chemins entre u et le plus petit descendant de v dans X (i.e, le chemin reliant u et $v.valInt[0]$) et le plus grand descendant de v dans X (i.e, le chemin reliant u et $v.valInt[1]$), de part et d'autre de la face extérieure X. Cette valeur est stockée dans ν.

Et puisque $v \in T_c$ alors il peut être atteint depuis un nœud $u \in X$. Soit en passant par le chemin ν relié à une des arêtes adjacentes à u et appartenant à X, soit en passant par le chemin relié à l'arête adjacente à u et appartenant à T_c noté ρ. Le fait que $v \in T_c$ donne deux possibilités : soit v est voisin direct d'un nœud de X, soit il ne l'est pas. La vérification est faite avec l'utilisation de la fonction *estVoisin* appelée par l'algorithme de routage **ROUTEab** dans son exécution (donc de l'appelle de la fonction **Route2**). Par conséquent, l'idée de router de u à v revient à déterminer par lequel des deux chemins faire transiter le message.

Ainsi, pour ce faire, il suffit de comparer ν et $2r_c - 1$ lorsque v est voisin direct d'un nœud de X ou de comparer ν et $2r_c - 2$ lorsque v n'est pas voisin direct d'un nœud de X. Dans le cas où v est un voisin direct d'un nœud de X donc si $\nu > 2r_c - 1$ alors le chemin ρ est le plus court chemin puisque aucune longueur ne peut dépasser $2r_c - 1$. Dans le cas contraire où v n'est pas voisin direct d'un nœud de X, donc si $\nu > 2r_c - 2$ alors le chemin ρ est le plus court chemin puisque aucune longueur ne peut dépasser $2r_c - 1$. Et si $\nu < 2r_c - 1$ ou $\nu < 2r_c - 2$ le message est envoyé à v en utilisant un chemin passant par X. Notons que le chemin utilisé en routant dans X n'est pas forcément un plus court chemin.

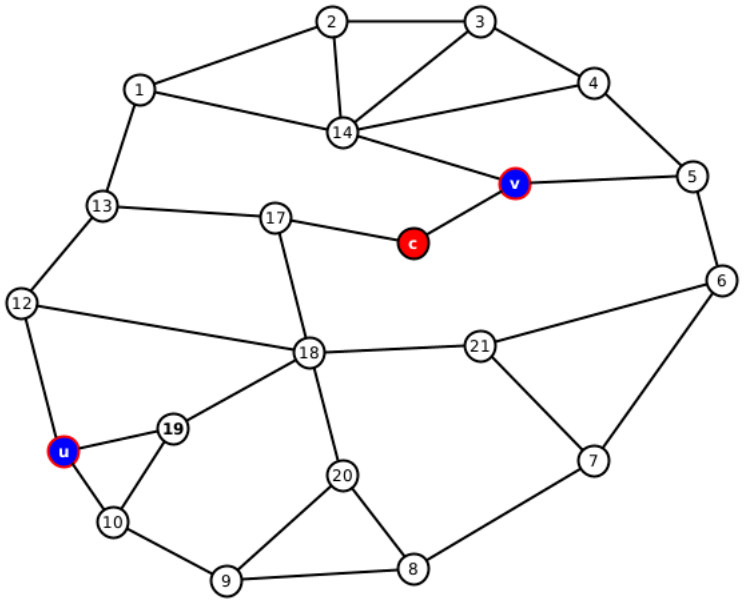

FIGURE 3.6 – Une disposition des nœuds u et v pour le cas **Route2**

3.3.3 Cas où le nœud destinataire est dans X et non le nœud source ($u \notin X$ et $v \in X$)

ROUTE3(u,v) **:**

Si $(estVoisin(v,X))$ ***alors***
  Si $(u.valInt[0] > v)$ ***alors***
    $v \leftarrow min(u.valInt[0]-v, |X|+v-u.valInt[1])$
  Sinon
    $v \leftarrow min(|v-u.valInt[1], |X|-u.valInt[0]+v)$
  Si $(v > 2r_c - 1)$ ***alors***
    $RouteT_c(v)$
  Sinon
    $RouteX(v)$

Si $(\neg estVoisin(v,X))$ ***alors***
    Si $(u.valInt[0] > v)$ ***alors***
        $v \leftarrow min(u.valInt[0]-v, |X|+v-u.valInt[1])$
    Sinon Si $(v.valInt[0] < u < v.valInt[1])$ ***alors***
    Sinon
        $v \leftarrow min(v-u.valInt[1], |X|+u.valInt[0]-v)$
    Si $(v > 2r_c - 1)$ ***alors***
        $RouteT_c(v)$
    Sinon
        $RouteX(v)$

Avéwé Basséne affirme dans ses explications que le cas 3 est similaire au cas 2 au facteur prés que le rôle du nœud source v change du nœud destinataire u.

3.3.4 Cas où le nœud source et destinataire ne sont pas dans X $(u, v \notin X)$

ROUTE4(u, v) **:**

Si $(v.valInt[0] > u.valInt[1])$ ***alors***
$\nu \leftarrow min(v.valInt[0] - u.valInt[1], |X| - v.valInt[0] + u.valInt[0])$
Sinon
$\nu \leftarrow min(u.valInt[0] - v.valInt[1], |X| - u.valInt[1] + v.valInt[0])$
Si $((estVoisin(u, X) \wedge estVoisin(v, X)))$ ***alors***
Si $(\nu > 2r_c - 2)$ ***alors***
$RouteT_c(v)$
Sinon
$RouteX(v)$
Si $((estVoisin(u, X))$ **<>** $(estVoisin(v, X)))$ ***alors***
Si $(\nu > 2r_c - 3)$ ***alors***
$RouteT_c(v)$
Sinon
$RouteX(v)$
Si $((\neg estVoisin(u, X)) \wedge (\neg estVoisin(v, X)))$ ***alors***
Si $(\nu > 2r_c - 3)$ ***alors***
$RouteT_c(v)$
Sinon
$RouteX(v)$

Lorsque les nœuds u, v ne sont pas dans X. La fonction *execute* de **ROUTEab** exécute la fonction **Route4**. Cette dernière s'exécute au niveau du nœud source, récupère l'entête du message contenant l'ensemble des informations nécessaire pour router un message de u à v. Et calcule la distance dans X entre le plus grand descendant de u et le plus petit descendant de v dans X, et la distance entre le plus petit descendant de u et le plus grand descendant de v dans X. Le minimum entre les deux distances est stocké dans ν.

Pour router un message de u à v, l'auteur a pris en compte trois positions possibles. Soit les deux sommets sont tous voisins directs de X, soit les deux sommets ne sont pas voisins directs de X, ou soit l'un est voisin direct de X et l'autre non. Et compare ν et $2r_c - 2$ si u et v sont voisins directs de X, ou ν et $2r_c - 3$ si u ou v est voisin direct de X, ou ν et $2r_c - 4$ si u et v ne sont pas voisins directs de X.

Dans le cas où ν est supérieur à $2r_c - 2$ ou $2r_c - 3$ ou $2r_c - 4$ alors le message est routé dans T_c par de plus court chemin puisque aucun chemin dans T_c ne dépasse pas $2r_c - 2$ ou $2r_c - 3$ ou $2r_c - 4$.

Dans le cas contraire ou ν est inférieur à $2r_c - 2$ ou $2_c - 3$ ou $2r_c - 4$, le message est routé dans X. Notons que le chemin de routage donné par la fonction de routage **ROUTEab** n'est pas un plus court chemin. Cependant, Avéwé Basséne [BAS17] a démontré le théorème 3.3 ci-dessous, qui confirme que le schéma de routage est de facteur d'étirement d'au plus 2.

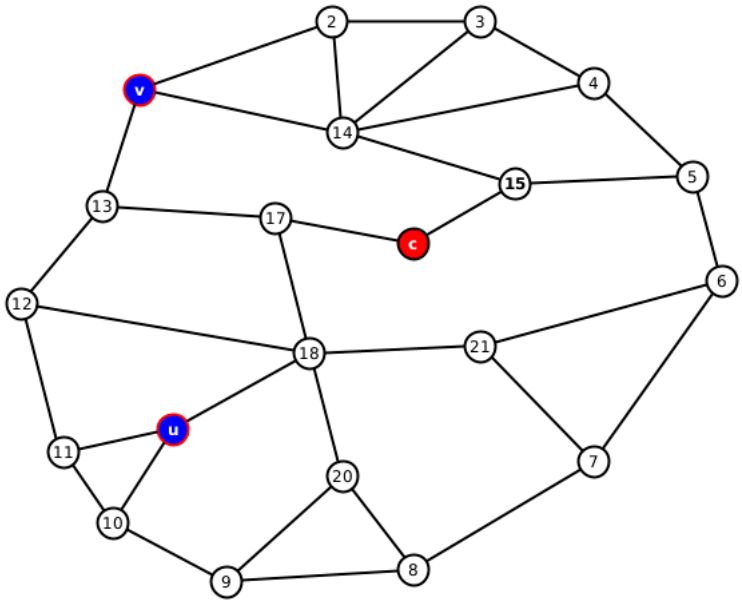

FIGURE 3.7 – Une disposition des nœuds u et v pour le cas **Route3**

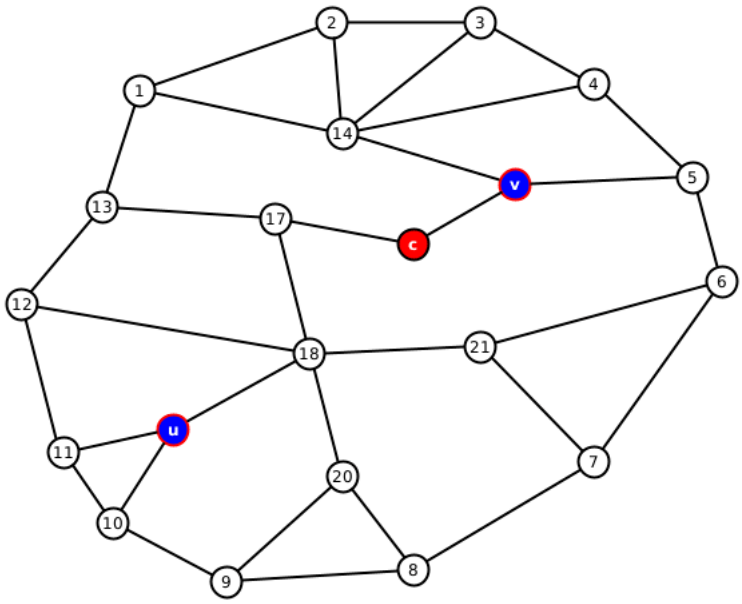

FIGURE 3.8 – Une disposition des noeuds u et v pour le cas **Route4**

Théorème 3.3 *[BAS17] Soit G un graphe de Halin et soit X le cycle extérieur de G, alors pour tout $u, v \in G$ l'algorithme **ROUTEab** permet de router le message sur un plus court chemin avec un facteur d'étirement d'au plus $\eta \leq 2$.*

CHAPITRE

4

CONTRIBUTION : ROUTAGE COMPACT DE PLUS COURT CHEMINS DANS LES GRAPHES DE HALIN CUBIQUES

Dans ce mémoire, nous avons portés notre réflexion sur la conception d'un schéma de routage compact de plus court chemin pour les graphes de Halin. Ce travail avait comme objectif de traiter la suite du résultat apporté par Avéwé Bassène [BAS17], dans le cadre de son mémoire de master, à savoir un schéma de routage compact dans les graphes de Halin avec un facteur d'étirement d'au plus 2.

Comme dans beaucoup de problème de recherche, lorsque ces derniers semble difficile, on est souvent appelé à les subdiviser en sous problèmes pour les rendre plus simple à étudier. Ainsi, n'ayant pas apporté une réponse à la question posée, nous avons proposé un schéma de routage compact de plus court chemin pour une sous-famille de graphes de Halin beaucoup plus simple, appelé les graphes de Halin cubique complet. Ce dernier est l'objet de ce chapitre.

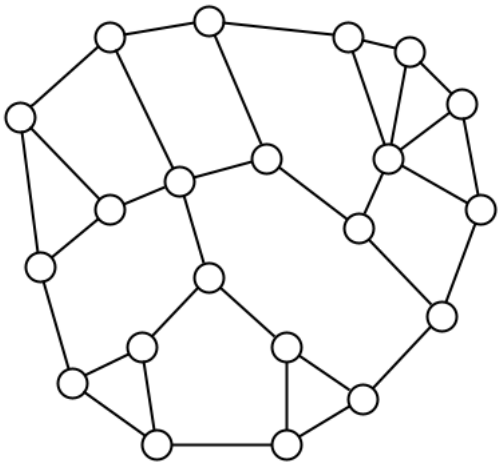

FIGURE 4.1 – Exemples de graphes de Halin

4.1 Graphe de Halin

Les graphes de Halin ont été introduits en 1971 par le mathématicien allemand *Rudolfd Halin*.

Les graphes de Halin ont gagnés en importance, lorsqu'on a découvert que beaucoup de problèmes algorithmiques qui étaient difficiles à résoudre par calcul pour les graphes planaires arbitraires pouvaient être résolus efficacement avec ceux de Halin. Exemple de graphe de Halin, voir figure 4.1.

Leurs structures proches des arbres, et aussi un atout pour la résolution de beaucoup de problèmes algorithmiques.

Définition 4.1 *Un graphe G=(V,E) est de Halin, s'il peut être obtenu par plongement d'un arbre sans sommet de degré 2 et avec au moins 4 sommets dans le plan, et de connecter ces feuilles par un cycle qui ne traversent pas ces arêtes.*

Proposition 4.1 *[SP83] Tout graphe de Halin est strictement 3-connexe.*

La face extérieure d'un graphe de Halin est de taille $\kappa(H) = m - n + 1$.

Proposition 4.2 *[SP83] Un graphe planaire 3-connexe H est de Halin si et seulement si une de ces faces est de taille $\kappa(H)$.*

Corollaire 4.1 *[SP83] Si H est un graphe de Halin, alors $\kappa(H)$ est la taille de sa face la plus large.*

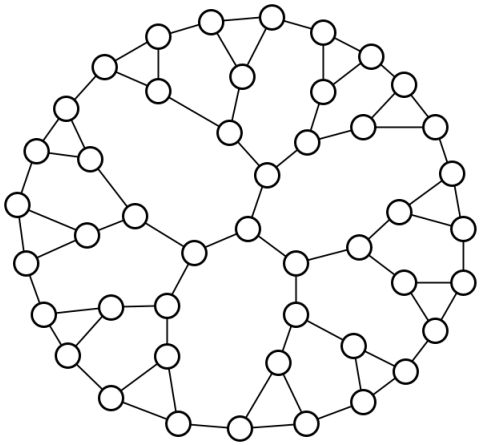

FIGURE 4.2 – Exemples de graphes de Halin cubique complet

Comme sous-famille de graphes de Halin on peut citer la famille des graphes de Halin cubique complet.

Définition 4.2 *[ST08] Un graphe de Halin cubique complet H est un graphe de Halin cubique (i.e, 3-régulier) dont l'arbre caractéristique T est un arbre cubique complet.*

Définition 4.3 *Un arbre cubique complet est un arbre dans lequel toutes les feuilles sont égales distance à la racine de l'arbre.*

4.2 Centre et rayon d'un graphe de Halin

On considère un graphe $G = (V, E)$, et deux sommets u et v de G.

L'écart $d(u, v)$ dénote la longueur du plus court chemin allant de u à v. Ainsi l'écartement d'un sommet u est définit par $e(u) = max_{v \in V, v \neq u} d(u, v)$.

Le centre est un sommet c_0 d'écartement minimum. Ainsi, l'écartement de c_0 est appelé le rayon du graphe G, et est noté $\rho(G)$. Pour plus de détaille voir *chapitre 4 dans [BER70]*.

Proposition 4.3 *L'écart d(u,v) vérifie*

1. *d(u,v) = 0 si u=v.*
2. $d(u, v) + d(v, w) \geq d(u, w)$.
   Dans le cas où le graphe est symétrique
3. *d(u,v)=d(v,u).*

Étant donné un graphe de Halin $H = (V, E)$, soit X l'ensemble des sommets de la face extérieur et T l'arbre caractéristique de H, c'est-à-dire l'arbre obtenu de H en enlevant toutes les arêtes de la face extérieur de H, on peut aussi dire que T est l'arbre obtenu de H en rendant X stable.

En effet, un ensemble est stable si les sommets sont deux à deux non-adjacents.

Ainsi, un centre et le rayon d'un graphe de Halin H peut être obtenu en utilisant l'algorithme ci-dessous.

CentreRayon(T l'arbre caractéristique de H **) :**

$k \leftarrow 0$
TantQue $|T| > 2$ ***Faire***
    $T \leftarrow EnleverFeuilles(T)$
    $k \leftarrow k + 1$
Si $|T| = 1$ ***alors***
    $c_0 \leftarrow c \in T$
    $\rho(G) \leftarrow k$
Sinon Si $|T| = 2$ ***alors***
    $c_0 \leftarrow c_1 \in T$ ou $c_2 \in T$
    $\rho(G) \leftarrow k$

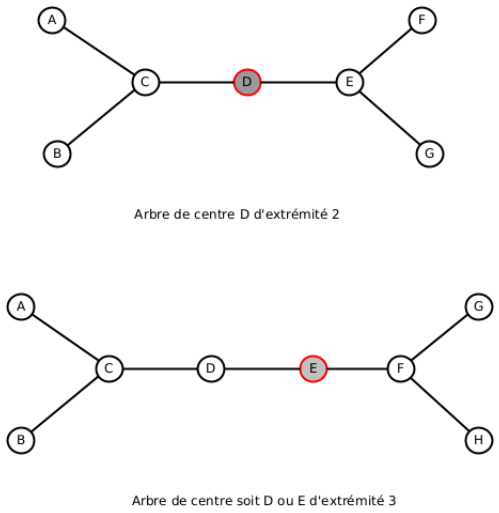

FIGURE 4.3 – Le graphe au dessus d'ordre 7 est de centre D et de rayon 2. Et le graphe en dessous est d'ordre 8 est de centre D ou E et de rayon 3

4.3 Schéma de routage compact de plus court chemin dans les graphes de Halin cubique complet

Dans cette section, nous allons présentés notre schéma de routage compact de plus court chemin pour les graphes de Halin cubique complet. Le résultat principal est donné par le théorème 4.1, ci-dessous.

Théorème 4.1 (Résultat principal) *Tout graphe de Halin cubique complet H=(V,E), admet un schéma de routage de plus court chemin avec des adresses et des entêtes de longueur* $O(\log n)$ *bits et des tables de routage de taille* $O(\log n)$ *bits. Le temps de décision est de* $O(m-n)$*, avec* $n = |V|$ *et* $m = |E|$.

4.3.1 Schéma de routage

Assignation des adresses et des ports

Soit H le plongement d'un graphe de Halin cubique complet d'ordre n et T son arbre caractéristique de centre c_0.

Soit une décomposition de l'ensemble des sommets de H en des sous-ensembles de sommets $L_0,...,L_{r_0}$ (avec r_0 le rayon du graphe H, donc de l'arbre T). L'ensemble L_i pour tout $0 \leq i \leq r_0$, est appelé *niveau* (*level*).

La décomposition consiste à regrouper tous les sommets situés à une distance i de la racine c_0 dans un *niveau* L_i.

Pour ce faire, on pose $L_0 = \{c_0\}$ et tout *niveau* L_i est définie comme suivant : $L_i = \{x : (\exists y \in L_{i-1}) \wedge ((x,y) \in E(G))\}$, c'est-à-dire l'ensemble des sommets qui ne sont pas affectés à un *niveau* et qui ont au moins un voisin dans le *niveau* précédent.

De cette décomposition en niveau du graphe H, il en découle le lemme **4.1** ci-dessus, qui donne une relation entre le numéro d'un niveau et le nombre de sommets de ce niveau.

Lemme 4.1 *Dans un graphe de Halin cubique $H = (V, E)$ complet un ensemble de niveau L_i est de cardinalité $|L_i| = 3.2^{i-1}$ pour $i > 0$. Le niveau L_0 contient un seul sommet (i.e, le centre du graphe c_0).*

Preuve :

On utilise la démonstration par récurrence.

Soit G un graphe cubique complet de centre c_0. Soit $L_0 = \{c_0\}$ le premier niveau, donc L_0 est de cardinalité 1. Puisque le graphe G est cubique alors c_0 est relié à 3 sommets qui sont de distance 1 à la racine c_0. Donc le deuxième niveau L_1 contient 3 sommets i.e $3.2^{1-1} = 3$.

Supposons que le n^{ieme} niveau L_{n-1} contient 3.2^{n-1} sommets et montrons que le $(n+1)^{ieme}$ niveau L_n contient 3.2^n.

Chaque sommet de L_{n-1} est relié à 3 sommets, un dans le niveau L_{n-2} et deux dans le niveau L_n, donc le nombre de sommets de L_n est égal à deux fois le nombre de sommets dans L_{n-1}, c'est-à-dire $2.3.2^{n-1} = 3.2^n$.

CQFD.

On affecte à chaque sommet $u \in V(G)$ un entier, noté *num(u)*.

Pour se faire, on commence par le $niveau\ L_0$, dont on donne à c_0 le numéro 1 (*num(c_0)=1*).

Puis on numérote les sommets de L_1 par des entiers de 1 à $3.2^{1-1} = 3$ en parcourant le niveau dans le sens contraire des aiguilles d'une montre.

Ensuite pour tout niveau L_k, $k > 1$. On numérote les sommets de L_k par des entiers de 1 à 3.2^{k-1}, en commençant par l'enfant de gauche du sommet numéro 1 dans le *niveau* L_{k-1}.

Dans l'arbre caractéristique T du graphe plongé H, un sommet s a un *parent* w, un enfant de *gauche* u et de *droite* v, ainsi nous avons démontrer le lemme suivant, qui sera utile pour notre algorithme de routage.

Lemme 4.2 *Soit H un graphe de Halin cubique complet. Soit s, u et v trois sommets de H tels que u et v sont des enfants de s. Alors dans notre schéma d'étiquetage si u est l'enfant de gauche et v l'enfant de droite alors on a :*

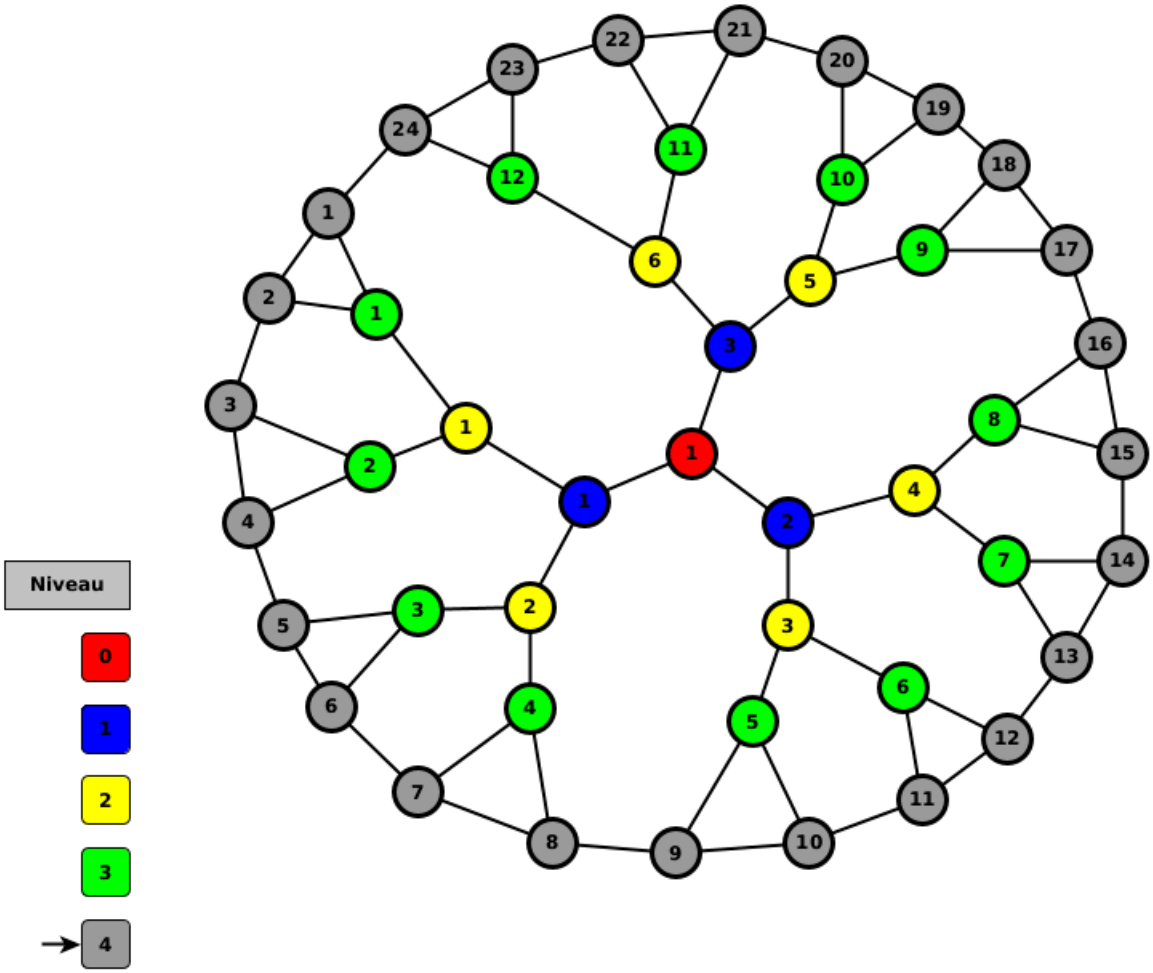

FIGURE 4.4 – Exemple de numérotation des sommets d'un graphe cubique complet

1.) num(v)=2.num(s)
2.) num(u)=2.num(s)-1

Preuve :
Soit $L_k = \{u_1, u_2, ..., u_n\}$, $k < \rho(H)$. Pour tout sommet u_i il existe deux sommets $u_{i,1}$ et $u_{i,2}$ respectivement l'enfant de gauche et de droite de u_i. Soit u_1 un sommet numéroté par 1 alors son enfant de gauche $u_{1,1}$ est numéroté par 1 et $u_{1,2}$ est numéroté par 2, d'où $u_{2,1}$ et $u_{2,2}$ sont respectivement numérotés par 3 et 4. Par itération il existe un entier $n_0 \leq n$ tel que pour $u_k \in L_k$ $num(u_{k,1}) = 2n_0 - 1$ et $num(u_{k,2}) = 2n_0$ donc $num(u_{k+1}) = n_0 + 1$ par conséquent $num(u_{k+1,1}) = 2n_0 + 1 = 2(n_0 + 1) - 1 = 2.num(u_{k+1}) - 1$ et $num(u_{k+1,2}) = 2n_0 + 2 = 2(n_0 + 1) = 2.num(u_{k+1})$.

CQFD.

Ainsi l'identifiant $id(u)$ d'un sommet $u \in V(G)$ est donné par le numéro de son *niveau* et son numéro dans ce dernier.

Soit L_i le $niveau$ contenant le sommet u et $num(u)$ le numéro de u dans L_i, donc l'identifiant de u est :

$$\boxed{id(u) =< num(u), L_i >}$$

Ainsi l'adresse d'un nœud est égale à l'identifiant de ce dernier.

$$\boxed{l(u) = id(u) =< num(u), L_i >}$$

Notons qu'ici l'entête d'un message est constituée uniquement de son adresse.

Dans la définition de notre schéma de routage. Chaque sommet pour faire le travail, qui consiste à transmettre des informations vers un autre sommet, a besoin juste de connaître son descendant qui a l'identifiant le plus petit et son descendant qui a l'identifiant le plus grand dans X (l'ensemble des sommets sur la face extérieur). Nous noterons tab_u le tableau à deux entrées contenant ces deux derniers, pour tout sommet u, $tab_u[0]$ contient le plus petit descendant de u et $tab_u[1]$ contient le plus grand descendant de u.

La comparaison des identifiants dans ce cas de figure n'est pas un problème, puisque les sommets de la face extérieur sont tous dans le même *niveau* alors comparer les identifiants revient à comparer les numéros des sommets dans le dernier *niveau* c'est-à-dire celui représentant la face extérieur.

Ainsi pour tout sommet $u \in V(H)$ la table de routage est :

$$\boxed{table(u) =< id(u), tab_u >}$$

Cette sous-section se termine par la numérotation des ports par des entiers $\{0, 1, 2, 3\}$. Le numéro 0 est attribué au port d'un sommet u faisant le lien à lui même *(c'est-à-dire la connexion entre le routeur et la machine qui lui est associée, son localhost)*. Et 1 le numéro du port qui relie u à son *parent*, 2 le numéro de port qui le relie à son *enfant de gauche* et le 3 celui qui le relie à son *enfant de droit*.

Nous allons ci-dessous définir la fonction de routage qui en exploitant le schéma d'étiquetage décide localement vers quel sommet faire transiter les informations.

Algorithme de routage :

Avant de définir la fonction ou l'algorithme de routage, nous allons définir deux fonctions **ANCETRECOMMUN** et **ANCETRE**. Ces deux fonctions sont indispensables pour le bon fonctionnement de notre fonction de routage.

ParentDe(u) **:**
$num(u) \leftarrow \lceil num(u)/2 \rceil$
$L_u \leftarrow L_u - 1$

La procédure **ParentDe** ci-dessus qui pour un sommet donné permet de passer à son parent. Dans la fonction **ANCETRECOMMUN** nous allons utilisés la procédure **ParentDe**.

ANCETRECOMMUN (u, v) **:**
Si $(L_u = L_v)$ ***alors***
TantQue $num(u) \neq num(v)$ ***Faire***
ParentDe(u)
ParentDe(v)
Sinon Si $(L_u > L_v)$ ***alors***
TantQue $L_u \neq L_v$ ***alors***
ParentDe(u)
TantQue $num(u) \neq num(v)$ ***Faire***
ParentDe(u)
ParentDe(v)
Sinon
ParentDe(v)
TantQue $num(u) \neq num(v)$ ***Faire***
ParentDe(u)
ParentDe(v)
$w \leftarrow u$
retourner w

La fonction **ANCETRECOMMUN** reçoit deux sommets, soit u et v, et détermine le parent commun de u et v le plus proche, c'est-à-dire le plus petit sous-arbre de l'arbre caractéristique contenant à la fois u et v.

On teste d'abord si u et v sont dans le même niveau ou pas, dans le cas où u et v sont dans le même niveau L_i alors on cherche le parent de u dans le niveau $L_i - 1$ et le parent de v dans le même niveau ; si le parent de u est différent du parent de v alors on recommence le même processus jusqu'à ce qu'on obtient un sommet qui est à la fois parent de u et de v et qui se trouve dans un niveau $L_k < L_i$.

Sinon si le niveau où se trouve u est supérieur à celui de v, alors on cherche un parent de u qui est dans le même niveau que v, soit z ce sommet. Et on cherche le parent commun des sommets z et v.

Sinon, c'est-à-dire si v se trouve dans un niveau supérieur à celui de u alors on exécute le même travail en échangeant les rôles de u et de v. Par exemple sur la figure 4.3.1 le sommet w est le parent commun le plus proche des sommets u et v, c'est-à-dire w est la racine du plus petit arbre contenant u et v.

ANCETRE (u, v, w) **:**
  $L_{w_i} \leftarrow L_u$
  $n_{w_i} \leftarrow num(u)$
  Repeter
    $n_{w_i} \leftarrow n_{w_i}/2$
    $L_{w_i} \leftarrow L_{w_i} - 1$
  Jusqu'à $((tab_u[1] < tab_{w_i}[1])$ ou $(tab_u[0] > tab_{w_i}[0])$ ou $(w_i = w))$
  Si $(L_w = L_{w_i})$ **alors**
    retourner w
  retourner w_i

La fonction **ANCETRE** qui prend u et v et retourne un sommet w_i parent de u (autre que w) tels qu'on a un des cas suivants :

$tab_{w_i}[1] \in [tab_u[1], tab_v[0]]$ ou $tab_{w_i}[0] \in [tab_u[0], tab_v[1]]$. Notons que si u est dans X alors il peut arriver que u soit confondu avec $tab_{w_i}[0]$ ou avec $tab_{w_i}[1]$, voir la figure 4.5 ci-dessus.

Avec le lemme ci-dessus, un message passe soit par l'ancêtre commun w, soit par un ancêtre w_i de la source du message tels que w_i est aussi un descendant de w.

Lemme 4.3 *Soit u, v, w trois sommets de G tels que w est l'ancêtre commun de u et de v, le plus proche. Si u doit envoyer un message à v alors le message passe soit par w, soit par l'arête (u_i, v_i) tels que $(u_i, v_i) \in X$ et w sont sur la même face ; de plus $u_i, v_i \in [tab_u[1], tab_v[0]]$ ou $u_i, v_i \in [tab_u[0], tab_v[1]]$.*

La preuve du lemme 4.3 est une évidence. En effet, tout sommet de G correspond à une arête dans X.

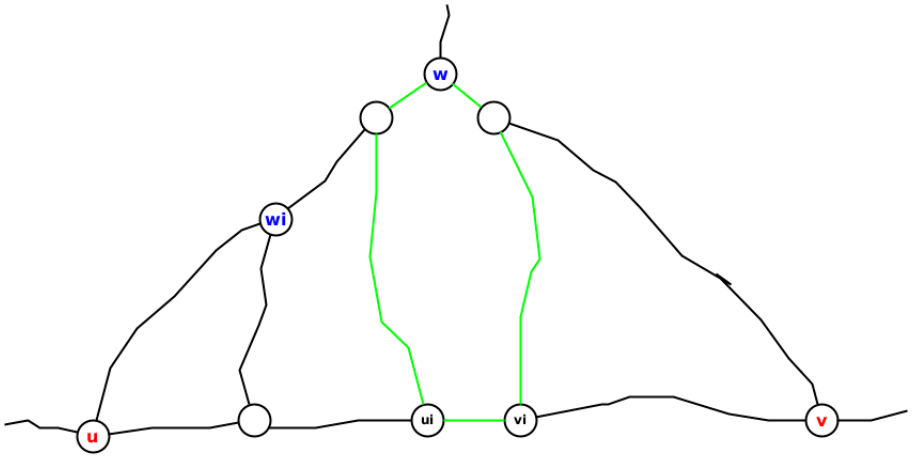

FIGURE 4.5 – w est le parent commun le plus proche de u et v, en vert la face contenant le sommet w et l'arête (u_i, v_i), w_i est évidement le parent de u

Dans notre schéma de routage, pour router par de plus court chemin d'un sommet u à un sommet v, la fonction de routage compare la longueur du chemin reliant u à v dans l'arbre calculer en utilisant la fonction $\mathbf{dist}_T(u,v)$ et la longueur du chemin dans X calculer en utilisant la fonction $\mathbf{dist}_X(u,v)$.

dist$_T(u,v,w)$:
    $d \leftarrow L_u + L_v - 2.L_w$
    retourner d

dist$_X(u,v)$:
    Si $(num(tab_u[1]) > num(tab_v[0]))$ **alors**
        $d \leftarrow L_{tab_u[1]} - L_u + (num(tab_u[1]) - num(tab_v[0])) + L_{tab_v[0]} - L_v$
    Sinon
        $d \leftarrow L_{u[1]} - L_u + (num(tab_v[0]) - num(tab_u[1]) + L_{tab_v[0]} - L_v$
    retourner d

L'algorithme de routage présenté ci-dessous est élémentaire, l'originalité vient de son explication.

ROUTE (u, v) **:**
  $w \leftarrow$ **ANCETRECOMMUN**(u, v)
  $k \leftarrow 0$
  Si ($(u[1] < v[0])$) **alors**
    $k \leftarrow 1$
  Si ($(dist_T(u, v, w) > dist_X(u, v))$) **alors**
    $w_i \leftarrow$ **ANCETRE**(u, v, w)
    Si ($(dist_T(u, w_i[k], w_i) \leq dist_X(u, w_i[k]))$) **alors**
      $RouteT$
    Sinon
      $RouteX$
  Sinon
      $RouteT$

Explication de l'algorithme :

Notre algorithme pour faire transité un message d'un sommet u vers un sommet v, cherche d'abord l'ancêtre commun le plus proche de u et v dans l'arbre, soit w le sommet donné par la fonction **ANCETRECOMMUN**. Puis de comparer la distance du chemin dans X et la distance du chemin dans T passant par w entre u et v.

Si la distance dans T est plus petite que la distance dans X alors on route le message dans T.

Sinon le message ne passe pas par w, dans ce cas le message passe soit par un parent de u dans l'arbre, qui est descendant de w, soit le message passe dans X.

Si un tel sommet w_i existe alors on compare de nouveau la distance dans X et celle dans T entre u et $tab_{w_i}[0]$ (si $tab_u[0] > tab_v[1]$) ou entre u et $tab_{w_i}[0]$ (si $tab_u[1] < tab_v[0]$). Si celle dans T est plus petite alors on route le message dans l'arbre c'est-à-dire vers son parent, sinon on le route dans X.

Si la distance du chemin dans T passant par w entre u et v est plus grand alors on route le message dans X. Voir figure 4.6

4.3.2 Correction :

Lemme 4.4 *Soit $G = (V, E)$ un graphe de Halin, de nombre de sommets $|V| = n$ et de nombre d'arêtes $|E| = m$, alors le rayon de G, noté $\rho(G)$ est au plus $\kappa(G)/2 = (m - n + 1)/2$, avec $\kappa(G)$ le nombre de sommets de la face extérieur.*

De plus si le graphe est 3-régulier alors $\rho(G) = \kappa(G)/2$

Preuve :

Le plus petit graphe de Halin est la roue à 4 sommets R_4.

Soit R_4^f le sous-graphe formé par les sommets de la face extérieur de R_4 et R_4^i le sous-graphe formé par les sommets internes de R_4. Ainsi, tout graphe de Halin G peut

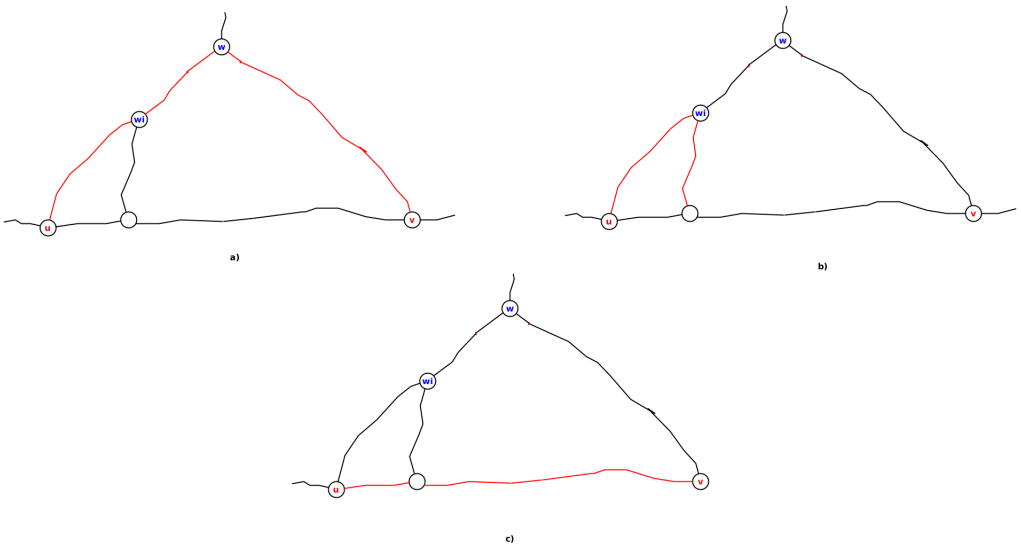

FIGURE 4.6 – w est le parent commun le plus proche de u et v, en vert la face contenant le sommet w et l'arête (u_i, v_i), w_i est évidement le parent de u

être construit à partir de la roue R_4 en ajoutant de manière itérative un sommet dans R_4^f (subdivision d'une arête de R_4^f) et en reliant ce sommet par un sommet de R_4^i déjà existant ou en le reliant avec un sommet que l'on ajoute d'abord dans R_4^i (subdivision d'une arête de R_4^i.

CQFD.

Le lemme 4.4 que nous avons démontré, nous permet de calculer de manière beaucoup plus précise le temps d'exécution de notre algorithme de routage.

Lemme 4.5 *Soit H un graphe de Halin cubique complet, soit X l'ensemble des sommets de la face extérieur, soient u et v deux sommets de H alors la fonction de routage permet de router tout message par de plus court chemin en utilisant des adresses et des tables de $O(\log n)$ bits. Le temps de décision est de $O(m-n+1)$ par sommet.*

Preuve :

D'après la sous-section 4.3.1 le schéma de routage est de plus court chemin. Donc reste à montrer la compacité de ce dernier.

L'adresse d'un sommet u est constitué par le numéro de u et son niveau. Donc l'adresse d'un sommet u, peut être coder en $2\log n$, donc en $O(\log n)$ bits. Et puisque la table de routage d'un sommet est égale à son adresse alors elle peut aussi être coder en $O(\log n)$.

Le temps d'exécution de **ROUTE** dépend de **ANCETRE** et de **ANCETRECOMMUN**. En effet, pour les deux fonctions **ANCETRE** et **ANCETRECOMMUN** on cherche l'ancêtre d'un sommet en passant d'un niveau à un autre. Hors le nombre de niveau est au plus le rayon du graphe $\rho(G)$. Ce qui fait que chacune des fonctions **ANCETRE** et **ANCETRECOMMUN** s'exécute en $O(m-n)$ puisque le graphe est 3-régulier. Donc **ROUTE** s'exécute en $O(m-n)$.

CQFD.

CHAPITRE

5

CONCLUSION GÉNÉRALE ET PERSPECTIVES

Dans cette partie, nous allons faire un rappel de l'ensemble des résultats obtenus dans le routage compact par intervalle et de plus court chemin. Et nous allons ensuite faire une brève synthèse de notre résultat sur les graphe de Halin cubique complet, et enfin nous allons faire une présentation des perspectives relatives aux résultats sur le routage par intervalle et de plus court chemin ; et les perspectives relatives à notre résultat.

5.1 Conclusion

Ce stage de recherche a été subdivision en cinq (5) chapitres. Après le chapitre d'introduction générale, on a un deuxième chapitre présentant les généralités sur les graphes, sur la notion de mineur et ainsi que la notion d'algorithmique et de complexité algorithmique. Dans le troisième chapitre, nous avons fait une présentation sur le routage par intervalle, le routage de plus court chemin dans les arbres et les graphes planaire-extérieurs. Ce dernier résultat qu'ils ont généralisé pour les graphes $(k,r)-constellation$. Dans cette partie nous avons aussi fait une présentation du résultat de A. BASSENE [BAS17] sur les graphe de Halin donnant un schéma de routage compact avec facteur d'étirement. Toujours dans le même chapitre, nous avons présentés les ré-

sultats sur le routage compact de plus court chemin. Parmi ces résultats, on peut citer celui de Fraigniaud et Gavoille [FG02], qui ont montré dans leur papier que tout arbre de n nœuds supporte un schéma de routage de plus court chemin avec des tables de routage, des en-tête de message et des adresses de nœuds de taille $O(\log n)$ bits.

Dans le quatrième chapitre, nous avons présentés notre résultat sur le routage compact de plus court chemin dans les graphes de Halin cubique complet. Et pour finir, ce chapitre fait l'objet d'une conclusion générale.

5.2 Perspectives

Dans cette partie, nous allons poser sous forme de questions l'ensemble des points ouverts de ce mémoire et qui seront l'objet de réflexion future.

Question 1

Dans [FG02], les auteurs ont définis le *clean-path* comme étant la séquence obtenue après élimination des rangs du *path* de valeur égale à 1. Le *path(u)* de tout sommet u est la séquence de rangs rencontrée sur le chemin de r à u. Basséne [BAS17] pose la question à savoir : En redéfinissant le *clean-path* comme étant la séquence de rangs du *path*, sans éliminé de valeur, pour obtenir le nombre exacte d'éléments du *path* dans les graphes non valués serons-nous en mesure de donner un schéma de routage compact de plus court chemin pour les graphes de Halin.

Question 2

P. Fraigniaud et C. Gavoille dans [FG02] et Thorup et Zwick dans [TZ01] ont obtenus indépendamment un schéma de routage de plus court chemin avec des tables de routage de $O(\log n)$ bits. Y. Dieng et C. Gavoille en se basant sur les travaux de P. Fraigniaud et C. Gavoille ont proposés dans [DG11] un schéma de routage compact de plus court chemin pour les graphes planaires extérieurs, qu'ils ont étendus dans les graphes (k,r)-constellation. Sachant que les graphes de Halin ont une structure très similaire à celle des arbres, une question se pose. Est-ce qu'on ne pourrait pas rendre les graphes de Halin planaire-extérieurs en supprimant un nombre k de sommets que l'on déterminera, et ensuite d'utiliser le résultat de Y. Dieng et C. Gavoille [DG11], résultat basé sur celui des arbres de P. Fraigniaud et C. Gavoille dans [FG02], pour router par de plus court chemin dans les graphes de Halin.

Question 3

Basséne dans son mémoire de master 2 [BAS17], il a constaté dans ces recherches qu'on est toujours appelé à comparer la longueur du chemin de routage à la longueur

maximale de l'arbre à chaque fois que nous désirons router dans G. De ce faite, il a poser la question à savoir si nous pouvons trouver une relation entre le nombre d'arêtes dans X noté $|X|$ et le rayon r_c de G, serait-il possible de faire un routage plus compact et par de plus court chemin dans les graphes de Halin.

Dans nos recherche nous avons trouver une relation d'inégalités entre le nombre de sommets dans X et le rayon du graphe r_c. Cependant le routage n'est toujours pas compact. Ainsi, on se pose de nouveau la question à savoir s'il est possible d'avoir une relation d'égalité entre le nombre de sommets dans X et le rayon du graphe, serons-nous en mesure de proposer un schéma de routage de plus court chemin plus compact.

Question 4

Basséne s'est aussi posé la question : est-il possible de trouver un étiquetage compact des sommets d'un graphe de Halin de tel sorte que la distance entre tout paire de sommets du graphe puisse être déduit en examinant seulement leurs étiquettes ?

Nous avons dans nos recherche proposer un tel étiquetage pour les graphes de Halin cubique complet. Cependant, on se pose toujours la question à savoir s'il est possible d'étendre notre schéma d'étiquetage pour les graphes de Halin ou de trouver un autre type de schéma d'étiquetage.

BIBLIOGRAPHIE

[AGGM06] Ittai ABRAHAM, Cyril GAVOILLE, Andrew V. GOLDBERG, and Dahlia MALIKHI. Routing in networks with low doubling dimension. *In 26 th International Conference on Distributed Computing Systems (ICDCS). In . IEEE Computer Society Press*, July 2006.

[BAS17] Avewe BASSÈNE. Etat de l'art sur le routage compact. Master's thesis, Université Assane Seck de Ziguinchor, 01 2017.

[BER70] C. BERGE. *Graphes et hypergraphes*. Collection Dunod Université. Paris, Dunod, 1970.

[BM76] John Adrian BONDY and U. S. R MURTY. *Graph theory with application*. North-Holland Publishing, 1976.

[BM08] John Adrian BOMDY and U. S. R MURTY. *Graph theory*. Springer, 2008.

[CLRS01] Thomas CORMEN, Charles LEISERSON, Ronald RIVEST, and Clifford STEIN. *Initiation à l'algorithme*, volume 1176. MIT Press, 2001.

[DG11] Youssou DIENG and Cyril GAVOILLE. Routage compact optimal dans les (k, r)-constellations. *LaBRI, Université de Bordeaux, RSTI-TSI*, 2011.

[FG02] Pierre FRAIGNIAUD and Cyril GAVOILLE. Routing in trees. *LNCS*, 2002.

[GAV00] Cyril GAVOILLE. A survey on interval routing. *Theoretical Computer Science*, page 217–253, 2000.

[MOH01] Bojan MOHAR. Graph minors and graphs on surfaces. *Inst. of Mathematics, Physics and Mechanics, Dep. of Mathematics, Univ. of Ljubljana*, 39(48729451), 2001.

[MT01] Bojan MOHAR and Carsten THOMASSEN. Graphs on surfaces. *The Johns Hopkins university Press*, 2001.

[NSHO08] Mohamed NOHQIR, André ST-HILQIRE, and Taha. B. OUARDA. The bayesian-regularized neural network approach to model daily water temperature in a small stream. *Journal of Water Science*, 21(3) :373–382, 2008.

[RSa] N. ROBERSTON and P. SEYMOUR. Graph minors iii : Planar tree-width, submitted. *Journal of Combinatorial Theory*.

[RSb] N. ROBERSTON and P. SEYMOUR. Graph minors iv : Tree-width and well-quasi-ordering, submitted. *Journal of Combinatorial Theory*.

[RSc] N. ROBERSTON and P. SEYMOUR. Graph minors v : Excluding a planar graph, submitted. *Journal of Combinatorial Theory*.

[RS83] N. ROBERSTON and P. SEYMOUR. Graph minors i : Excluding a forest. *Journal of Combinatorial Theory*, 35 :39–61, 1983.

[RS86a] N. ROBERSTON and P. SEYMOUR. Graph minors ii : Algorithmic aspects of tree-width. *Journal of Combinatorial Theory*, 7 :309–322, 1986.

[RS86b] N. ROBERSTON and P. SEYMOUR. Graph minors. vi. disjoint paths across a disc. *Journal of Combinatorial Theory*, 41 :115–138, 1986.

[RS86c] N. ROBERSTON and P. SEYMOUR. Graph minors xiii : The disjoint paths problem. *Journal of Combinatorial Theory*, septembre 1986.

[RS88] N. ROBERSTON and P. SEYMOUR. Disjoint paths on a surface. *Journal of Combinatorial Theory*, 45 :212–254, 1988.

[RS90a] N. ROBERSTON and P. SEYMOUR. Disjoint crossed paths. *Journal of Combinatorial Theory*, 49 :40–77, 1990.

[RS90b] N. ROBERSTON and P. SEYMOUR. A kuratowski theorem for general surfaces. *Journal of Combinatorial Theory*, 48 :255–288, 1990.

[RS91] N. ROBERSTON and P. SEYMOUR. Obstructions to tree-decomposition. *Journal of Combinatorial Theory*, 52 :40–77, 1991.

[RS95a] N. ROBERSTON and P. SEYMOUR. Circuits on a surface. *Journal of Combinatorial Theory*, 60 :72–106, 1995.

[RS95b] N. ROBERSTON and P. SEYMOUR. Distance on a surface. *Journal of Combinatorial Theory*, 64 :240–272, 1995.

[RS95c] N. ROBERSTON and P. SEYMOUR. Excluding a non-planar graph. *Journal of Combinatorial Theory*, 89 :43–76, 1995.

[RS95d] N. ROBERSTON and P. SEYMOUR. Extending an embedding. *Journal of Combinatorial Theory*, 65 :23–50, 1995.

[RS95e] N. ROBERSTON and P. SEYMOUR. Giant steps. *Journal of Combinatorial Theory*, 68 :65–110, 1995.

[RS99] N. ROBERSTON and P. SEYMOUR. Taming a vortex. *Journal of Combinatorial Theory*, 77 :162–210, 1999.

[RS03] N. ROBERSTON and P. SEYMOUR. Tree-decompositions and well-quasi-ordering,. *Journal of Combinatorial Theory*, 89 :77–108, 2003.

[RS04a] N. ROBERSTON and P. SEYMOUR. Wagner's conjecture. *Journal of Combinatorial Theory*, 92 :325–357, 2004.

[RS04b] N. ROBERSTON and P. SEYMOUR. Well-quasi-ordering on a surface. *Journal of Combinatorial Theory*, 90 :325–385, 2004.

[SK85] N. SANTORO and R. KHATIB. Labelling and implicit routing in networks. *Comp. J.*, page 5–8, 1985.

[SP83] Maciej M. SYSTO and Andrzej PROSKUROWSKI. On halin graphs. *Mathematische Annalen*, (1), 1983.

[ST08] W. C SHIU and W. K TAM. The strong chromatic index of complete cubic halin graphs. *Elsevier*, 2008.

[TOU92] Claude TOUZET. *LES RESEAUX DE NEURONES ARTIFICIELS, INTRODUCTION AU CONNEXIONNISME*, volume 130. Collection de l'EERIE, 1992.

[TZ01] Mikkel THORUP and Uri ZWICK. Compact routing schemes. *In Proc. 19th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 2001.

[WAG37] K. WAGNER. Über eine eigenschaft der ebenen komplexe. *Mathematische Annalen*, (1) :570–590, December 1937.

[WT93] R WARLIMONT and R.B. TAN. Factorisatio numerorum with constraints. *Journal of Number Theory*, 1993.

Schéma de routage compact de plus court chemin dans les graphes de Halin cubiques complets

Résumé :

Savoir comment transmettre une information est fondamental dans un réseau. Il est essentiel que chaque entité du réseau dispose d'une fonction de routage lui permettant de décider localement, avec sa vue du réseau, du chemin par lequel l'information doit passer. Dans le routage compact, on cherche à mettre en place de tels algorithmes tout en optimisant la table de routage.

Dans leur travaux, sur le routage compact, Dieng et al., ont posé la question de savoir s'il est possible de router par de plus court chemin dans un graphe de Halin avec des tables de routage et des entêtes de message de taille $O(\log n)$ bits. Les seuls graphes possédant un tel schéma de routage sont les arbres, les graphes planaire extérieurs et les (k,r)-constellations. En ce qui est des graphes de Halin, les résultats apportés sont de Bassène et al. avec la proposition d'un schéma de routage compact avec un facteur d'étirement de 2.

Dans ce mémoire, nous proposons un schéma de routage de plus cours chemin, dans les graphes de Halin cubiques complets ; une sous famille des graphe de Halin.

Discipline : Informatique

Mots clefs : Routage compact, routage par intervalle
graphe planaire, graphe planaire-extérieur,
graphe de Halin

Université Assane Seck de Ziguinchor (UASZ)

Compact shortest path routing scheme in complete cubic Halin graphs

Abstract :

Knowing how to transmit information is fundamental in a network. It is essential that each entity in the network has a routing function that allows it to decide locally, with its view of the network, which path the information should take. In compact routing, we try to implement such algorithms while optimizing the routing table.

In their work on compact routing, Dieng et al. asked the question of whether it is possible to route through the shortest path in a Halin graph with routing tables and message headers of size $O(\log n)$ bits. The only graphs with such a routing scheme are trees, external planar graphs and (k,r)-constellations. As far as Halin graphs are concerned, the results are from Bassène et al. with the proposal of a compat routing scheme with a stretching factor of 2.

In this paper, we propose a shortest path routing scheme in complete cubic Halin graphs ; a sub-family of Halin graphs.

Discipline : Computer science

Keywords : Compact Routing, interval routing
planar graph, outerplanar-graph
Halin graph

Assane Seck University of Ziguinchor (UASZ)