N° ordre 4960
LaBRI, Université de Bordeaux
INRIA Bordeaux Sud-Ouest
Projet Européen Euler

THÈSE

présentée à l'université de Bordeaux 1
pour obtenir le grade de Docteur en Sciences
spécialité Informatique

Algorithmes de routage
de la réduction des coûts de communication à la dynamique

CHRISTIAN GLACET

soutenue le 6 décembre 2013 devant le jury composé de

Directeurs	NICOLAS HANUSSE et DAVID ILCINKAS
Rapporteurs	SEBASTIEN TIXEUIL et LAURENT VIENNOT
Examinateurs	DAVID COUDERT et CYRIL GAVOILLE

Remerciements

Je tiens tout d'abord à remercier mes directeurs de thèse, Nicolas Hanusse et David Ilcinkas qui ont tous deux su m'encadrer avec parcimonie, me laissant la chance de m'exprimer par moi-même tout en me gardant sur la bonne voie. Grâce à eux et également à Cyril Gavoille j'ai vraiment eu une expérience de travail de recherche en équipe, durant laquelle j'ai eu de leur part un support moral et scientifique très appréciable. De la même façon j'aimerais remercier Colette Johnen avec qui j'ai eu le plaisir de travailler à plusieurs reprises et qui m'a également aidé à peaufiner ma thèse.

J'ai également été très honoré d'avoir Sebastien Tixeuil et Laurent Viennot comme rapporteurs, ils m'ont tous deux permis d'améliorer la qualité de mon tapuscrit. Merci à eux ainsi qu'à David Coudert pour avoir pris le temps d'assister à ma soutenance et d'avoir discuté avec moi de l'avenir de mes recherches.

Je me dois également de remercier Dimitri Papadimitriou, qui a mis sur pieds le projet Européen qui à donné jour à mon financement de thèse.

Merci à Lucas, avec qui la cohabitation a été très agréable, et ce même pendant les périodes les plus difficiles. Merci à lui de m'avoir aidé dans mes recherches et également d'avoir organisé à l'aide d'Ilouane mon pot de thèse.

Je tiens également à remercier tous mes amis sans qui cette thèse aurait été bien plus difficile, merci à mes colocataires Fabien, Maxime, Pti Ben et Valentine, merci également à Benzaie, David, François-Xavier, Guillaume, Jean-François, Joseph, Manu, Marie-Alex, Nicolas, Olivier, Sébastien, Thibault, Vivian, Yoan.

Un grand merci également à tous les doctorants ayant séjourné dans le bureau 123 du LaBRI durant mes trois années de thèse et avec qui j'ai pu partager mes travaux et autres discussions diverses. En particulier je remercie Ève, Gabriel, Jérôme, Nesrine, Omar, Oualid, Pierre, Tom, Wafa. Merci également aux autres membre du LaBRI Ahmed, Andra, Antoine, Arnaud, Akka, Émilie, Florian, Isabelle, Julien, Louise-Amélie, Michel, Nicholas, Nicolas, Romaric, Sofian, Thomas et tout particulièrement Philippe Narbel, sans qui je n'aurais probablement pas poursuivi en thèse.

Je voudrais bien évidement, pour finir, remercier mes parents qui ont cru en moi malgré les difficultés de parcours, ainsi que Carole qui a toujours su être là pour m'épauler durant ces trois années. Merci à elle qui a su me conforter dans mes choix et me soutenir lors des derniers mois difficiles que sont ceux de l'écriture du tapuscrit.

Résumé

Algorithmes de routage, de la réduction des coûts de communication à la dynamique

Répondre à des requêtes de routage requiert que les entités du réseau, nommées routeurs, aient une connaissance à jour sur la topologie de celui-ci, cette connaissance est appelée *table de routage*. Le réseau est modélisé par un *graphe* dans lequel les *nœuds* représentent les routeurs, et les *arêtes* les liens de communication entre ceux-ci. Cette thèse s'intéresse au calcul des tables de routage dans un modèle *distribué*. Dans ce modèle, les calculs sont effectués par un ensemble de processus placés sur les nœuds. Chaque processus a pour objectif de calculer la table de routage du nœud sur lequel il se trouve. Pour effectuer ce calcul les processus doivent communiquer entre eux. Dans des réseaux de grande taille, et dans le cadre d'un calcul distribué, le maintien à jour des tables de routage peut être coûteux en terme de communication. L'un des thèmes principaux abordés et celui de la réduction des coûts de communication lors de ce calcul. L'une des solutions apportées consiste à réduire la taille des tables de routage, permettant ainsi de réduire les coûts de communication. Cette stratégie classique dans le modèle *centralisé* est connue sous le nom de *routage compact*. Cette thèse présente notamment un algorithme de routage compact distribué permettant de réduire significativement les coûts de communication dans les réseaux tels que le réseau internet, i.e. le réseau des systèmes autonomes ainsi que dans des réseaux *sans-échelle*. Ce document contient également une étude expérimentale de différents algorithmes de routage compact distribués. Enfin, les problèmes liés à la dynamique du réseau sont également abordés. Plus précisément le reste de l'étude porte sur un algorithme auto-stabilisant de calcul d'arbre de plus court chemin, ainsi que sur l'impact de la suppression de nœuds ou d'arêtes sur les tables de routage stockées aux routeurs.

Mots clefs : routage compact, calcul distribué, graphe, dynamique, coût de communication, communication asynchrone

Abstract

Routing algorithms, from communication cost reduction
to network dynamics

In order to respond to routing queries, the entities of the network, nammed routers, require to have a knowledge concerning the topology of the network, this knowledge is called *routing table*. The network is modeled by a graph in which nodes represent routers and edges represent communication links between nodes. This thesis focuses on routing tables computation in a distributed model. In this model, computations are done by a set of process placed on nodes. Every process has for objective to compute the routing table of the node on which he is placed. To perform this computation, processes have to communicate with each other. In large scale network, in the case of a distributed computation, maintaining routing tables up to date can be costly in terms of communication. This thesis focuses mainly on the problem of communication cost reduction. One of the solution we propose is to reduce routing tables size which allow to reduce communication cost. In the centralised model this strategy is well known under the name of *compact routing*. This thesis presents in particular a distributed compact routing algorithm that allows to reduce significantly the communication costs in networks like Internet, i.e. the autonomous systems network and others networks that present *scale-free* properties. This thesis also contains an experimental study of several distributed compact routing algorithms. Finally, some problems linked to network dynamicity are addressed. More precisely, the problem of network deconnexion during a shortest path tree computation with auto-stabilisation guaranties, together with a study of the impact of several edges or nodes deletion on the state of the routing tables .

Keywords : compact routing, distributed computing, graph, dynamics, communication cost, asynchronous communication

Sommaire

Sommaire

Introduction et motivations 1

Sommaire

Le *routage* est la tâche qui consiste à trouver une route d'un point A vers un point B dans un réseau. Il s'effectue en réponse à une *requête de routage*. Ici, la requête est "comment se rendre au point B depuis le point A ?". Cette requête vient également avec une attente implicite, dépendante du contexte, sur la qualité de la route fournie en réponse. La qualité d'une route se mesure classiquement par sa longueur comparativement à la distance réelle dans le réseau. Ce réseau sera modélisé par un graphe $G = (V, E)$, dans lequel les n nœuds (ensemble V) sont les destinations sources ou intermédiaires utilisées pour *router*, et les m arêtes (ensemble E) sont les liens permettant d'aller d'un nœud à un de ses *voisins*.

L'algorithme qui répond à une requête de routage est appelé *algorithme de routage*. La réponse à une requête de routage est une indication de route, partielle ou complète selon les cas. On distingue ces deux cas, que ce soit pour du routage dans un réseau de télécommunication ou encore dans un réseau routier :

- Par exemple, lorsqu'une personne utilise une carte complète du réseau, elle est capable de déterminer avant même le départ l'ensemble de son itinéraire. Ce type de routage est appelé *routage à la source* (ou *source-routing* en anglais).
- Cependant, si le chemin est inconnu au départ et que cette même personne utilise les panneaux de direction pour se rendre au point B, c'est au fur et à mesure de sa progression vers B que l'itinéraire va être découvert/calculé. À chaque intersection les panneaux directionnels sont utilisés pour prendre une décision, ces derniers indiquant, dans un monde idéal, la ville B à chaque intersection. Ce type de routage est appelé *routage pas à pas* (ou *destination-based routing* en anglais).

Protocole de routage Les règles de routages sont décrites par une sous-partie de l'algorithme de routage, le *protocole de routage*. Dans le cas d'un réseau de

télécommunication, le routage d'un paquet s'effectue en utilisant des informations stockées aux nœuds et appelées *tables de routage*.

Table de routage. Une table de routage est composée d'*entrées*, chaque entrée donne des informations permettant de router vers un autre nœud du graphe. Dans le cadre du routage dans un réseau de télécommunication, les tables de routage sont des tables de correspondance entre identifiants de destination et chemins permettant d'atteindre le nœud ayant cet identifiant.

Schéma de routage et routage distribué. Les tables de routage sont calculées par une sous-partie de l'algorithme de routage, appelé *schéma de routage*. Un algorithme de routage est qualifié de *distribué* si le schéma de routage n'utilise aucune connaissance centralisée/globale concernant le graphe, chaque nœud étant un processus qui exécute une copie de l'algorithme distribué et n'ayant qu'une vision locale du réseau. Souvent, chaque nœud ne connaît avant l'exécution du schéma de routage que son propre identifiant et l'ensemble de ses ports de sortie et doit échanger des informations avec les autres nœuds du graphe pour construire sa table de routage. Remarquons que le protocole de routage est par définition toujours distribué. Dans le cas d'un algorithme de routage distribué, les nœuds s'échangeront donc deux types de messages distincts :

- des *messages de données*, i.e. des envois de paquets ;
- des *messages de contrôle*, utilisés pour construire les tables de routage de manière distribuée.

Classiquement, le routage se fait via des plus courts chemins dans le graphe. Cela paraît naturel, il est en effet logique de désirer aller d'un point A à un point B le plus rapidement possible, i.e. par un plus court chemin.

1.1 Routage de plus court chemin

Un algorithme de routage de plus court chemin est un algorithme qui fourni, pour tout nœud destination $v \in V$ et depuis toute source $u \in V$, une route de longueur égale à la distance entre u et v dans le graphe. Les algorithmes de routage de plus court chemin utilisent différents schémas de routage, certains seront décrits dans le chapitre 2 (DSDV, LSR, ...). Il est cependant connu que tout algorithme de routage de plus court chemin requiert qu'une partie significative des nœuds connaissent l'ensemble des destinations [GP96a]. Cette contrainte sur la connaissance minimale requise pour le routage de plus court chemin peut poser problème dans certains contextes, notamment pour le routage entre systèmes autonomes sur Internet, exemple qui est détaillé en suivant.

1.1.1 Problématique du routage dans le réseau des systèmes autonomes

Sur Internet, le routage est découpé hiérarchiquement. Le routage de plus haut niveau se fait entre entités appelées *systèmes autonomes*. Chaque système autonome (AS) est un ensemble de réseaux administrés par une même entité décisionnelle, par exemple un fournisseur d'accès internet. Un AS annonce plusieurs destinations (préfixes IP), chacun de ces préfixes étant un identifiant de destination potentiel pour une requête de routage. Autrement dit le réseau des systèmes autonomes peut

être vu comme un graphe dans lequel chaque nœud a plusieurs identifiants (un par préfixe annoncé). Lors du routage dans ce graphe, si un paquet doit être acheminé vers une destination d'identifiant A alors l'algorithme de routage devra router ce paquet vers le nœud ayant A parmi ses identifiants (il est possible que plusieurs nœuds aient A parmi leurs identifiants, ce phénomène est appelé *multihoming*).

Aujourd'hui, le nombre de préfixes annoncés par un AS est en moyenne de 19, le nombre d'AS est de plus de 45 220 et le nombre total de préfixes annoncés est de plus de 944 320 (statistiques obtenues via [HBS13]). Remarquons néanmoins qu'un AS ne connaît pas nécessairement une route vers tout préfixe annoncé. L'algorithme de routage de plus court chemin utilisé dans le réseau des AS est appelé BGP, pour *Border Gateway Protocol*. La figure 1.1 montre que le nombre d'entrées générées par ce schéma de routage augmente d'environ 15% par an depuis l'an 2000 pour un AS donné. Cette hausse s'explique notamment par la multiplication du nombre de préfixes annoncés.

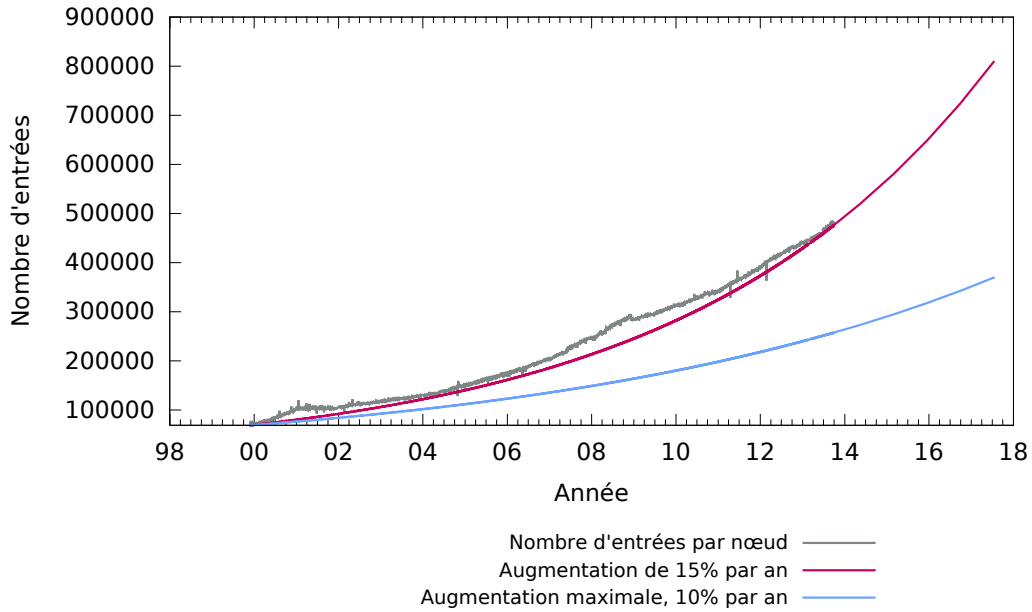

Figure 1.1: Nombre d'entrées pour le système autonome AS65000 d'après les statistiques provenant du dernier rapport CIDR de [HBS13]. Cette figure propose un comparatif entre l'évolution réelle et l'évolution maximale qui permettrait de conserver la rapidité de traitement des messages qui était observable en l'an 2000.

L'existence de tables de routage aussi volumineuses a un impact majoritaire sur les coûts de mise à jour, autrement dit sur la charge du réseau. Sans entrer dans les détails techniques, assez intuitivement, plus la quantité d'informations à maintenir à jour est importante, plus il est coûteux de les maintenir à jour. En effet

- chaque changement dans la topologie du graphe risque d'altérer une route connue et donc d'impliquer une mise à jour de celle-ci ;
- le temps de recherche de route associée à un préfixe dans les tables de routage augmente.

La question clef est donc : *Est-ce que les évolutions matérielles sont suffisamment rapides pour couvrir l'augmentation du nombre d'entrées dans les tables* BGP *?* Cette question est directement liée à la vitesse d'évolution des technologies semi-conducteurs et donc à la conjecture de Moore [M+65] qui stipule que le nombre de composants par circuit intégré double chaque année, et ce en considérant l'utilisation de composants de coût minimum.

Le compte rendu du séminaire *IAB sur l'adressage et le routage* [MZF+07] présente en détails les problèmes techniques liés à cette croissance rapide du nombre d'entrées sur les routeurs. Meyer *et al.* expliquent que la conjecture de Moore ne tient pas pour la fabrication de routeurs. Le silicium utilisé dans les routeurs est particulier et uniquement utilisé dans ce domaine. Or ce silicium particulier est produit en bien plus faible quantité que le silicium classique. Les investissements faits dans ce domaine sont moindres ce qui implique que la loi de Moore telle quelle ne s'applique pas directement à l'évolution des performances des routeurs.

D'autres sont même plus pessimistes concernant la conjecture de Moore et pensent qu'elle ne sera plus vérifiée dans un futur proche. Dans [Kis02], Kish expose une limitation physique à la compression des circuits intégrés. Les évaluations théoriques proposées dans cet article montrent que le bruit électrique empêcherait de dépasser un certain niveau de concentration des composants, ce qui aurait évidemment pour effet direct de stopper presque irrémédiablement l'évolution des circuit intégrés en terme de concentration spatiale*.

Un second problème est exposé dans ce même compte rendu. Les auteurs insistent en s'appuyant sur les travaux menés par Molinero-Fernández dans sa thèse [MF03], sur le fait que la vitesse d'accès à la mémoire de type DRAM, utilisée dans les tables de routage (RIB, pour *routing information base*) augmente seulement de 10% par an. D'après ce même compte rendu, le coût de mise à jour d'une entrée est linéaire en fonction du nombre d'entrées contenues dans la table de routage, ce qui implique que le temps de propagation des mises à jour augmente d'un facteur proche de 4.5% chaque année ($15\%/10\%$), soit environ 80% en treize ans. La figure 1.1 montre l'évolution réelle du nombre d'entrées comparée à une augmentation de 10% par an, qui est la limite permettant de conserver les même temps de mise à jour qu'il y a treize ans.

Il est donc possible que l'augmentation du nombre de systèmes autonomes devienne un réel problème pour le routage. C'est pourquoi il est important de prévoir des alternatives au routage actuel. L'objectif initial de cette thèse est donc de réduire la taille des tables dans l'espoir de réduire les coûts de communication. Dans cette optique, d'après [GP96a], il est nécessaire d'accepter de faire des détours lors du routage. En effet, l'usage de plus courts chemins implique qu'une grande partie des nœuds stockent des informations sur la majorité des nœuds.

Pour résumer, les questions principales qu'amène cette section sont :

Q1 : Comment réduire la taille des tables de routage ?

Q2 : Réduire la taille des tables de routage permet-il de réduire les coûts de communication ?

*D'après l'auteur, la seule solution serait d'augmenter la fréquence d'horloge.

1.2 Réduire la taille des tables en acceptant de faire des détours

Les problèmes du routage de plus court chemin présentés dans la section précédente ont amené à considérer de nouvelles stratégies de routage. Remarquons premièrement qu'il existe deux sous-classes d'algorithmes de routage :

- Les algorithmes *dédiés* à certaines familles de graphes (planaire, genre borné, ...). Certains algorithmes dédiés ne fonctionnent que si le graphe possède certaines propriétés, d'autres fonctionnent quelle que soit la topologie mais n'ont de garanties de performances que sur certains graphes.
- La seconde classe regroupe les algorithmes ayant des garanties indépendamment du type de graphe considéré. Les algorithmes appartenant à cette deuxième classe sont qualifiés d'*universels*.

Nous verrons dans ce chapitre quelques exemples d'algorithmes pour ces deux catégories.

L'algorithme idéal serait celui qui a de bonnes garanties quel que soit le graphe considéré tout en étant très efficace pour le réseau sur lequel il sera déployé. La section 1.3 décrira les propriétés observées pour de nombreux réseaux sous l'appellation "graphe de terrain". Les mesures les plus importantes pour comparer les algorithmes de routage distribués sont :

- L'**étirement des routes** : le ratio de la longueur de la route sur la distance réelle dans le graphe. L'étirement d'un algorithme de routage est le ratio maximal pouvant être obtenu sur l'ensemble des routes.
- La **taille des tables de routage** ;
- La **mémoire de travail** : c'est la mémoire utilisée pour calculer et stocker les tables de routage. Elle est mesurée pour chaque nœud et est au moins égale à la taille de la table de routage de ce dernier.
- Le **coût de construction** : il est mesuré en fonction des interactions entre les nœuds, par exemple le nombre de messages envoyés par l'ensemble des nœuds durant l'exécution de l'algorithme, i.e. le *coût de communication*.
- Et le **temps de convergence** : c'est le nombre de rondes passées avant que tous les nœuds n'atteignent leur état final et restent dans cet état. La définition précise de ronde est variable selon les modèles, elle sera précisée par la suite, mais correspond intuitivement à une période durant laquelle chaque nœud a eut l'occasion d'échanger des informations avec ses voisins au moins une fois.

Ces mesures de performances seront décrites plus en détails dans le chapitre 2.

Pour réduire le nombre d'entrées nécessaire à chaque nœud, une première stratégie consiste à changer les noms des nœuds et à utiliser des noms ayant une valeur sémantique plutôt que des noms arbitraires. Un algorithme nécessitant un renommage des nœuds est habituellement qualifié d'*étiqueté* en opposition aux algorithmes de *routage avec indépendance des noms* qui ne s'autorisent pas de renommage.

Exemple jouet. Pour se faire une idée rapide de cette différence, considérons le graphe suivant :

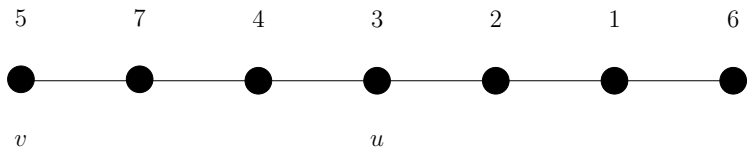

Figure 1.2: Routage dans une chaîne avec un algorithme avec indépendance de noms.

Pour router depuis le nœud u d'identifiant 3 vers le nœud v d'identifiant 5, il est nécessaire que le nœud u stocke une entrée dans sa table de routage lui permettant de déterminer si v se trouve à sa gauche ou à sa droite. Pour être capable de router vers tous les nœuds du graphe, v doit donc stocker une table de correspondance entre $n-1$ identifiants et un port de sortie. Supposons maintenant que l'algorithme de routage soit autorisé, avant d'exécuter le schéma de routage, à renommer les nœuds comme suit :

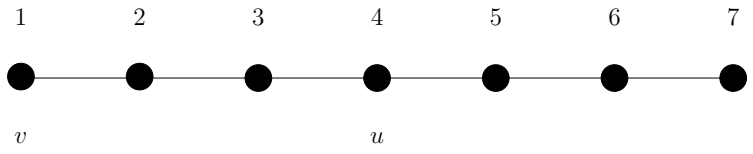

Figure 1.3: Routage dans une chaîne avec un algorithme étiqueté.

Le routage dans ce cas devient très simple, le nœud u n'a plus qu'à savoir à quel port de sortie correspondent les identifiants inférieurs puis en comparant son identifiant à celui proposé par la requête de routage il décidera s'il doit router à gauche ou à droite. Ici v a l'identifiant 1 et u a l'identifiant 4, le routage se fera donc vers le côté gauche correspondant aux identifiants plus petits. Pour être capable de router vers tous les nœuds du graphe, v doit donc uniquement savoir, dans ce second cas, à quel port correspond le coté gauche (ou droit).

Une supposition classique et qui sera faite dans cette thèse est que les identifiants des nœuds sont *distincts*, quel que soit le modèle considéré.

1.2.1 Routage utilisant un renommage des nœuds.

Les algorithmes de *routage étiquetés* utilisent des identifiants de nœuds qui correspondent à une position du nœud dans le graphe ou dans un plan dans le cas du routage géométrique. Ces identifiants sont appelés classiquement *localisations* (*locators* en anglais) car ils donnent une information sur la localisation du nœud ou encore *labels* ou *étiquettes* de routage.

Routage géométrique glouton. Cette catégorie de routage géométrique est basée sur une idée simple introduite notamment dans [TK84]. Pour un graphe $G = (V, E)$ donné, lors du routage d'un nœud $u \in V$ vers un nœud $v \in V$, à chaque saut dans le graphe, le nœud courant envoie la requête de routage à un de ses voisins les plus proches de v. Le nom d'un nœud u étant des coordonnées (x_u, y_u) d'un plan euclidien

(comme le montre par exemple la figure 1.4), toute décision de routage est prise par un nœud en utilisant uniquement les identifiants de tous ses voisins lui inclus et l'identifiant de la destination. Cela peut en pratique impliquer, pour des nœuds de degré linéaire en n, d'avoir une mémoire linéaire. Toutefois, la mémoire totale des nœuds n'excède pas la somme des degrés $2m$. Cette solution permet, lorsque le degré des nœuds est faible de réduire drastiquement la mémoire requise comparativement à un algorithme de plus court chemin.

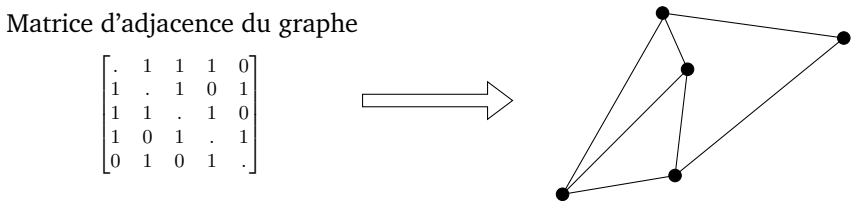

Figure 1.4: Association d'un placement des nœuds dans le plan d'un graphe.

Le problème du routage glouton est que le routage n'est pas garanti dans le cas général, son succès étant lié au positionnement des points dans le plan. La présence de *minima locaux* peut empêcher certains routages d'aboutir comme le montre la figure 1.5. En effet le routage de u vers v induit nécessairement une boucle car le nœud u est le plus proche nœud du graphe en terme de distance euclidienne, il est donc le plus proche nœud de v du point de vue de u_1 ou u_2. Donc si le nœud u route vers un de ces deux voisins le routage glouton reconduira vers u. Si le routage géométrique glouton est possible dans un graphe on dit qu'il existe un plongement glouton de ce graphe. Dans [May06], Maymounkov prouve qu'un plongement glouton ne peut pas être obtenu pour tout graphe dans un plan euclidien de dimension constante. Plus précisément, effectuer un plongement glouton d'un graphe de n nœuds requiert un plan euclidien d'au moins $\log n$ dimensions. Un algorithme de routage glouton sur un plongement à $\log n$ dimension est présenté dans [WP09].

Il existe cependant différentes techniques pour éviter ce type de blocage. Une d'entre-elles consiste à changer de technique de routage lorsque la route mène à un minimum local.

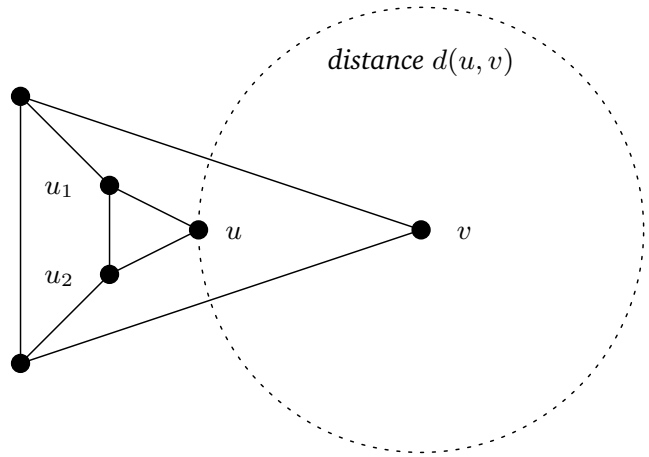

Figure 1.5: Echec du routage de u vers v dû à la présence d'un minimum local.

Routage par faces. À l'instar du routage glouton, le routage géométrique par faces ne souffre pas du problème des minima locaux. Cependant, il requiert que le graphe soit planaire[†]. Si les nœuds du graphe une fois placés sur un plan permettent de dessiner un graphe sans que ses arêtes s'intersectent, alors on parle de plongement planaire dans le plan. La technique de routage est encore une fois très simple, elle consiste à suivre les faces du graphe planaire tant que la droite passant par la source et la destination n'est pas intersectée [KSU99, BMSU01]. Cette méthode a cependant un gros inconvénient, l'étirement est en toute généralité non borné et peut, comme le montre [KSU99], avoir une longueur allant jusqu'à $4m$, cet algorithme a donc un étirement de $4m$.

Que peut-on faire dans un espace non-euclidien ? Le routage géométrique dans un plan hyperbolique est basé sur les même principes que le routage géométrique classique (euclidien). La différence réside dans le plongement du graphe, chaque nœud a des coordonnées dans un plan hyperbolique. L'avantage de cette technique est, comme le montre Kleinberg dans [Kle07], que tout graphe connexe fini a un plongement glouton dans un plan hyperbolique. La preuve de ce théorème est constructive, Kleinberg décrit un algorithme distribué qui attribue des positions dans un plan hyperbolique à chacun des nœuds du graphe en garantissant que le routage glouton classique réussi pour tout couple de nœuds. Indépendamment du coût de construction de plongement géométrique (non abordé ici), le routage géométrique souffre donc d'un manque de garanties concernant l'étirement du routage.

Routage d'étirement borné, routage compact Une autre alternative au routage de plus court chemin est appelée *routage compact*. Cette technique est généralement basée sur du routage dans des arbres de plus courts chemin. Mais contrairement à du routage de plus court chemin, le nombre d'arbres est sous-linéaire et chaque arbre ne couvre pas forcément la totalité du graphe. Les routages se font en utilisant plusieurs arbres. Tout nœud a ainsi une table de routage qui dépend uniquement du nombre d'arbres qui le couvrent et peut donc être de taille sous-linéaire. Toute route passe donc potentiellement par plusieurs arbres. Le routage compact prendra une part importante des recherches présentées dans cette thèse. L'avantage de cette méthode sur le routage géographique est qu'elle permet de donner des algorithmes ayant un étirement borné et ce même dans le cas général [KK77, Cow99, ABNLP90, AP92, TZ01]. L'algorithme [TZ01] donne un étirement garanti inférieur à 3 avec des tables de routage ayant de l'ordre de $\sqrt{n}$ entrées.

Problème du renommage des nœuds pour le routage. Le renommage des nœuds implique dans un cas pratique d'effectuer une correspondance entre les adresses réelles des nœuds (IP au autre identifiant) en étiquettes utilisées pour le routage. Cette correspondance pourrait être tenue à jour par une entité centralisée, à l'image d'un service DNS. Le problème de cette solution est que dans le cas d'un réseau dynamique il est coûteux de maintenir à jour cette correspondance car les étiquettes attribuées aux nœuds dépendent de la topologie du réseau (contrairement aux adresses IP). C'est notamment pour cette raison que cette thèse se focalise sur le routage sans renommage des nœuds.

[†]Comme le montre l'article [KGKS05] ce n'est pas absolument nécessaire, il existe en fait une propriété un peu moins forte qui une fois vérifiée permet le routage par faces.

1.2.2 Router sans renommer les nœuds.

En opposition au routage étiqueté, le routage avec indépendance des noms suppose que les identifiants des nœuds soient arbitraires. Autrement dit, les identifiants sont imposés par une entité extérieure. Une requête de routage ne contient au départ que l'identifiant du nœud destination et donc aucune information sur la localisation de celui-ci dans le graphe. Dans ce contexte, il est donc impossible d'utiliser les techniques de routage géométrique. Cependant, de façon assez surprenante, les algorithmes de routage compact avec indépendance des noms ont les même performances théoriques que les algorithmes étiquetés, comme le prouvent [ACL+06, AGM+08]. L'algorithme proposé dans [AGM+08] a exactement les même garanties de pire cas que l'algorithme étiqueté proposé dans [TZ01], ces deux algorithmes atteignent un compromis optimal en terme de mémoire et d'étirement (mémoire de l'ordre de $\sqrt{n}$ et étirement d'au plus 3). En effet, les limites connues pour le routage compact sont les suivantes :

1. Tout algorithme de routage compact ayant une mémoire par nœud de $o(n)$ a un étirement d'au moins 3 ;
2. Tout algorithme de routage compact ayant un étirement strictement inférieur à 5 a au moins une mémoire de $\Omega(\sqrt{n})$ pour une proportion constante des nœuds.

C'est pourquoi dans cette thèse nous nous intéresserons aux algorithmes de routage avec indépendance des noms et plus particulièrement à l'algorithme décrit dans [AGM+08]. Cependant, les solutions de routage compact de la littérature sont décrites de manière centralisée. Il s'ensuit la question suivante venant s'ajouter à celles déjà énoncées :

Q3 : Est-il possible de proposer un schéma de routage compact distribué économe en coût de construction des tables (coût de communication) ?

1.3 Propriétés des graphes de terrain

Les réseaux que l'on observe dans la nature/au quotidien ont souvent des propriétés en commun qui seront présentées dans cette section. Des graphes tels que les graphes des AS, des mots utilisés dans une même phrase [iCS01], des liens hypertexte [AJB99], des réseaux aériens [WC03] (d'autres exemples peuvent être trouvés dans [WC03]), ou encore le petit graphe d'exemple présenté en figure 1.6, présentent toutes les propriétés suivantes :

- **Sans échelle.** Un graphe *sans échelle* est un graphe dans lequel la répartition des degrés des nœuds suit une loi de puissance, i.e. le nombre de nœuds ayant un degré donné x est proportionnel à $x^{-\beta}$ avec β une petite constante. Intuitivement, de tels graphes ont donc beaucoup de nœuds de faibles degrés et quelques rares nœuds de degrés élevés, comme le montre la figure 1.6 dans laquelle la taille des nœuds correspond à leurs degrés respectifs. Cette particularité implique bien évidemment que le degré moyen des nœuds est faible. Il a été observé qu'en pratique, dans de nombreux réseaux [WC03], la valeur de β est comprise dans $[2, 3]$.
- **Petit-monde.** Un graphe *petit-monde* est un graphe de diamètre faible dans lequel les nœuds sont peu connectés (degré moyen faible par rapport à la taille

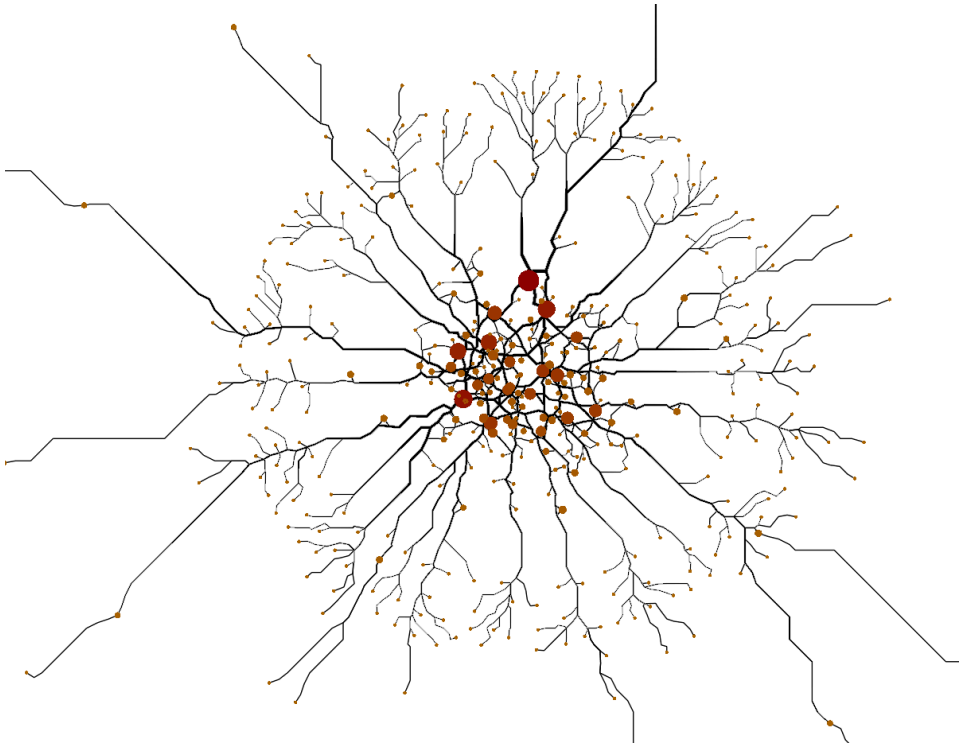

Figure 1.6: Graphe généré à l'aide de Tulip [Aub03, Tul13], présentant un graphe de 500 nœuds et supposé avoir des propriétés proches du graphe des AS (petit diamètre, nœuds centraux de haut degré, structure quasiment arborescente à la périphérie, degré moyen faible, ...). Ce graphe est un graphe de type GLP [Bol01]. Cette représentation utilise de plus une technique appelée *regroupement d'arêtes* qui est présentée dans [LBA10], qui accentue les propriétés énoncés.

du graphe). Degré et diamètre sont considérés faibles dans cette thèse s'ils sont logarithmiques en fonction du nombre de nœuds. Les graphes petit-monde sont également de bons candidats pour le routage glouton [Kle04].

Beaucoup de réseaux que l'on voit apparaître dans la vie de tous les jours présentent cette propriété. C'est notamment le cas des réseaux de relations sociales longtemps admis comme ayant un diamètre de 7, autrement dit, tout couple de personnes est séparé par au plus 6 liens de connaissance. Une première expérience menée par Milgram *et al.* en 1967 [TM69] tentait de montrer que cette déclaration est vraie. Elle consistait à faire parvenir des lettres à une personne donnée les envoyant uniquement à des connaissances, chaque lettre partant d'une personne distincte et ayant la même destination prévue. Cependant cette expérience n'eut pas le succès escompté et seulement 20% des lettres arrivèrent à leur destinataire (64 lettres parmi les 296 originellement émises).

Avec les moyens de communication actuels et notamment en utilisant les relations entre les utilisateurs de Facebook, cette expérience peut être simulée facilement, en évitant ainsi toute "perte" de lettre ou autre désagrément lié à la nature humaine des participants. Une expérience menée par Backstrom *et al.* sur des données fournies par Facebook ($720 \cdot 10^6$ utilisateurs et $69 \cdot 10^9$ liens

d'amitié) montrent dans [BBR+11] que la distance moyenne entre deux nœuds du graphe est de 4.74 et donc que le degré de séparation moyen entre deux individus est de 3.74. Enfin, le diamètre du graphe et donc la distance maximale entre deux individus est de 41, cependant 92% des couples de personnes sont à une distance de 5 ou moins.

La notion de petit-monde sous-entend également parfois que le *coefficient d'agrégation* (clustering coefficient en anglais) est fort [WS98]. Cela signifie grossièrement que deux nœuds voisins ont un voisinage très similaire. Du point de vue social, cela se traduit par l'expression bien connue "les amis de mes amis sont mes amis". Ce dernier point ne sera pas pris en considération dans cette thèse.

Remarque. La connaissance de ce type de propriétés peut aider à la conception d'algorithmes plus efficaces sur ces graphes. Certains algorithmes de routage compact universels utilisent ce genre de connaissances sur la structure du graphe pour obtenir de meilleures garanties que des algorithmes de routage compact classiques [KFY04, CSTW09, TZLL12].

De cette remarque découle la question suivante :

Q4 : Est-il possible de proposer un algorithme de routage universel performant si le graphe est petit-monde et/ou sans échelle ?

1.4 Dynamique du réseau

Dans un cas réel, la topologie du réseau est bien souvent vouée à évoluer au cours du temps. Selon le type de réseau considéré, l'évolution est plus au moins rapide et plus ou moins contrainte, mais dans tous les cas, il est important de décrire des algorithmes pouvant supporter ces variations. Dans le cas du routage, les modifications topologiques impliquent que certaines tables de routage stockent des routes erronées. La présence d'erreurs peut impliquer des détours dans certains cas, mais pire encore cela peut créer des boucles dans les routes. Une route n'étant pas mise à jour peut donc impliquer que des messages n'arriveront pas à leur destination.

Éviter les échecs lors du routage. Il existe plusieurs stratégies pour réduire le nombre d'échecs lors du routage en présence d'erreurs :

- **Approche proactive.** La première est de maintenir à jour les tables en échangeant régulièrement des messages de contrôle, cette stratégie est appelée *proactive*. Néanmoins, cette stratégie ne garantie pas que lors du routage il n'y aura pas d'erreurs du tout, mais réduit simplement la probabilité d'échec. De plus, maintenir à jour toutes les routes peut être coûteux et pas nécessairement utile en terme de routage. Supposons par exemple qu'une seule route R soit modifiée toute les 5 secondes à cause de la dynamique du graphe. Si la fréquence des requêtes de routage concernant la route R est journalière, alors cette table est maintenue à jour tout au long de la journée pour ne finalement être utilisée qu'une seule fois.

- **Approche réactive.** Suite à cette dernière observation il parait évident que dans certains cas la stratégie proactive n'est pas bonne. L'approche *réactive*

consiste à ne pas maintenir les routes à jour tant qu'aucune requête n'est faite pour celles-ci. Ainsi le coût de maintenance est réduit, mais la latence augmente si la route requise n'est pas disponible (à jour). Dans cette thèse l'approche réactive ne sera pas abordée.

- **Routage tolérant aux erreurs.** Des algorithmes tels que ceux présentés dans [DTCR08, BK09], permettent de garantir le routage de toute source vers toute destination si une seule entité est en panne dans le réseau, indépendamment du nombre d'erreurs induites dans les structures de données des différents nœuds. D'autres algorithmes probabilistes tels que [HIKN10, HKKK08] ou dédiés à certaines topologies [HKK02] garantissent le routage quelque soit le nombre de tables non à jour dans le réseau. En revanche pour ces algorithmes probabilistes, plus le nombre d'erreurs est important, moins la garantie sur l'étirement des routes est bonne.

  Le routage tolérant aux pannes peut être utilisé de paire avec un algorithme de maintien à jour des tables pour le rendre plus robuste.

Maintenir à jour une table de routage est souvent lié au calcul d'arbres de plus court chemins enracinés, comme dans [Hed88] ou [BG92]. L'un des problèmes majeur rencontré pour le maintien à jour d'arbres de plus court chemin est rencontré lors de déconnexions du graphe. Dans certains cas il est possible que des messages de contrôle continuent de circuler hors de la composante connexe de la racine de l'arbre. Cela ayant pour effet de saturer une partie du réseau en messages de contrôle. Il existe néanmoins des méthode pour empêcher ces messages de circuler infiniment longtemps, cependant ces méthodes requirent une connaissance globale sur le graphe (nombre de nœuds ou diamètre) [Mal94]. Cela peut impliquer des problèmes lorsque le diamètre ou le nombre de nœuds est amené à changer.

Les question principales amenées par la dynamique sur lesquels cette thèse se penchera sont les suivantes :

Q5 : Quels est l'impact de la dynamique sur le nombre d'erreurs ?

Q6 : Est-il possible de décrire un algorithme proactif de maintien à jour d'arbre de plus court chemin sans connaissance globale sur le réseau ?

1.5 Contributions

Les thèmes principaux de cette thèse sont l'étude d'algorithmes universels ainsi que les compromis entre mémoire, étirement et coût de maintien à jour de tables de routage. Le second chapitre présente les modèles qui seront utilisés dans la thèse. Le reste de la thèse s'articule autour des points suivants.

Réduire les coûts de communication grâce au routage compact. Le chapitre 3 présente des bornes inférieures liées au routage compact ainsi qu'un algorithme de routage compact distribué économe en coût de communication.

- La première borne inférieure donnée concerne le coût de communication et prouve que tout algorithme de routage générant pour $\Omega(n)$ nœuds des tables de routage qui requièrent une mémoire de M bits a un coût de communication de $\Omega(n \cdot M)$. D'après [GP96a], cela implique donc que tout schéma de routage

de plus court chemin requiert au moins $\Omega(n^2)$ messages pour construire ses tables, même pour un graphe petit-monde.

La seconde borne spécifie que tout algorithme de routage compact d'étirement borné a un temps de convergence au moins égal au diamètre du graphe.

- L'algorithme proposé est un algorithme de routage compact distribué et universel. C'est le premier algorithme distribué à garantir un étirement constant d'au plus 5. Cet algorithme est une adaptation de l'algorithme AGMNT (des noms de ses auteurs) centralisé universel donné dans [AGM+08]. Il utilise comme ce dernier des tables de routage de taille $\tilde{\mathcal{O}}(\sqrt{n})$ bits. La mémoire de travail et la taille des tables ont la même complexité asymptotique. Il a de plus un coût de communication plus faible que tout algorithme de plus court chemin ($\tilde{\mathcal{O}}(n^{3/2})$) pour des graphes petit monde. Cette implémentation de l'algorithme AGMNT est optimale à un facteur poly-logarithmique près d'après la borne inférieure sur la mémoire donnée dans ce même chapitre. Cela confirme l'intuition, le fait d'avoir de plus petites tables de routage peut effectivement permettre de réduire le coût de communication (au moins dans le cas non dynamique).

Le coût de l'universalité du routage. Le chapitre 4 ce concentre sur l'impact du choix de l'universalité sur les complexités en temps, mémoire et coût de communication des algorithmes présenté dans le chapitre 3. Plus précisément, ce chapitre apporte des réponses aux questions suivantes :

- Quelles sont les performances de ces algorithmes en comparaison à des algorithmes dédiés sur des graphes de terrain ?
- A quel point la topologie du réseau a un impact sur les performances des algorithmes dédiés et universels ?
- En relaxant la contrainte de compacité et en limitant uniquement le nombre d'entrées moyen par nœud, est-il possible de proposer un algorithme plus efficace sur des graphes de terrain ?

Le chapitre 4 apporte des réponses à ces questions en comparant de manière expérimentale l'implémentation distribuée présentée dans le chapitre 3 avec différents algorithmes de routage dédiés aux des graphes sans échelle. Ce quatrième chapitre présentera, par la même occasion, deux autres algorithmes distribués originaux LO et CLUSTER, ainsi qu'une implémentation distribuée de l'algorithme présenté dans [TZLL12].

- L'algorithme LO présenté est compact en moyenne et universel, qui garanti un étirement de 3 avec des tables de taille $\tilde{\mathcal{O}}(\sqrt{n})$ bits en moyenne, mais pouvant aller jusqu'à n bits pour au moins un nœud du graphe. L'algorithme LO a un coût de communication de $\tilde{\mathcal{O}}(\sqrt{n}m)$ pour tout graphe.
- Le second algorithme, CLUSTER, est quant à lui dédié aux graphes en loi de puissance et est compact avec forte probabilité. Il garantit dans cette même famille de graphes un étirement constant des routes. De plus, CLUSTER a en pratique de très petites tables de routage pour des graphes de terrain (5 en moyenne et 296 au maximum, pour un graphe représentant le réseau des AS et

possédant $17\,000$ nœuds). Le coût de communication de cet algorithme est de $\mathcal{O}(n^{3/2})$ pour des graphes sans échelle.

Réseaux dynamiques. La suite de la thèse se penche sur l'aspect dynamique du routage. Le chapitre 5 présente un algorithme auto-stabilisant qui garantit le maintien à jour d'un arbre de plus court chemin et ce sans avoir de connaissances globales sur le graphe : nombre de nœuds, ou diamètre. Cet algorithme permet notamment d'éviter le phénomène du *comptage à l'infini* sans connaître l'ensemble du graphe comme le propose le protocole [MRR80], et sans paramétrage requis, contrairement à [RLH06]. Le problème du comptage à l'infini traduit l'apparition de boucles infinies dans la propagation de changements d'états suite à des modifications de topologie. Ce phénomène augmente le temps de convergence, jusqu'à l'empêcher si l'algorithme est mal configuré ou que la configuration ne convient plus au graphe après des changements trop importants de sa topologie. L'algorithme proposé dans ce chapitre est intéressant car il est robuste et converge quel que soit la dynamique du graphe (même si le diamètre ou le nombre de nœuds est amené à changer radicalement).Il est également important de noter que l'algorithme proposé est valable même lors de déconnexion du graphe.

Impact de la dynamique sur l'état des tables de routage. Enfin le chapitre 6 étudie, pour des algorithmes basés sur des arbres de plus court chemin, la tolérance à la dynamique du réseau et montre que le nombre d'erreurs après $\mathcal{M}$ suppressions d'arêtes ou de nœuds est linéaire en fonction de $\mathcal{M}$ et proportionnel au diamètre D en moyenne. Plus précisément, le nombre d'erreurs créé est de $\mathcal{O}(\mathcal{M}\frac{Dn}{m})$. Cette borne est prouvée être précise pour $\mathcal{M} = 1$.

Modèles et outils 2

Sommaire

Ce chapitre présente les principales notations ainsi que les modèles et outils utilisés dans cette thèse. Il n'a aucune prétention d'exhaustivité. Commençons par présenter les notations utilisées pour les graphes.

2.1 Graphes

2.1.1 Les modèles de graphes

Un graphe est caractérisé par deux ensembles V et E, respectivement les nœuds et les arêtes de celui-ci. Le nombre de nœuds et le nombre d'arêtes d'un graphe $G = (V, E)$ sont respectivement notés $|V| = n$ et $|E| = m$. L'ensemble des voisins d'un nœud $u \in V$ est noté $\Gamma(u)$.

Orientation. Un graphe contenant des arêtes orientées, appelées *arcs*, est qualifié de *graphe orienté*. Une arête reliant un nœud u à un nœud v sera notée $\{u, v\}$, un arc liant ceux-ci sera noté (u, v).

Pondération. Le *poids* d'une arête est noté de manière standard, $\omega(u, v)$.

2.1.1.1 Chemins distances et diamètre

Un chemin est une liste de nœuds $P_{u,v}$ permettant d'aller d'un nœud u vers un nœud v dans le graphe G :

$$P_{u,v} = (u_1 = u, u_2, u_3, ..., u_l = v)$$

tel que pour tout $i \in [1, l-1]$, le nœud u_i est voisin du nœud u_{i+1}, autrement dit, l'arête $\{u_i, u_{i+1}\}$ appartient à E. Le poids total d'un chemin, plus communément

appelé *longueur du chemin*, est la somme des poids des arêtes qui le constituent :

$$\text{len}(P_{u,v}) = \sum_{i=0}^{l-1} \omega(u_i, u_{i+1})$$

Dans le cas non pondéré, la longueur du chemin $P_{u,v}$ est $\text{len}(P_{u,v}) = |P_{u,v}| - 1$. Notons $\mathcal{P}_{u,v}$ l'ensemble des chemins allant d'un nœud u à un nœud v. La *distance* entre un nœud u et un nœud v est :

$$d(u, v) = \min \{\text{len}(P_{u,v}) \mid P_{u,v} \in \mathcal{P}_{u,v}\}$$

La notion de *hop-distance* ou *distance en nombre de sauts* qui est le nombre de sauts minimal d'un plus court chemin entre u et v. La hop-distance plus formellement définie comme suit :

$$\text{hd}(u, v) = \min \{|P_{u,v}| - 1 \mid (P_{u,v} \in \mathcal{P}_{u,v} \wedge \text{len}(P_{u,v}) = d(u, v))\}$$

Le *diamètre* d'un graphe est la hop-distance maximale entre deux nœuds :

$$D = \max_{(u,v) \in V^2} \{\text{hd}(u, v)\}$$

Les termes diamètre et *hop-diamètre* seront utilisés indifféremment : à aucun moment dans cette thèse le terme diamètre ne se référera à la distance (classique) maximale entre deux nœuds.

2.1.2 Graphes aléatoires

2.1.2.1 Les types de modèles

La majorité des modèles de graphes aléatoires sont répartis dans trois grandes catégories. Ils sont utilisés pour générer/analyser des graphes ayant des propriétés proches de celles de graphes de terrains, notamment proche des propriétés du réseau des AS introduites dans la section 1.3.

1. **Les modèles à distribution de degré fixée.** Dans cette classe, la distribution des degrés des nœuds est laissée en paramètre, sous forme d'une liste de n degrés correspondant aux degrés respectifs des nœuds du graphe de sortie. L'algorithme de construction est simple, il crée n nœuds en leur attribuant les degrés passés en paramètre, puis il effectue un couplage entre les demi-arêtes de manière aléatoire uniforme. La distribution de degrés utilisée ne peut pas être totalement arbitraire, celle-ci doit permettre d'effectuer le couplage, par exemple la somme des degrés doit être paire.

2. **Les modèles à arêtes indépendantes.** Dans la classe précédente, les probabilités d'avoir une arête entre deux nœuds x et y n'est pas indépendante de la probabilité d'avoir une arête entre le nœud x et un autre nœud z. C'est pourquoi cette deuxième classe de générateurs de graphes aléatoires a vu le jour. Cette propriété rend l'analyse des propriétés structurelles des graphes de cette famille plus simple.

3. **Les modèles de graphes incrémentaux.** Ce type de modèle produit des graphes en partant d'un graphe initial G_0. Pour tout pas de calcul i, l'algorithme ajoute des arêtes et/ou des nœuds au graphe courant G_{i-1}. L'ajout d'arêtes se fait en considérant une loi de probabilité dépendante de l'ensemble des nœuds du graphe G_{i-1}.

Graphes géométriques (plan) aléatoires. Ce type de modèles de graphes est plutôt utilisé pour générer des graphes simulant des réseaux de capteurs, ou des réseaux radio. Ces graphes sont appelés UDG (*Unit Disk Graph*) et ont été introduits par [Gil61, CCJ90]. Le principe de génération de ces graphes est simple, n nœuds sont placés à des coordonnées choisies de manière aléatoire dans un carré unité $[0,1]^2$. Deux nœuds sont connectés par une arête si et seulement si ils sont à une distance inférieure à un paramètre R. Dans le cas d'un réseau radio, ce rayon R correspond à la portée d'émission des nœuds.

Attachement préférentiel. L'attachement préférentiel est la tendance à connecter les nœuds à d'autres nœuds de forte importance, selon un critère de préférence pouvant varier mais étant typiquement lié au degré des nœuds. L'attachement préférentiel permet de créer des graphes à mi-chemin entre les graphes totalement aléatoires et les graphes réguliers. L'attachement préférentiel est assez naturel à traduire dans les modèles incrémentaux, il suffit en effet d'attacher les nouveau nœud insérés à des nœuds existants de fort degré.

2.1.3 Quelques modèles de graphes aléatoires

Cette section présente les modèles (générateurs) de graphes utilisés dans cette thèse. Commençons par le modèle historique présenté sous deux formes différentes par Paul Erdős et Alfréd Rényi dans [ER59] ainsi que par Edgar Gilbert dans [Gil59] en 1959. Par simplicité ce modèle sera référencé par la suite comme étant le modèle d'ERDŐS-RÉNYI.

ERDŐS-RÉNYI [Forme de Gilbert] L'ensemble des graphes générés suivant ce modèle est usuellement dénoté $\mathcal{G}_{n,p}$. Ce modèle appartient à la catégorie des modèles à arêtes indépendantes. Dans un graphe du modèle ERDŐS-RÉNYI $G=(V,E)\in\mathcal{G}_{n,p}$, $|V|=n$ et tout couple de nœuds a une probabilité p d'être connecté par une arête :

$$\forall (u,v)\in V^2,\ \Pr\left(\{u,v\}\in E\right)=p$$

Voici quelques observations simples et communes sur ce modèle de graphe :

- Pour pn constant et supérieur à 1, la distribution des degrés suit une loi de poisson et le graphe contient une unique composante connexe de $\Theta(n)$ nœuds ;
- Tout graphe $G\in\mathcal{G}_{n,1/2}$ a un diamètre de 2 avec grande probabilité ;
- Pour toute probabilité $p<(1-\epsilon)\frac{\log n}{n}$, le graphe généré n'est pas pas connexe avec grande probabilité. Il est donc peu probable de pouvoir générer des graphes connexes de degré moyen plus petit que $(1-\epsilon)\log n$;
- Pour toute probabilité $p>(1+\epsilon)\frac{\log n}{n}$, le graphe généré est connexe avec grand probabilité.

Dans cette thèse deux paramétrages seront utilisés, soit $p=1/2$, soit $p=c\frac{\log n}{n}$ avec c une petite constante supérieure à un. D'après [BFdlV82] avec cette dernière configuration,le diamètre du graphe est avec grande probabilité inférieur à $\frac{\log n}{\log\log n}$.

Le problème de ce modèle de graphe est qu'il représente assez mal la distribution des degrés que l'on peut observer dans les graphes réels.

AIELLO-CHUNG-LU L'objectif de ce modèle présenté dans [ACL00] est de générer des graphes dont la distribution des degrés suit une loi en puissance. Il appartient à la catégorie des modèles à distribution de degré fixée. Il est paramétré par α, le logarithme de la taille du graphe et β, le paramètre de la loi de puissance. Plus précisément, la distribution des degrés suit la loi suivante (pour tout degré x il y a y nœud ayant ce degré) :

$$\log y = \alpha - \beta \log x$$

Autrement dit, la probabilité d'avoir un degré x est de $x^{-\beta}$, et donc :

$$|\{v \mid \text{degre}(v) = x\}| = y = \frac{e^\alpha}{x^\beta}$$

Aiello *et al.* prouvent également que, si $\beta < 3.48$, alors tout graphe généré avec ce modèle contient une unique composante connexe de $\Theta(n)$ nœuds. Dans les expériences, seuls des graphes connexes seront considérés, nous utiliserons donc toujours cette composante comme graphe d'entrée.

Le diamètre des graphes obtenus avec ce modèle n'est pas étudié dans cet article, mais est étudié dans un autre modèle très similaire qu'est RPLG.

RPLG Ce modèle, présenté dans [Lu01], appartient à la catégorie modèle à arêtes indépendantes. Un graphe $G = (V, E)$ généré par le modèle RPLG est paramétré par un vecteur de *poids* $\vec{w} = \{w_1, w_2, \ldots, w_n\}$. La distribution des poids dans $\vec{w}$ est définie comme suit,

$$\begin{cases} |\{i \mid k \leqslant w_i \leqslant k+1\}| = \left\lfloor \frac{e^\alpha}{k^\beta} \right\rfloor \text{ pour } k \in [2, e^{\frac{\alpha}{\beta}}] \\ |\{i \mid 1 \leqslant w_i < 2\}| = \lfloor e^\alpha \rfloor - r \end{cases}$$

avec $r = n - \sum_{k=2}^{e^{\frac{\alpha}{\beta}}} \left\lfloor \frac{e^\alpha}{k\beta} \right\rfloor$.

Tout nœud $v_i \in V$ se voit attribué un poids $w_i \in \vec{w}$. Pour tout couple de nœuds $\{v_i, v_i'\} \in V^2$ la probabilité qu'une arête existe entre v_i et v_i' est de

$$\Pr(\{v_i, v_i'\} \in E)) = \min\left\{ \frac{w_i \cdot w_i'}{\sum_{j \in [1,n]} w_j}, 1 \right\}$$

Pour ce modèle, la loi de distribution des degrés est prouvée dans [Lu01] être la même que pour le modèle AIELLO-CHUNG-LU en utilisant le fait qu'un nœud de poids w_i a une espérance de degré d'approximativement w_i [*] :

$$|\{v \mid \text{degre}(v) = x\}| \approx \frac{e^\alpha}{x^\beta}$$

Tout comme précédemment, e^α est la taille du graphe et β le facteur de croissance exponentiel. Remarquons que le degré maximum du graphe est donc de $e^{\alpha/\beta}$ soit $\mathcal{O}(e^\alpha)$ si $\beta > 1$. Enfin, le diamètre peut être donné pour certaines valeurs de β :

- $2 + \left\lceil \frac{1}{2-\beta} \right\rceil + 5$ si $\beta < 2$;
- $\Theta(\log n)$ avec grande probabilité si $\beta > 2$.

De même que précédemment, pour ce modèle le graphe n'est pas garanti connexe, mais contient une composante connexe de taille $\Theta(n)$.

[*] le degré d'un nœud est défini comme suit : $\sum_{j \in [1,n]} w_i w_j p \approx w_i$

GLP Ce modèle de graphe présenté dans [BT02] appartient à la classe des graphes incrémentaux, il est paramétré par (n_0, p). Au départ le graphe G_0 contient un nombre de nœuds fixé n_0 connectés par $n_0 - 1$ arêtes. À toute itération i, un nœud est ajouté à G_{i-1} et avec probabilité $1 - p$ il est attaché à x nœuds de G_{i-1}, avec probabilité p ce sont $x - 1$ arêtes qui sont ajoutées à G_{i-1}. L'attachement d'une arête à un nœud v du graphe se fait avec la probabilité suivante :

$$\Pr(\text{attachement à } v) = \frac{(\text{degre}(v) - \gamma)}{\sum_{w \in V} (\text{degre}(w) - \gamma)}$$

L'avantage de cette technique est qu'elle permet de générer des graphes connexes et est censés mieux refléter les propriétés des graphes issus de données réelles, tels que les cartes CAIDA.

2.2 Modèles d'adversaire

Lors de l'analyse d'algorithme, la notion d'adversaire est utile pour caractériser l'ensemble des exécutions possibles, dans le cas d'un algorithme distribué. L'adversaire choisit en respectant le modèle l'ordonnancement des activations de nœuds. L'adversaire peut également être vu comme un second algorithme dont le but est d'empêcher/de ralentir le calcul en choisissant l'ordre d'exécution le pire possible. Les adversaires considérés dans l'analyse d'algorithmes distribués sont caractérisés par leur connaissance des choix faits par l'algorithme [BDBK+94, BEY98] :

- Si l'adversaire connaît tous les choix fait par l'algorithme, il active les nœuds en fonction de ce qu'il observe pendant l'exécution, on dit qu'il est *adaptatif*.
- Si l'adversaire ignore les choix aléatoires faits par l'algorithme, il ne connaît que l'algorithme contre lequel il est en compétition : on dit que l'adversaire est oublieux (*oblivious* en anglais).

Ces deux types d'adversaires seront considérés dans la thèse.

2.3 Modèle de communication pour les systèmes distribués

La première composante du modèle distribué concerne la connaissance des nœuds sur leur propre identité au sein du réseau :

Définition 2.1 (Formes d'anonymats)**.** *Un modèle distribué est dit anonyme* uniforme*, si tous les nœuds exécutent exactement le même code. Il est qualifié de* quasi-uniforme *si un nœud particulier peut exécuter un code différent.*

Modèles de communication. Il existe différents modèles de communication dans les systèmes distribués, le modèle de mémoire partagée et le modèle de passage de messages seront les deux modèles considérés dans cette thèse. Cette section ne présentera que le modèle de passage de message qui sera utilisé dans les chapitres 3 et 4. Le modèle à état sera présenté dans l'unique chapitre utilisant ce modèle, le chapitre 5.

Équivalence des modèles. La simulation du modèle de mémoire partagée avec le modèle de passage de message est présenté dans [ABND95].

Passage de messages. Les modèles à passage de message utilisés dans cette thèse correspondent aux modèles $\mathcal{LOCAL}$ et $\mathcal{ASYNC}$ présentés par Peleg dans [Pel00]. Chaque processus possède deux primitives pour échanger des informations avec ses voisins et est capable d'effectuer un calcul localement. Les primitives de communication sont les suivantes :

- Un processus $u \in V$ est capable d'envoyer un message à l'ensemble de ces voisins.
- Un processus $u \in V$ est capable d'envoyer un message à un voisin ciblé s'il connaît le numéro de port de sortie correspondant au voisin. Envoyer un message ciblé implique donc un coût en mémoire.

Dans le modèle classique, les liens de communication sont fiables, tout message émis arrive à sa destination en un temps fini. Le temps de propagation des messages et l'instant d'émission sont variables et dépendent du modèle de synchronisme considéré. De plus la quantité d'information envoyé par un lien à un instant donné n'est pas borné par le modèle, la taille des messages fera donc partie complexités mesurées. Dans la thèse la problématique de la congestion ne sera pas abordé, le troisième modèle $\mathcal{CONGEST}$ présenté par Peleg *et al.* [Pel00] ne sera pas utilisé.

2.3.1 Communication synchrone, modèle $\mathcal{LOCAL}$

Dans le modèle synchrone, tous les processus effectuent l'ensemble de leurs envois simultanément, ils sont synchronisés. De plus, le temps de propagation des messages est uniforme. Tous les nœuds recevront donc simultanément un ensemble de messages entrant. Enfin, ce modèle considère que les nœuds sont "réveillés" tous en même temps, ils exécutent leur code pour la première fois simultanément.

2.3.2 Communication asynchrone, modèle $\mathcal{ASYNC}$ et Δ-borné

Dans le modèle $\mathcal{ASYNC}$, tout message émis par un nœud arrive à destination en un temps fini, mais non uniforme. Le modèle le plus classique considère également que les liens de communication sont FIFO (de l'anglais "*First in First Out*"), ce qui implique que deux messages traversant un lien de communication partiront et arriveront dans le même ordre. Le temps de traversée d'un lien de communication par un message dépend de la *latence* du lien. Quel que soit le modèle de communication considéré, la latence maximale d'un lien sera fixé à 1. Toute configuration de latences pour un réseau donné peut être normalisé vers une configuration de latences choisies dans l'ensemble $]0, 1]$.

Modèle Δ-borné. Une variante du modèle $\mathcal{ASYNC}$ considère que les liens de communication ont une latence non seulement finie, mais bornée. Ce modèle est caractérisé par le ratio entre la latence maximale et la latence minimale des liens de communications. Usuellement, la latence minimale est fixée à 1 et la latence maximale à Δ, d'où le nom qui est donné à ce modèle : *asynchrone Δ-borné*. En normalisant les latences vers $]0, 1]$, les latences dans ce modèle appartiennent à $[\frac{1}{\Delta}, 1]$.

Initialisation de l'algorithme. Dans les modèles $\mathcal{ASYNC}$ et Δ-borné, les processus ne sont pas nécessairement réveillés simultanément au début de l'exécution de l'algorithme. À l'initialisation, au moins un processus commence son exécution. Un

nœud non réveillé se réveille lorsqu'il reçoit son premier message, il est donc possible en fonction de l'algorithme considéré que certains nœuds ne soient pas réveillés.

Remarque. Il est important de noter la différence entre le modèle asynchrone 1-borné et le modèle synchrone. En effet, dans le modèle 1-borné deux messages en provenance de deux voisins distincts et émis en même temps arrivent simultanément. Cependant, rien dans ce modèle ne stipule que les messages sont émis au même moment. Même dans ce cas particulier il est donc possible d'observer les effets indésirables de l'asynchronisme.

2.4 Mesures de performances

2.4.1 Complexité en temps, ou temps de convergence

Pour définir la mesure du temps il est d'abord nécessaire de définir la notion de *ronde*. Informellement, une ronde est une période de temps durant laquelle tout nœud a eu le temps d'effectuer au moins une fois une séquence de réception/exécution/émission. D'après les définitions de latences données dans la section 2.3 cette période de temps est d'au maximum une unité (nous considérerons que les latences sont normalisées vers $]0, 1]$).

- **Ronde synchrone.** Dans le modèle synchrone une ronde se termine lorsque tous les nœuds pouvant s'exécuter se sont exécutés.
- **Ronde asynchrone.** Dans le cas asynchrone la notion de ronde est un peu moins naturelle, une ronde d'une exécution donnée se termine lorsque :
  - tous les messages envoyés avant le début de cette ronde sont arrivés à leurs destinations respectives ;
  - tous les messages reçus pendant la ronde sont traités ;
  - et tous les messages devant être émis sont émis.

  Tous les calculs locaux et les émissions sont considérés immédiats, le délai entre le début d'une ronde et la fin d'une ronde dépend donc uniquement des latences des liens de communication. Autrement dit, dans le modèle asynchrone, lors d'une ronde tout nœud à qui un message est adressé a le temps de traiter ce message.

Le temps de convergence d'un algorithme A sur un graphe G est noté $\mathsf{Time}(\mathrm{A}, G)$, chacun de ces paramètres pouvant être omis si le contexte le permet. Ce temps correspond au nombre de rondes nécessaires à l'algorithme A pour converger vers un état stable, plus aucun message n'est envoyé par les nœuds. Le temps d'origine considéré dans les analyse est le suivant :

Définition 2.2 (Temps d'origine)**.** *Le temps d'origine pour les calculs de complexité en temps est l'instant auquel tout nœud a été réveille, i.e. le temps auquel tout nœud a reçu ou émis au moins un message.*

Remarque. La différence avec le temps auquel le premier nœud a été réveille ne diffère que d'au plus D unités de temps. Il serait donc très similaire de considérer que le temps d'origine pour les calculs de complexité en temps, correspond au temps auquel le premier nœud est réveille.

2.4.2 Complexité mémoire

Pour un algorithme A sur un graphe G, la mesure de la mémoire se fait par nœud :

- La mémoire utilisée pour le stockage des tables de routage. Cette mémoire sera notée $|\mathsf{Table}(\mathsf{A},\mathsf{G})|$.
- La mémoire utilisée pour calculer les tables de routage, la *mémoire de travail*, notée $\mathsf{Mem}(\mathsf{A},\mathsf{G})$. Cette mémoire comprend la taille des tables et donc $\mathsf{Mem}(\mathsf{A},\mathsf{G}) \geqslant |\mathsf{Table}(\mathsf{A},\mathsf{G})|$.

Ces deux mesures se feront en nombre d'entrées de routage. La taille maximale des entrées $|\mathsf{Entrées}(\mathsf{A},G)|$, donnée en nombre de bits, sera utilisée en complement de cette première mesure. Le nombre maximum de bits utilisés pour un nœud donné est donc borné par $|\mathsf{Entrées}(\mathsf{A},G)| \cdot \mathsf{Mem}(\mathsf{A},\mathsf{G})$.

2.4.3 Coût de communication

Le coût de communication d'un algorithme A sur un graphe G correspond au nombre d'entrées (de taille $|\mathsf{Entrées}(\mathsf{A},G)|$) total échangées entre les nœuds pour arriver dans l'état final. Dans le cas synchrone cette mesure sera notée $\mathsf{Msg_s}(\mathsf{A},G)$ et dans le cas asynchrone elle sera notée $\mathsf{Msg_a}(\mathsf{A},G)$. Dans le cas où un message de taille $|M|$ supérieure à la taille d'une entrée est échangé, le coût de communication correspondant à ce message est de $\left\lceil \frac{|M|}{|\mathsf{Entrées}(\mathsf{A},G)|} \right\rceil$.

La prochaine section présente des paradigmes permettant la création d'algorithme $\mathcal{ASYNC}$ à partir d'algorithmes décrits dans le modèle $\mathcal{LOCAL}$. Cette section permet d'introduire les résultat de la littérature concernant les algorithmes de routage (section 2.6).

2.5 Outils de transformation d'algorithmes distribués

Les paradigmes présentés dans cette section sont utilisés pour transformer un algorithme distribué en un second algorithme ayant des propriétés différentes (complexités et tolérance à l'asynchronisme). Par exemple, un algorithme distribué synchrone peut être utilisé dans un environnement asynchrone en utilisant un synchroniseur avec un sur-coût sur la communication et la complexité en temps. Cette section ne donne pas les techniques utilisées pour effectuer ces traductions et expose uniquement les implications de ces transformations sur les complexités.

2.5.1 Synchroniseurs

Les deux méthodes de transformation suivantes sont décrites par Awerbuch dans [Awe84]. Le temps et le coût de communication sont impactés par le temps et le coût de communication nécessaires à l'initialisation du synchroniseur ainsi qu'à un surcoût supplémentaire lié à chaque ronde de calcul.

α-Synchroniseurs. L'application d'un α-synchroniseur sur un algorithme A produit un algorithme α-sync(A) ayant les complexités suivantes :

$$\mathsf{Time}(\alpha\text{-sync}(\mathsf{A})) = D + 3 \cdot \mathsf{Time}(\mathsf{A})$$
$$\mathsf{Msg_a}(\alpha\text{-sync}(\mathsf{A})) = m + \mathsf{Msg_s}(\mathsf{A}) + m \cdot \mathsf{Time}(\mathsf{A})$$

β-Synchroniseurs. L'application d'un β-synchroniseur sur un algorithme A produit un algorithme β-sync(A) ayant les complexités suivantes :

$$\mathsf{Time}(\beta\text{-sync(A)}) = D + D \cdot \mathsf{Time(A)}$$
$$\mathsf{Msg_a}(\beta\text{-sync(A)}) = m + \mathsf{Msg_s(A)} + n \cdot \mathsf{Time(A)}$$

2.5.2 Sparsers

Remarquons qu'il est possible de simuler un algorithme centralisé dans un modèle distribué en envoyant toutes les informations sur la topologie du réseau à un unique nœud qui se chargera d'effectuer le calcul centralisé. Puis, une fois ce calcul effectué le nœud diffusera le résultat du calcul centralisé. La technique proposée par Afek *et al.* dans [AR93] se place à mi-chemin entre cette simulation d'algorithme centralisé et un algorithme purement distribué. L'application d'un sparser sur un algorithme A ayant un coût de communication de $\mathsf{Msg_a(A)} = f \cdot m$ produit un algorithme sparser(A) ayant les complexités suivantes :

$$\mathsf{Time(sparser(A))} = \log n \cdot \mathsf{Time(A)}$$
$$\mathsf{Msg_a(sparser(A))} = f \cdot n \log n + m \log n$$

2.6 Routage

Définition 2.3 (Source routing)**.** *Le routage de plus court chemin d'un nœud $u \in V$ vers un nœud $v \in V$ se fait de deux façons. Soit il existe un plus court chemin $P = (u_0, u_1, ..., u_p = v)$ tel que pour tout $i \in [0, p-1]$ le nœud u_i sait router vers v via u_{i+1}. Dans ce cas on parle de* routage pas par pas*. Soit il n'existe aucun plus court chemin de ce type, il est alors nécessaire d'utiliser une technique différente qu'on appelle* routage à la source*,* source-routing *en anglais. Dans ce cas, le nœud source envoie dans l'entête de routage le chemin complet de u à v.*

Les arbres sont très utilisés par les algorithmes de routage (compact) et occupent donc une place importante dans cette thèse. Cette section présente les résultats principaux existants pour le calcul d'arbres et leur utilisation pour le routage compact. Rappelons tout d'abord qu'il est connu que le temps et le nombre de messages nécessaires au calcul d'un arbre de plus court chemin sont respectivement d'au moins D et n.

2.6.1 Calcul d'arbres de plus court chemin

Le calcul d'arbre de plus court chemin se fait soit pour une racine donnée, un nœud $r \in V$, soit pour l'ensemble des racines. Ces deux problèmes sont différents, en effet, le moyen optimal de calculer n arbres de plus court chemin dans un même graphe n'est pas nécessairement d'appliquer n fois le meilleur algorithme permettant le calcul d'un unique arbre sur ce même graphe. La table 4.1 résume les complexités dues à la construction d'arbres de plus court chemin. On remarque que l'usage de la synchronisation peut peut réduire les coûts de communication.

Remarque sur la notation $\tilde{\mathcal{O}}(.)$ La notation $\tilde{\mathcal{O}}(f(x))$, sera utilisée pour simplifier la lecture. Si $g(x) = \tilde{\mathcal{O}}(f(x))$ alors la fonction g est asymptotiquement bornée par f à un facteur multiplicatif $\log^c(f(x))$ près, avec c constant. Par exemple, $\tilde{\mathcal{O}}(n)$ est équivalent à $\mathcal{O}(n \log n)$, de la même façon $\tilde{\mathcal{O}}(\log n)$ est équivalent à $\mathcal{O}(\log n \cdot \log\log n)$.

Algorithme	Temps	Coût de communication		Référence
		Synchrone	Asynchrone	
$n\times$DVECTOR	$\mathcal{O}(D)$	$\mathcal{O}(nm)$	$\mathcal{O}(n^2m)$	[Gal82, Seg83]
α-sync(DVECTOR)	$\mathcal{O}(D)$	$\mathcal{O}(nm+Dm)$	$\mathcal{O}(nm+Dm)$	[Awe84]
β-sync(DVECTOR)	$\mathcal{O}(D^2)$	$\mathcal{O}(nm+Dn)$	$\mathcal{O}(nm+Dn)$	[Awe84]
sparser(DVECTOR)	$\tilde{\mathcal{O}}(D)$	$\tilde{\mathcal{O}}(n^2)$	$\tilde{\mathcal{O}}(n^2)$	[AR93]
DFS	$\mathcal{O}(n)$	$\mathcal{O}(m)$	$\mathcal{O}(m)$	[Awe85]
Bornes inférieures	$\Omega(D)$	$\Omega(n^2)$	$\Omega(n^2)$	th. 3.1,3.2

Table 2.1: Comparatif d'algorithmes distribués pour calculer n arbres de plus court chemin dans le modèle $\mathcal{LOCAL}$ et pour des graphes non-pondéré. Plus précisément on souhaite avoir un arbre enraciné en chaque nœud du graphe

Pour en revenir aux objectifs de la thèse, le but principal sera de décrire un schéma de routage garantissant un temps de convergence $\mathcal{O}(D)$ avec un coût de communication de $\mathcal{O}(n^2)$ messages.

2.6.1.1 Calculer un arbre de plus court chemin

L'algorithme le plus commun pour calculer un arbre de plus court chemin est un simple Bellman-Ford (BFS) distribué, appelé *vecteur de distance/vecteur de chemin* (variante permettant le routage à la source), abrégé DVECTOR et également connu sous le nom de $\min+1$. Cet algorithme a est décrit dans [Gal82, Seg83] et se trouve également sous le nom de DBFS dans la littérature. Voici une description non formelle de DVECTOR :

> **Algorithme DVECTOR :** Pour un arbre enraciné en un nœud r, tout nœud u stocke une métrique de distance d_u et un nexthop nexthop_u, i.e. l'identifiant du voisin lui ayant appris cette distance (dans la variante vecteur de distance le nexthop est remplacé par un chemin complet vers la destination). Un nœud change sa métrique de distance s'il reçoit un message d'un voisin v contenant une distance d telle que $d+1<d_u$, d'où l'appellation $\min+1$. Dès qu'un nœud change de distance à la racine il envoie un message à ses voisin contenant sa nouvelle distance.

Cet algorithme à un coût de communication de $\mathsf{Msg_s}(\text{DVECTOR}) = \mathcal{O}(m)$ et de $\mathsf{Msg_a}(\text{DVECTOR}) = \mathcal{O}(mn)$.

2.6.1.2 Calculer tous les arbres de plus court chemin (*All Pairs Shortest Path problem*)

L'algorithme naïf pour effectuer ce calcul est d'initier un algorithme DVECTOR par nœud. Le coût de communication de cet algorithme est de $\mathsf{Msg_s}(n\times\text{DVECTOR}) = \mathcal{O}(nm)$. Il est relativement simple, en utilisant le graphe proposé en figure 2.1, de montrer que pour le lemme suivant :

Lemme 2.1. *Pour tout graphe H de n nœuds et m arêtes, il existe un graphe G, composé du graphe H et contenant $\mathcal{O}(n)$ nœuds $\mathcal{O}(m)$ arêtes, tel que* $\mathsf{Msg_a}(n \times \text{DVECTOR}) = \Omega(n^2 m)$.

Preuve. Notons n et m le nombre de nœuds et d'arêtes d'un graphe H connexe. Utilisons également la construction présentée en figure 2.1 pour cette preuve. Sur cette figure, nous considérons que les arêtes bleues sont activées dans l'ordre indiqué sur chacune d'entre-elles et que les autres arêtes sont activées avec une fréquence qui tend vers l'infini (latence tend vers 0). Dans cette exécution, pour tout $i \in [0, n]$ la distance connue par u pour le nœud v_i va changer i fois. Donc, pour l'ensemble des nœuds v_i, l'état du nœud u va changer de l'ordre de n^2 fois. À chaque changement d'état, le nœud u envoie un message à l'ensemble de ses voisins, en particulier à tous les nœuds du sous-graphe quelconque H. Tout nœud w appartenant à H va donc changer d'état $\Omega(n^2)$ fois et enverra donc $\text{degre}(w)$ messages. En sommant sur les degrés des nœuds de H le nombre total de messages envoyés pour le calcul des distances des nœuds du sous-graphe H à l'ensemble des nœud v_i est de $\Omega(n^2 \cdot m)$.

Remarquons que pour construire le graphe au complet nous avons utilisé seulement $2n$ arêtes et nœuds supplémentaires, le nombre total d'arêtes dans cette construction est donc de $\mathcal{O}(m)$ et le nombre de nœuds de $\mathcal{O}(n)$ ($m \geqslant n$ car H est connexe). ∎

Dans le cas asynchrone et pondéré il est connu que le nombre de messages peut être exponentiel. Le cas asynchrone et pondéré ne sera pas au centre de l'attention de cette thèse.

En utilisant un β-synchroniseur, l'algorithme DVECTOR dans le modèle asynchrone a une complexité de communication de $\mathsf{Msg_a}(n \times \text{DVECTOR}) = \mathcal{O}(nm + Dm)$. Le meilleur algorithme existant pour ce problème est obtenu en utilisant la technique de sparser appliquée à l'algorithme DVECTOR classique, cet algorithme a une complexité en nombre de messages de $\mathsf{Msg_a}(Sparser(\text{DVECTOR})) = \tilde{\mathcal{O}}(n^2)$.

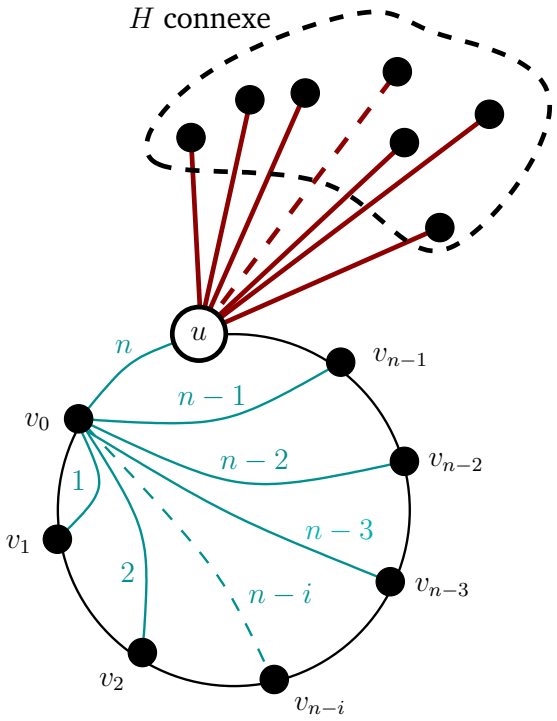

Figure 2.1: Preuve de la borne inférieure pour la construction de n arbres de plus court chemin en utilisant l'algorithme DVECTOR, quel que soit le graphe H.

2.6.2 Routage compact étiqueté dans les arbres

Les techniques décrites dans cette section ([FG01a] et [TZ01]) permettent de compresser des chemins dans les arbres et ainsi permettre de faire du routage à la source de plus court chemin dans les arbres en ayant une borne sur la taille des entrées stockées. En effet, stocker un chemin peut impliquer d'avoir des entrées de taille $\mathcal{O}(n)$, avec cette technique elle peuvent être réduite à une taille poly-logarithmique. L'utilisation de ces chemins compressés, que nous appellerons étiquettes de routage, n'induisent pas de détours lors du routage.

Ces techniques de routage compact sont utilisées comme base dans les algorithmes de routage compact universels tels que [TZ01, AGM+08]. Une autre caractéristique intéressante de ces algorithmes est que toute décision de routage se fait en temps constant.

Considérons un arbre T_r enraciné en un nœud r. Dans ces deux algorithmes, chaque nœud $u \in T_r$ connaît une version compressée du chemin $P_{u,r}$. Ce chemin est stocké sous forme d'étiquette de routage $\ell(u, T_r)$. Cette étiquette sera utilisée à la fois :

- comme identifiant de destination pour une requête de routage vers le nœud u ;
- et comme "table de routage" du nœud u.

Autrement dit, la décision de routage au nœud u pour une destination v d'identifiant $\ell(v, T_r)$ est prise en utilisant uniquement $\ell(u, T_r)$ et $\ell(v, T_r)$. Ces deux algorithmes sont quasiment équivalents et proposent les complexités suivantes :

- **Thorup et Zwick**, requiert des étiquettes de routage de $|\mathsf{Entrées}(TZ - labeling)| = \log(n) \cdot (1 + \mathcal{O}(1))$ bits ;
- **Fraigniaud et Gavoille** quant à eux font la différence entre deux modèles de numérotation des ports et la taille des étiquettes de routage est de :
  - $|\mathsf{Entrées}(\mathsf{FG}\text{-}\textsc{fixedPort})| = 5 \log n + \mathcal{O}(\log n)$ bits si les ports sont dans $[1, \mathrm{degre}]$
  - $|\mathsf{Entrées}(\mathsf{FG})| = \mathcal{O}\left(\frac{\log^2 n}{\log \log n}\right)$ sans a priori sur les noms des ports.

Remarquons également que ces deux algorithmes de routage sont décrits de manière centralisée.

Schéma de routage compact distribué 3

Sommaire

Ce chapitre décrit un algorithme distribué et asynchrone qui a des garanties d'étirement quelque soit le graphe considéré, tout en ayant un coût de communication plus faible qu'un algorithme de plus court chemin Bellman-Ford distribué. Il garantit également, dans le cas synchrone, que le coût de communication est meilleur que tout algorithme de plus court chemin sur des graphes sans-échelle. Le gain observé est d'un facteur de l'ordre de $\sqrt{n}$. La question principale de ce chapitre tourne autour des compromis théoriques envisageables avec une mémoire, un temps de convergence et un coût de communication restreints. Il n'existe à l'heure actuelle aucun algorithme distribué avec indépendance de noms, donnant un étirement borné avec une mémoire de travail sous-linéaire.

Ce chapitre présente tout d'abord les résultats principaux concernant les algorithmes de routage compact, en section 3.1, ainsi que le détail des résultats obtenus, en section 3.2. Par la suite, sont présentés deux bornes inférieures, une première sur le temps de convergence de tout algorithme d'étirement borné, en section 3.3.1, et une seconde sur le coût de communication minimal pour tout algorithme de plus court chemin, en section 3.3.2. Dans la suite du chapitre, la section 3.4.1 présente l'algorithme de routage centralisé avec indépendance des noms AGMNT [AGM+08], qui a inspiré l'algorithme distribué présenté. La section 3.4 présente le premier algorithme distribué de routage compact avec indépendance des noms ayant un étirement garanti borné par 5 pour tout graphe. Cet algorithme utilise utilise le même

protocole de routage que celui d'AGMNT. Les techniques utilisées dans cette section pour distribuer le calcul des tables est intéressant de manière plus générale dans le sens où les techniques utilisées dans les différents algorithmes compact centralisé utilisent des construction très similaire. La section 3.10 présente l'analyse générale de l'algorithme de routage distribué et utilise les propriétés présentées tout au long des sections 3.5, 3.6, 3.7 et 3.8.

3.1 Les algorithmes de routage compact avec indépendance des noms de la littérature

Les algorithmes de routage compacte décrits dans la littérature sont soit centralisés ou distribués mais considèrent un environnement synchrone. Il est néanmoins intéressant d'observer la mémoire des algorithmes centralisé car avoir une complexité mémoire faible permettrait intuitivement de proposer un algorithme utilisant moins de messages dans des modèles distribués tels que $\mathcal{ASYNC}$ ou $\mathcal{LOCAL}$. Commençons donc par présenter les résultats principaux en ne considérant que la mémoire et l'étirement.

Algorithmes universels. Le premier algorithme présentant un compromis entre l'étirement et la mémoire est celui de Kleinrock [KK77]. Cependant, cet algorithme n'est pas universel et requiert qu'il existe un partitionnement donné du réseau. La première étude d'algorithme universel de routage ayant une mémoire sous-linéaire se trouve dans [PU89]. Dans cet article Peleg *et al.* présentent un algorithme utilisant $\mathcal{O}\big(k^3 \cdot n^{1+1/k}\big)$ bits de mémoire pour l'ensemble des n tables de routage et garantissant un étirement borné par $12k+3$. Cependant, cet algorithme n'est valable que dans un graphe non-pondéré. Dans [ABNLP90] Awerbuch *et al.* présentent un algorithme dans le modèle pondéré, utilisant $\tilde{\mathcal{O}}\big(k \cdot n^{1+1/k}\big)$ bits de mémoire au total et garantissant que l'étirement maximum est de $2^k - 1$. Dans [ACL+06], le premier algorithme garantissant une mémoire bornée par $\tilde{\mathcal{O}}\big(n^{2/k}\big)$ pour chaque nœud est décrit. Cet algorithme a une garantie d'étirement de $\mathcal{O}\big(k^2\big)$ et considère des poids polynomiaux en n sur les arêtes. Autrement dit, cet algorithme permet d'obtenir un étirement garanti inférieur à 3 avec une mémoire aux nœuds de $\tilde{\mathcal{O}}\big(n^{2/3}\big)$, ou encore un étirement de 5 avec une mémoire de $\tilde{\mathcal{O}}(\sqrt{n})$. Enfin, dans [AGM06], Abraham *et al.* proposent un algorithme universel valable pour toute pondération du graphe. Cet article présente à la fois un algorithme garantissant un étirement borné de $\mathcal{O}(k)$ avec une mémoire par nœud de $\tilde{\mathcal{O}}\big(n^{1/k}\big)$. Ainsi qu'une borne inférieure montrant qu'il existe pour tout k, un arbre de profondeur 1 avec des poids dans $\{1, k\}$ tel que tout algorithme de routage avec indépendance des noms, ayant un étirement $< 2k+1$, requiert une mémoire de $\Omega((n \log n)^{1/k})$ bits par nœud. D'après cette borne inférieure, tout algorithme de routage d'étirement inférieur à 5 nécessite des tables de routage de $\Omega(\sqrt{n \log n}\,)$ bits. De plus, pour tout algorithme de routage compact avec des tables de routage de $o(n \log n)$ bits il existe un graphe dans lequel cet algorithme ne pourra pas garantir un étirement strictement inférieur à 3. Un schéma de routage atteignant cet étirement minimal est proposé dans [AGM+08]. Celui-ci utilise des tables de routage de $\tilde{\mathcal{O}}(\sqrt{n})$ bits avec un étirement de 3. Cet algorithme est donc optimal en étirement et en mémoire dans la catégorie des algorithmes universels.

Algorithmes dédiés. Pour des classes de graphe plus restreintes, il est possible d'obtenir de meilleurs compromis étirement/mémoire. Notamment, les articles suiv-

ants proposent des algorithmes avec indépendance des noms, étirements constants et mémoire poly-logarithmique : dans les graphes d'expansion bornée [AM05], pour des graphes de dimension doublante bornée [AGGM06, KRX06], pour des arbres [Lai07] ou encore pour des graphes planaires [AGMW10]. Tang *et al.* proposent dans [TZLL12] un algorithme efficace dans les graphes de loi en puissance. Cet algorithme est analysé théoriquement et expérimentalement pour cette famille de graphe. La borne théorique sur la taille des tables de routage est avec grande probabilité $\tilde{\mathcal{O}}(\sqrt{n})$ et la longueur maximale d'une route pour deux nœuds u et v est bornée par $2d(u,v) + 2D_L$ avec D_L dépendant du graphe et constant pour les graphes en loi de puissance. Dans le cas général, pour toute famille de graphe, cet algorithme garantit le routage, mais la taille des tables et l'étirement ne sont pas bornés.

3.1.1 Dans le modèle distribué

Pour utiliser les techniques de routage compact en pratique il est nécessaire de proposer des implémentations distribuées. Parmi les algorithmes déjà présentés, celui de [ABNLP90] est distribué sous le modèle $\mathcal{LOCAL}$ et utilise une mémoire de travail de $\tilde{\mathcal{O}}\big(\mathrm{degre} + n^{1/k}\big)$ pour un nœud de degré degre. D'autres études sur ce modèle ont été faites, notamment par Ankit *et al.* dans [SGF+10] qui propose un algorithme d'étirement borné 7 avec des tables de routage de taille $\tilde{\mathcal{O}}(\sqrt{n})$ dans un modèle de graphe dynamique. Cependant le coût de communication de cet algorithme n'est pas borné de manière analytique. De plus, chaque entrée peut être de taille $\Omega(D)$ bits et la mémoire de travail d'un nœud peut être de $\Omega(\mathrm{degre}\sqrt{n})$ pour un nœud de degré degre. Pour des graphes non-pondérés, en utilisant des graphes couvrants peu denses, comme dans [Elk05, EZ06] il est possible d'avoir un étirement proche de 1 avec une complexité de communication dans le modèle $\mathcal{LOCAL}$ de $\tilde{\mathcal{O}}\big(mn^{\epsilon_1} + n^{2+\epsilon_2}\big)$ avec $0 < \epsilon_1, \epsilon_2 < 1$ deux constantes influençant la valeur de l'étirement.

En résumé. Il n'existe à l'heure actuelle aucun algorithme distribué avec indépendance des noms donnant un étirement borné avec une mémoire de travail sous-linéaire dans le modèle $\mathcal{ASYNC}$.

3.2 Résumé des résultats

Ce chapitre présente un nouvel algorithme de routage compact distribué, DCR, ainsi que deux bornes inférieures concernant le routage de manière générale. Rappelons tout d'abord que le modèle considéré est le modèle $\mathcal{ASYNC}$ présenté en section 2.3 et que l'adversaire considéré est oublieux, il ne connaît notamment pas le choix de couleurs fait par les nœuds.

- L'algorithme présenté est asynchrone, distribué et avec indépendance des noms. Il est conçu pour les graphes pondéré de n nœuds et de diamètre D. L'étirement cet algorithme est de 5 et la complexité en temps de $\mathcal{O}(D)$, avec une petite constante cachée (12). De plus, à tout instant de la construction des tables de routage, le schéma de routage requiert une mémoire de travail de $\tilde{\mathcal{O}}(\sqrt{n})$ à chaque nœud. En particulier, les tables de routage ont une taille de $\tilde{\mathcal{O}}(\sqrt{n})$. Enfin, la complexité en nombre de messages est de $\tilde{\mathcal{O}}\big(m\sqrt{n} + n^{3/2}\min\{D, \sqrt{n}\}\big)$ dans le modèle $\mathcal{LOCAL}$ avec des messages de taille poly-logarithmique.

- Pour le cas réaliste des graphes sans-échelle (diamètre et degré moyen polylogarithmiques), la complexité de message est de $\tilde{\mathcal{O}}(n^{3/2})$. La table 3.1 présente un résumé des complexités.

Schéma	Étirement	Mémoire	#Messages	Temps	Référence
Vecteur de distance	1	$\Omega(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(D)$	
DCR	5	$\tilde{\mathcal{O}}(n^{1/2})$	$\tilde{\mathcal{O}}(n^{3/2})$	$\mathcal{O}(D)$	cor. 3.1
Bornes inférieures					
Mémoire	$< 2k+1$	$\Omega((n\log n)^{1/k})$	-	-	[AGM06]
Communication	1	-	$\Omega(n^2)$	$\mathcal{O}(n)$	th. 3.2
Temps	$\leqslant n/(3D)$	-	-	$\Omega(D)$	th. 3.1

Table 3.1: Schémas distribués rapides avec indépendance des noms pour des graphes sans-échelle, i.e., avec $\tilde{\mathcal{O}}(n)$ arêtes et $D = \log^{\mathcal{O}(1)} n$. La colonne mémoire contient les résultat pour la mémoire de travail (et donc la taille des tables). Les bornes inférieures sont données en nombre de bits, aussi bien pour la mémoire que pour le coût de communication (complexité bit-messages).

Les bornes inférieures montrent qu'un temps $\Omega(D)$ est requis par tout schéma de routage d'étirement constant et que tout algorithme de plus court chemin requiert que le coût de communication en nombre de bits échangés soit de $\Omega(n^2 \log n)$. Cette dernière complexité est également valable pour les graphes sans-échelle. Plus précisément :

(1) Tout algorithme distribué synchrone d'étirement constant et avec indépendance des noms a une complexité en temps de $\Omega(D)$ pour un graphe non-pondéré de diamètre D. Cette borne est indépendante du coût de communication du schéma de routage et de sa mémoire de travail.

(2) Il existe des graphes non pondérés de n nœuds, de diamètre $\mathcal{O}(\log n)$ et de degré maximum 3 pour lesquels tout algorithme de plus court chemin distribué synchrone (même étiqueté) a un coût de communication de $\Omega(n^2 \log n)$ et ce même si la complexité en temps est $\mathcal{O}(n)$.

Pour ces deux bornes inférieures, nous considérons un scénario synchrone ces résultat tiennent donc bien évidemment dans un scénario asynchrone. Rappelons également qu'aucune restriction n'est faite sur la taille des messages (de la même manière que pour les analyses de bornes supérieures).

3.3 Bornes inférieures

3.3.1 Temps de convergence - Borne inférieure

Cette section donne la preuve qu'un temps $\Omega(D)$ est nécessaire à la construction distribuée de tables de routage permettant un routage en étirement constant. Cette preuve est valable quel que soit le nombre de messages et la mémoire de travail des nœuds.

Il est relativement simple de montrer que cette borne est valable pour du routage de plus court chemin. Considérons un chemin de longueur impair, ainsi qu'une requête de routage traitée par le nœud équidistant des deux extrémités et à destination d'une des extrémités. Si l'algorithme de construction des tables de routage a convergé en temps $\mathcal{O}(D)$, alors le nœud central ne connaît aucune des extrémité et ne sera donc pas capable de déterminer localement dans quel sens aller, l'étirement sera donc strictement supérieur à 1.

Une (d,k)-étoile obtenu en remplaçant chaque arête d'une étoile composée de d par un chemin de longueur k. La racine de cette (d,k)-étoile est le centre de l'étoile de de d nœuds utilisée pour la construire.

Théorème 3.1. *Tout algorithme de routage synchrone avec indépendance des noms pour la famille des (d,k)-étoile convergeant en temps $t < k$, produit au moins une route de longueur supérieure à $(2d-1)(k-t)+t$.*

En particulier, pour un graphe non-pondéré de de diamètre $D \in [2, n-1]$ et pour un algorithme avec indépendance des noms, d'étirement au plus $\frac{1}{3}n/D$, le temps de convergence est de $\Omega(D)$ dans le modèle $\mathcal{LOCAL}$.

Première partie du théorème 3.1

Preuve. L'intuition derrière cette preuve est que pour décider depuis la racine d'une (d,k)-étoile dans quelle branche se trouve la destination il est nécessaire de visiter au moins $k-t$ nœuds de la branche correspondante. Donc, en visitant strictement moins de $d-1$ branches, il n'est pas possible de savoir dans quelle branche se trouve la destination. Intéressons nous maintenant à une preuve formelle de ce théorème :

Considérons la famille $\mathcal{F}$ des (d,k)-étoiles dont les feuilles sont étiquetées avec des identifiants distincts dans $[1,d]$, les autres nœuds ayant des identifiants arbitraires. Pour une étoile G donnée, la feuille de la i-ème branche est notée b_i. Le nœud à distance t de b_i sur cette branche est noté a_i. L'ensemble de ces nœuds est noté $A = \bigcup_{i\in[1,d]} \{a_i\}$. Les notations utilisées dans cette preuve sont résumées en figure 3.1.

Observons l'algorithme de routage synchrone R convergeant en temps t sur cette famille de graphe et plus particulièrement le routage depuis la racine vers toutes les feuilles. Par contradiction, considérons que cet algorithme R soit capable de produire une route de longueur $L(R) < (2d-1)(k-t)+t$ pour tout routage de la racine vers une feuille. L'algorithme R est choisi comme étant l'algorithme optimal pour ces routages, autrement dit, l'algorithme de routage R minimise $L(R)$.

Pour une route P produite par R, ayant une longueur $|P| = L(R)$ et une destination b_i d'identifiant x dans une étoile $G \in \mathcal{F}$. Notons $\overline{P}$ le plus court préfixe de P contenant a_i, alors :

Proposition 3.1. $|\overline{P}| = |P| - t$.

En effet, quel que soit l'algorithme R, si depuis le nœud a_i le chemin vers b_i n'est pas un plus court chemin alors R n'est pas optimal. Il serait possible d'ajouter à R un mécanisme permettant à a_i d'apprendre l'identifiant de tous les nœuds de sa sous-branche et donc de déterminer si b_i appartient à celle-ci ou non. Le nœud a_i étant à distance t de b_i cet algorithme convergerait également en temps t et pourrait être exécuté en parallèle de R. Donc $|\overline{P}| = |P| - t$.

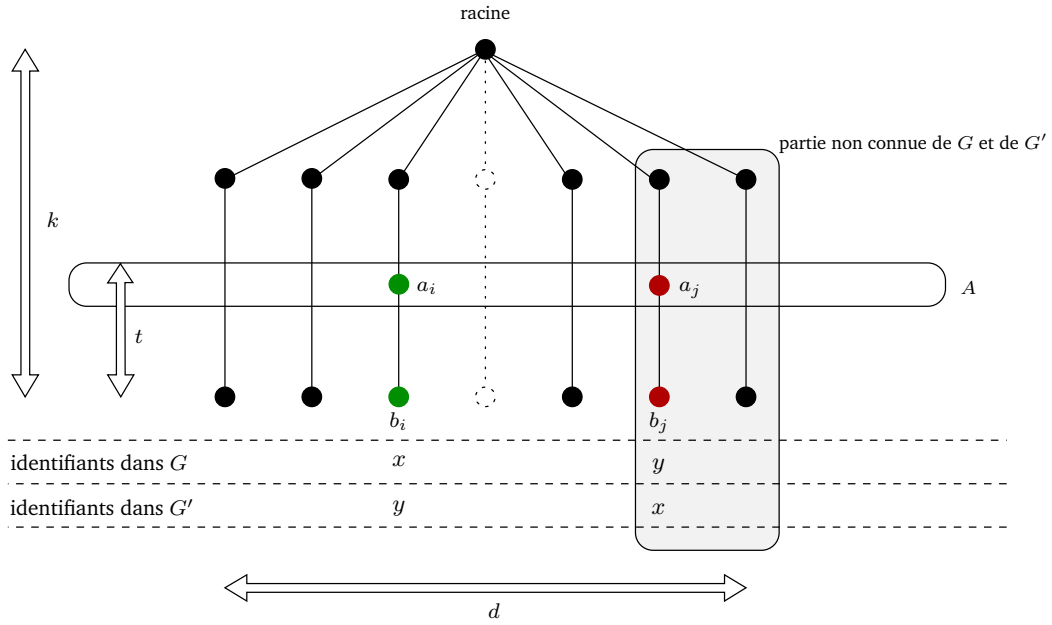

Figure 3.1: Graphe et notations utilisées pour la borne inférieure pour le temps de convergence d'un algorithme de routage compact.

Considérons ℓ la longueur minimale d'un chemin contenant tous les nœuds de A.

$$\ell = (2d-1)\cdot(k-t)$$

Par définition de P et d'après la proposition 3.1 $|\overline{P}| < \ell$ car :

$$|\overline{P}| = |P| - t = L(R) - t \ < \ (2d-1)\cdot(k-t) = \ell$$

Il s'ensuit que le chemin $\overline{P}$ contient au maximum $d-1$ nœuds de A. Donc :

Proposition 3.2. *Il existe un nœud $a_j \in A$ n'appartenant pas au chemin $\overline{P}$.*

Notons $G' \in \mathcal{F}$ le graphe identique à G et dans lequel seul les identifiants de b_j et b_i sont intervertis. Dans G les identifiants de b_i et b_j sont respectivement x et y alors qu'ils sont respectivement y et x dans G'. Notons également P' la route produite par R pour le même identifiant de destination que P, i.e. : x.

Proposition 3.3. *Les routes P et P' ont un préfixe commun de longueur au moins $|\overline{P}|$.*

En effet, tout nœud à distance $> t$ de b_i ou de b_j aura la même table de routage dans G et dans G' car ces nœuds sont trop loin pour être influencés par cette unique différence entre G et G'. Plus précisément, seuls les nœuds de la sous-branche de a_i et a_j ont des tables différentes dans ces deux instances. Autrement dit, les routes P et P' traversent des nœuds dont les tables sont similaires jusqu'à ce que le nœud a_i soit atteint. Or, par définition $\overline{P}$ se termine lorsque le nœud a_i est atteint. donc $\overline{P}$ est un préfixe commun de P et de P'.

Observons à présent la portion de route restante entre a_i et le nœud d'identifiant x dans le graphe G'. Le nœud d'identifiant x n'appartient pas à la même branche que a_i dans G', il est donc à distance $2k-t$ de b_j le nœud d'identifiant x. De plus par définition b_j n'appartient pas à $\overline{P}$ donc $|P'| \geqslant |\overline{P}| + 2k - t$. Donc d'après les

propositions 3.1 et 3.3 : $|P'| \geqslant |P|$ (rappelons que $|P| = L(R)$). Ce qui contredit le fait que $L(R)$ est la plus longue route produite par l'algorithme R. Il en résulte que $L(R) \geqslant (2d-1) \cdot (k-t) + t$.

∎

Deuxième partie du théorème 3.1

Preuve. Pour prouver ce point, considérons la famille des (d,k)-étoiles avec $d = \lfloor 2(n-1)/D \rfloor$ et $k = \lfloor D/2 \rfloor$. Ces graphes ont un diamètre de $2k \leqslant D$ et contiennent au plus $dk+1 \leqslant n$ nœuds. Si $t \geqslant k$, le temps de convergence est de $\Omega(D)$. Considérons alors le cas où $t < k$, l'étirement s du routage de la racine vers toutes les feuilles est alors comme le suggère la première partie du théorème 3.1 :

$$s \geqslant \frac{(2d-1) \cdot (k-t) + t}{k} = \Omega\left(\left(1 - \frac{t}{k}\right) d\right)$$

Qui est équivalent à :

$$t = \Omega\left(1 - \frac{s}{d}k\right) = \Omega\left(1 - \frac{s}{d}D\right) = \Omega(D) \tag{3.1}$$

tant que $1 - s/d = \Omega(1)$. Or, comme $s \leqslant \frac{1}{3}n/D$, nous avons :

$$s/d \leqslant \frac{\frac{1}{3}n/D}{\lfloor 2(n-1)/D \rfloor} < \frac{\frac{1}{3}n/D}{2(n-1)/D - 1}$$

de plus, $D \leqslant n-1$, donc :

$$s/d \leqslant \frac{1}{3}\frac{n}{n-1}$$

Pour $n \geqslant 3$, le ratio s/d est inférieur à $1/2$ et donc $1 - s/d \geqslant 1/2$ ce qui confirme la validité de l'équation 3.1 et conclue la preuve de ce théorème. ∎

Remarque. Pour un $t = 0$, lorsqu'aucun prétraitement n'est fait, le problème établi par le théorème 3.1 est équivalent au problème de recherche dans sur une chaîne dans lequel la distance à destination est connue. La borne présentée ici statue que l'étirement est dans ce cas d'au moins $2d-1$ ce qui est connu être optimal [BYCR93, KRT96].

3.3.2 Complexité de communication - Borne inférieure

Cette section donne la preuve qu'aucun algorithme ne peut garantir un coût de communication de $\mathcal{O}(n^2)$ bits tout en conservant des routages de plus court chemin pour des graphes peu denses. Commençons par observer la relation entre la quantité d'information envoyée par les nœuds et la taille maximale des tables de routage déductible de ces messages (Lemmes 3.1 et 3.2). Puis en utilisant une famille de graphes pour laquelle tout algorithme de plus court chemin doit avoir une table volumineuse nous pourrons donner une borne supérieure ($\Omega(n^2)$) sur le nombre de messages requis pour router en plus court chemin pour sur graphe (théorème 3.2).

Lemme 3.1. *Supposons que durant l'exécution d'un algorithme distribué synchrone, après un temps t un nœud u a reçu d'un voisin v un total de $L_{u,v}$ bits d'information séparés dans x messages. Le nœud u peu déduire de ces messages au plus $L_{u,v} + \log\left(\binom{L_{u,v}}{x}\binom{t}{x}\right) \leqslant 2L_{u,v} + t$ bits d'information.*

Preuve. Le volume d'informations utiles reçu par un nœud u est directement lié au nombre de configurations induites par les messages reçus. Toute configuration étant constituée d'une séquence de $L_{u,v}$ bits découpés en k, ainsi qu'un d'un temps $t_i \geqslant i$ associé à tout message M_i. Analysons donc le nombre de configurations produites par :

- la composition de $2^{L_{u,v}}$ séquences de bits L_u différentes ;
- chacune pouvant être découpée de $\binom{L_{u,v}}{x}$ façons distinctes en k messages non vides ;
- l'ensemble de ces messages pouvant être étiquetés de $\binom{t}{x}$ manières distinctes.

Au total le nombre de configurations est donc de $C = 2^{L_{u,v}}\binom{L_{u,v}}{x}\binom{t}{x}$. Or $\binom{L_{u,v}}{x} \leqslant 2^{L_{u,v}}$ et $\binom{t}{x} \leqslant 2^t$, la quantité d'informations encodées dans les messages reçus par le nœud u est donc d'au plus :

$$\log\left(2^{L_{u,v}} \cdot \binom{L_{u,v}}{x} \cdot \binom{t}{x}\right) \leqslant 2L_{u,v} + t$$

∎

Lemme 3.2. *Tout algorithme de routage R générant pour en ensemble de nœuds $U \subseteq V$ des tables de taille M bits en un temps t a un coût de communication de supérieur à :*

$$\frac{1}{2} \cdot \sum_{u \in U}(M - \mathcal{O}(\log n) - \text{degre}(u) \cdot t))$$

Preuve. Considérons un nœud $u \in U$ et observons la relation entre la quantité d'information envoyés par ses voisins et la taille en bits de sa table de routage $|R_u|$. D'après le lemme 3.1, un nœud u peut déduire un total de $2L_{u,v}+t$ bits d'informations par voisin v lui envoyant $L_{u,v}$ bits sur une période de temps t. Donc la quantité totale d'information disponible au nœud u après une période de temps t est de :

$$\sum_{v \in \Gamma(u)} (2L_{u,v} + t)$$

Or le nœud u doit avec cette quantité d'information construire sa table de routage qui, d'après l'hypothèse de départ, a une taille M. Or le nœud u a à l'initialisation ne connaît que son identifiant et a donc une mémoire de $\mathcal{O}(\log n)$, donc :

$$\begin{aligned}
|R_u| - \mathcal{O}(\log n) &\leqslant \sum_{v \in \Gamma(u)} (2L_{u,v} + t) \\
2\sum_{v \in \Gamma(u)} L_{u,v} &\geqslant |R_u| - \mathcal{O}(\log n) - \sum_{v \in \Gamma(u)} t \\
\sum_{v \in \Gamma(u)} L_{u,v} &\geqslant \frac{1}{2} \cdot (M - \mathcal{O}(\log n) - \text{degre}(u) \cdot t)
\end{aligned}$$

Autrement dit, tout nœud $u \in U$ doit recevoir de ces voisins au moins

$$\frac{1}{2} \cdot (M - \mathcal{O}(\log n) - \text{degre}(u) \cdot t)$$

bits pour calculer sa table.

En sommant sur tous les nœuds appartenant à l'ensemble U, le coût de communication de l'algorithme R peut être borné comme suit :

$$\mathsf{Msg_s}(\mathrm{R}) \geqslant \sum_{u \in U} \sum_{v \in \Gamma(u)} L_{u,v} \geqslant \frac{1}{2} \cdot \sum_{u \in U} (M - \mathcal{O}(\log n) - \mathrm{degre}(u) \cdot t)$$

∎

Théorème 3.2. *Il existe une constante $\lambda > 0$, et un graphe non pondéré de diamètre $\mathcal{O}(\log n)$ et de degré maximum 3 pour lequel tout algorithme de plus court chemin distribué dans le modèle synchrone de complexité en temps d'au plus λn a un coût de communication de $\Omega(n^2)$ bits. Cette borne est valable que l'algorithme considéré soit étiqueté ou non.*

La suite de cette section contient la preuve de ce théorème, elle repose notamment sur la famille de graphes proposée par le lemme suivant qui est valable aussi bien pour les algorithmes avec indépendance des noms qu'étiquetés :

Lemme 3.3 ([GP96b])**.** *Pour tout entier $d \in [3, n/2]$, il existe une famille de graphe de degré maximum d et de diamètre $\mathcal{O}(\log_d n)$ tel que tout algorithme de routage de plus court chemin utilisant des noms dans l'espace $[1, n]$ génère pour certains graphes de la famille des tables de routage de $\Omega(n \log d)$ bits pour $\Theta(n)$ nœuds.*

Considérons la famille de graphes $\mathcal{H}$ comme définie dans le lemme 3.3 pour $d = 3$ ainsi qu'un algorithme de routage de plus court chemin R ayant un coût en temps de t sur les graphes $\mathcal{H}$. D'après le lemme 3.3, pour tout algorithme R, il existe un graphe $G \in \mathcal{H}$ pour lequel R génère des tables de routage de $\Theta(n)$ bits pour $\Theta(n)$ nœuds de G.

Considérons l'ensemble des nœuds de G se voyant attribuer par R une table de routage d'au moins δn bits, avec δ le nombre de bits requit pour stocker une entrée de routage par R ($|B| = \Theta(n)$).

D'après le lemme 3.2 le coût de communication nécessaire pour construire l'ensemble de ces tables est d'au moins

$$\mathsf{Msg_s}(\mathrm{R}) \geqslant \frac{1}{2} \cdot \sum_{b \in B} (\delta n - \mathcal{O}(\log n) - \mathrm{degre}(b) \cdot t)$$

Soit

$$\mathsf{Msg_s}(\mathrm{R}) \geqslant \Omega(n \log d) \cdot (\delta n - \mathcal{O}(\log n) - d \cdot t)$$
$$\mathsf{Msg_s}(\mathrm{R}) = \Omega(n^2)$$

3.4 Un schéma de routage distribué asynchrone

Le schéma de routage, DCR(k) (pour *Distributed Compact Routing*), présuppose que chaque nœud ait initialement reçu une couleur* parmi $[1, k]$, ou k est un nombre entier paramètre du schéma. En pratique, chaque nœud tire une couleur de manière aléatoire uniforme dans $[1, k]$. Notre schéma, est en dehors de ce choix, déterministe. Comme nous le verrons dans le théorème 3.3, la correction du schéma DCR(k) est indépendante de la coloration des nœuds, ce qui n'est pas le cas du schéma [AGM+08] duquel est inspiré DCR(k).

Théorème 3.3. *Soit G un graphe pondéré connexe ayant n nœuds et de hop-diamètre D. Pour toute k-coloration de G,* DCR(k) *est un schéma de routage distribué asynchrone déterministe pour G. Il s'exécute en un temps $\mathcal{O}(D)$. Le coût de communication est inférieur à $\mathcal{O}(n)$ fois le nombre de message qu'un Bellman-Ford distribué consomme par source dans G.*

L'algorithme de routage qu'il produit à un étirement de 5, et des entêtes de $|\mathsf{Entrées}(\text{DCR}(k))| = \mathcal{O}(\min\{D, \log n\} \cdot \log n) = \mathcal{O}\big(\log^2 n\big)$ *bits. Chaque décision de routage est prise en temps constant et l'entête d'un message est modifiée au plus une fois lors du routage.*

Le schéma DCR(k) dépend directement d'un Bellman-Ford distribué asynchrone pouvant générer $\Omega(2^n)$ messages dans le pire scénario asynchrone pour des graphes ayant un grand ratio d'aspect (voir [ABNG94]). Dans certaines circonstances, notre schéma peut donc générer un nombre exponentiel de messages. Cependant, dans un scénario synchrone et pour un graphe de petit ratio d'aspect, la complexité en moyenne d'un Bellman-Ford distribué est polynomial et même $\mathcal{O}\big(n^2\Delta^3\big)$ avec grande probabilité, Δ étant le dégrée maximum du graphe [TS95].

Le résultat suivant (théorème 3.4) spécifie la taille des tables de routages et la complexité en nombre de messages. Ces deux résultats dépendent de la coloration des nœuds, du ratio d'aspect W du graphe et de l'hypothèse synchrone. Les paramètres entrant en jeu dans l'analyse , n, m, D, W, ne sont pas initialement connus des nœuds. Nous ferons les suppositions suivantes:

Coloration Aléatoire La coloration des nœuds est aléatoire uniforme dans $[1, k]$ et $k = n^\alpha$ avec $\alpha \in (0, 1)$. Les résultats obtenus avec cet hypothèse sont valides en espérance ou avec forte probabilité †.

Scénario synchrone. Le modèle $\mathcal{LOCAL}$ est considéré

Nous définissons maintenant $\xi = 1 + D(1 - 1/W)$. Cette valeur apparaît dans la complexité de notre schéma pour un scénario synchrone. Elle correspond au nombre maximum de changements d'état d'un nœud u lors du calcul de la hop-distance vers un nœud v. Pour chaque changement, u envoie un message a ses voisins. Observons que pour les graphes non pondérés $\xi = 1$ car $W = 1$.

Théorème 3.4. *Soit G un graphe pondéré connexe ayant n nœuds, m arêtes, de hop-diamètre D et de ratio d'aspect W. Sous l'hypothèse de coloration aléatoire,* DCR(k) *sur*

*Nous n'imposons pas que deux nœuds voisins aient une couleur différente.

†Cela signifie qu'ils sont valides avec probabilité au moins $1 - 1/n^c$ avec $c \geqslant 1$.

G induit avec forte probabilité des tables de routages et une mémoire de travail de taille $\mathcal{O}(k \log k + n/k)$. De plus, dans le cas synchrone, le coût de communication est de :

$$\mathsf{Msg_s}\left(\mathrm{DCR}(k)\right) = \begin{cases} \mathcal{O}\Big(\xi m \left(k \log k + \frac{n}{k}\right) + \frac{n^2}{k} \cdot \min\left\{D, k\right\}\Big) & \textit{en moyenne} \\ \mathcal{O}\Big(\xi m \left(k \log k + \frac{n}{k}\right) + \frac{n^2}{k} \cdot \min\left\{D, k \log \mathbf{k}\right\}\Big) & \textit{avec grande probabilité.} \end{cases}$$

Pour $k = \sqrt{n/\log n}$ les tables de routages ont donc $\mathcal{O}\big(\sqrt{n \log n}\,\big)$ entrées. Dans le cas non pondéré ($W = \xi = 1$), le coût de communication du théorème 3.4 devient que ce soit en espérance ou avec grande probabilité :

$$\mathsf{Msg_s}\left(\mathrm{DCR}(\sqrt{n/\log n})\right) = \tilde{\mathcal{O}}\Big(m\sqrt{n} + n^{3/2} \cdot \min\left\{D, \sqrt{n}\right\}\Big)$$

Notons également que dans le cas asynchrone, avec une légère modification de l'algorithme, en ajoutant un α-synchroniseur (décrits dans la section 2.5.1) aux différentes constructions d'arbres (partiels ou couvrants), nous pouvons conserver les mêmes garanties sur le coût de communication et la complexité en temps. Rappelons néanmoins que l'utilisation de ce type de synchroniseurs implique un coût fixe de $\mathcal{O}(mD)$ messages, ce qui peut dans certains cas détériorer la complexité. Plus précisément, l'utilisation d'un α-synchroniseur implique une augmentation de la complexité de communication lorsque $D = \Omega(k \log k)$ et $m = \Omega(\log k \cdot n^2/D)$.

Le corollaire suivant est un point important de l'analyse:

Corollaire 3.1. *Sous les hypothèses de coloration aléatoire et de communication synchrone. Pour les graphes non pondérés à n nœuds, $\tilde{\mathcal{O}}(n)$ arêtes et de hop-diamètre poly-logarithmique. Le schéma de routage distribué* $\mathrm{DCR}(\sqrt{n}\,)$ *a un coût de communication $\tilde{\mathcal{O}}\big(n^{3/2}\big)$, produit un algorithme de routage d'étirement 5 et induit avec grande probabilité que la mémoire de travail et les tables de routage de ont une taille de $\tilde{\mathcal{O}}(\sqrt{n}\,)$.*

L'algorithme de routage DCR utilise de nombreuses techniques et idées. Commençons ce chapitre par une présentation résumée de celles-ci. La section 3.4.1 décrit l'algorithme de routage et la section 3.4.2 les différentes phases du schéma de routage distribué. Les parties les plus originales de cette section sont :

- la construction de réseaux logiques (section 3.7) ;
- l'analyse de des bornes supérieures pour des construction d'arbres non-couvrants (section 3.5.5) ;
- une nouvelle technique pour réduire l'étirement dans certains cas de routage (section 3.8) ;

3.4.1 Principes de l'algorithme de routage

Considérons une k-coloration aléatoire des nœuds du graphe avec $c(u) \in [1, k]$ la couleur sélectionnée par le nœud u. Notons que, dans la plupart des cas, le meilleur choix pour k sera $\tilde{\Theta}(\sqrt{n}\,)$. En plus de la coloration, les nœuds sont divisé en k groupes de taille $\mathcal{O}(n/k)$ avec grande probabilité. Cette séparation en groupe sera faite par une fonction de hachage équilibrée, décrite dans [AGM+08], qui associe en temps constant à tout identifiant de nœud un éléments de l'ensemble $[1, k]$. Un nœud de couleur i sera alors responsable des informations de routages de tout nœud dont la

valeur haché est i. Les nœuds de couleurs 1, appelés *landmarks*, ont un rôle spécial dans le schéma.

Tables de routage. Soit u un nœud du graphe. Le nœud u conserve trois types d'informations de routage :

(1) Il conserve dans une table $\mathcal{D}_u$, appelé table de routage-direct, les informations de routage de plus court chemin vers les landmarks et vers sa *boule de voisinage* $\mathrm{B}(u)$. $\mathrm{B}(u)$ est un ensemble contenant les $\mathcal{O}(k \log k)$ nœuds plus proches de u. Plus précisément, cette boule contient le nombre minimal de nœuds les plus proches de u tel que chaque couleur soit représentée au moins une fois dans la boule. Pour chaque nœud $v \in \mathcal{D}_u$, le nœud u conserve un plus court chemin de u vers v.

(2) Pour chaque couleur i, le nœud u maintient dans une table $\mathcal{M}_u$, appelé table des représentants, un nœud de couleur i le plus proche de u.

(3) Pour chaque nœud v tel que $h(v) = c(u)$, le nœud u conserve dans une table $\mathcal{I}_u$, appelée table de routage indirecte, un landmark l proche de v et un plus court chemin de v vers l.

Plus exactement, les chemins contenus dans les tables directes et indirectes ne sont pas arbitraires. Ils sont extraits d'arbre de plus court chemin fixé T_v, enraciné en chaque nœud v. De plus, les chemins sont compressés dans des *étiquettes de routage* grâce a une variante distribuée de la technique [FG01b]. Ces étiquettes donnent une information partielle sur le chemin dans l'arbre et n'utilisent que $\mathcal{O}(\min\{D, \log n\} \cdot \log n)$ bits. Au total, chaque nœud u stocke au plus $\mathcal{O}(k \log k + n/k)$ entrées puisqu'il y a $\mathcal{O}(n/k)$ landmarks et nœuds ayant une valeur hachée égale à $c(u)$.

Algorithme de routage. Nous allons maintenant décrire le routage d'une source s a une destination t en utilisant ces tables. Si $t \in \mathcal{D}_s$, la table $\mathcal{D}_s$ permet de transférer le paquet le long d'un plus court chemin vers t. Dans le cas contraire, le nœud s fait suivre le paquet au représentant u de la couleur $h(t)$ stocké dans la table $\mathcal{M}_u$. Notons que u peut être son propre manager pour la couleur $h(t)$ ce qui implique que $u = s$. Une fois arrivé au nœud u, le landmark l et le chemin compressé de t à l sont récupérés dans la table de routage indirecte $\mathcal{I}_u$ et placés dans l'entête du paquet.

A présent, grâce aux informations ajoutées dans l'entête du paquet, tout nœud v du graphe est capable en utilisant sa table $\mathcal{D}_v$ et l'étiquette de routage de t de faire suivre le paquet jusqu'au nœud t via un plus court chemin dans l'arbre de plus court chemin T_l enraciné en l (voir figure 3.2). Cet algorithme de routage sera décrit de manière plus formelle dans la section 3.9.

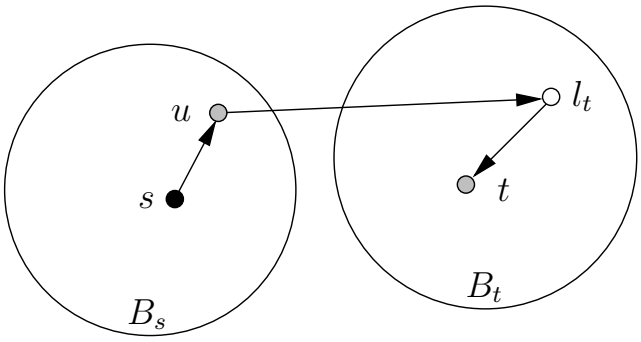

Figure 3.2: Routage s de t avec $c(u) = h(t)$.

Étirement. L'analyse de l'étirement de cet algorithme est le suivant. Si $t \in \mathcal{D}_s$, l'étirement est 1. Dans le cas contraire, supposons s et t à distance d l'un de l'autre. De plus supposons pour l'instant que l est le plus proche landmark de t. La longueur de cette route $s \rightsquigarrow u$ est alors au plus d car $t \notin \mathcal{D}_s$. Le chemin $l \rightsquigarrow t$ est de longueur au plus $2d$ car le landmark de s (qui appartient à $\mathrm{B}(s)$) est à distance au plus $2d$ de t et l_t est le plus proche landmark de t. Il en découle que la longueur du chemin $u \rightsquigarrow l$ est borné par la longueur du chemin $u \rightsquigarrow s \rightsquigarrow t \rightsquigarrow l$ qui est au plus $4d$. Ainsi, le coût de la route $u \rightsquigarrow t$ dans l'arbre de plus court chemin couvrant enraciné en l est d'au plus $4d + 2d = 6d$. En tout l'étirement de la route $s \rightsquigarrow u \rightsquigarrow t$ est donc $d + 6d = 7d$. L'algorithme a donc un étirement de 7.

Dans ce dernier cas, un étirement de 5 est atteignable si le segment de la route $u \rightsquigarrow t$ est parcouru dans l'arbre T_{l_s} plutôt que dans T_{l_t}, l_s et l_t étant respectivement les landmarks plus proche de s et t. En effet, la route $u \rightsquigarrow t$ n'est alors pas plus longue que la route $u \rightsquigarrow s \rightsquigarrow l_s \rightsquigarrow s \rightsquigarrow t$ ou chacun des quatre segments est un plus court chemin de longueur au plus d menant à un total de $5d$ depuis s. Autrement dit, u peut conserver un meilleur chemin dans l'arbre de landmark dans $\mathcal{I}_u$ pour v. Grâce a cela notre algorithme atteint un étirement de 5.

3.4.2 Principe du schéma de routage distribué

Le but du schéma de routage distribué est de calculer pour chaque nœud u, les tables $\mathcal{D}_u$, $\mathcal{M}_u$ et $\mathcal{I}_u$. Le calcul de la table de routage-indirecte $\mathcal{I}_u$ est effectué après que chaque nœud v ait calculé sa table de routage-directe $\mathcal{D}_v$ et sa table de représentants $\mathcal{M}_v$. Nous utilisons pour cela une synchronisation faible qui permets de réduire le nombre de messages dans le modèle asynchrone car aucune information non-fiable de la table de routage-directe n'est envoyée.

L'algorithme effectuant la construction de $\mathcal{D}_u$ et $\mathcal{M}_u$ est composé des deux phases suivantes, prenant chacune un temps $\mathcal{O}(D)$:

Phase 1 - Construction d'arbres de plus court chemin couvrants ou tronqués et élection d'un landmark (tables $\mathcal{D}_u$ et $\mathcal{M}_u$). Grâce à un algorithme proche d'un Bellman-Ford distribué, chaque nœud v construit un arbre de plus court chemin T_v. Cet arbre couvre le graphe entier si le nœud v est un landmark mais seulement une partie des nœuds s'il n'est pas landmark. Dans cette étape, le nœud u stocke ses parent dans l'arbre enraciné en v et prend connaissance du landmark de plus petit identifiant qui deviendra le leader noté $l_{\min}$. Le nœud u construit en même temps sa table de représentants $\mathcal{M}_u$. L'algorithme détecte ensuite la terminaison de cette phase.

Plus précisément, chaque nœud v détecte que tout les nœuds devant appartenir à T_v appartiennent effectivement à T_v. Simultanément les nœuds v cherchent à déterminer le landmark d'identifiant minimum dans T_v. Chaque landmark et en particulier $l_{\min}$ déterminent donc que $l_{\min}$ est le landmark d'identifiant minimum. Lorsque $l_{\min}$ apprend qu'il est le leader, il diffuse son nom et récupère les accusés de terminaison de tout les autres nœuds. Cette phase est décrite en détail dans la section 3.5. Dans un scénario synchrone, la phase 1 consomme $\mathcal{O}(\xi m(k \log k + n/k))$ messages.

Lorsque le nœud $l_{\min}$ a détecté la terminaison de cette phase, il initie la phase 2 en prévenant tous les nœuds qu'ils doivent commencer la phase 2.

Phase 2 - Calcul des étiquettes de routage (table $\mathcal{D}_u$). Durant cette phase, tout nœud v va initier le calcul des étiquettes de routage pour son propre arbre T_v. Tout nœud u appartenant à T_v va finir par détecter qu'il a construit son étiquette dans tout arbre T_w enraciné en w. Il détecte cela assez simplement car pour tout nœud w tel que $u \in T_w$, le nœud u a une entrée $w \in \mathcal{D}_u$ (calculée pendant la phase 1). Ces détections de terminaison locales sont ensuite collectés par $l_{\min}$ dans son propre arbre, de la même manière que durant la phase 1. Les étiquettes de routage sont décrites en section 3.6.1, et cette phase est décrite dans la section 3.6. La phase 2 à un coût de communication de $\mathcal{O}(m(k \log k + n/k))$ messages.

Le rôle des deux dernières phases est de construire $\mathcal{I}_u$. Pour cela, u a besoin d'apprendre l'étiquette de routage de chaque nœud v tel que $h(v) = c(u)$ dans l'arbre enraciné au landmark le plus proche de v, l_v. Si u est relativement proche de v, il collectera en fait plusieurs étiquettes issue des arbres enracinés en plusieurs landmarks proche de v. Il ne conservera alors que le label correspondant au chemin le plus court vers v. Ceci est fait en deux étapes menées en parallèle. Dans la phase 3.1, chaque nœud v informe tout représentant u tel que $h(v) = c(u)$. Dans la phase 3.2, chaque nœud v envoi ses étiquettes dans les arbres des landmarks proches à ses représentants respectif, cette deuxième phase permet d'améliorer l'étirement dans certains cas.

Phase 3.1 - Construction d'arbres logiques et diffusions dans ces arbres (table $\mathcal{I}_u$). L'idée générale de cette phase est la suivante. Chaque nœud v de valeur hachée $h(v)$ envoie ses étiquettes au plus proche nœud de couleur $h(v)$, noté w. Le nœud w est alors chargé de diffuser cette information à tout nœud u de couleur $h(v)$. Notons que notre borne sur la mémoire de travail ne nous permet pas de diffuser directement dans un arbre en $\mathcal{O}(n)$ messages, un nœud ne pouvant pas stocker tout ses enfants. Nous utilisons donc une technique plus complexe. Nous construisons un schéma de diffusion efficace composé de k *arbres logiques*, un de chaque couleur (item 1). Nous les utiliserons pour diffuser les étiquettes (item 2).

1. Pour chaque couleur $i \in [1, k]$, nous construisons un arbre logique $\mathcal{T}_i$ dont les nœuds sont les nœuds de couleur i dans G. Une arête entre w et w' dans $\mathcal{T}_i$ représente un chemin de w a w' dans l'arbre $T_{l_{\min}}$ sans aucun nœud intermédiaire de couleur i. Comme il existe $\Theta(n/k)$ nœud de couleur i, chaque nœud w peut désormais stocker ses voisins dans $\mathcal{T}_i$. Des voisins dans $\mathcal{T}_i$ ne sont, à priori, pas voisins dans G. Un nœud w stocke alors l'étiquette de routage de ses voisins w' dans $T_{l_{\min}}$ afin de pouvoir communiquer avec eux.

   Nous pouvons prouver que l'espérance des longueur des arcs de $\mathcal{T}_i$ est d'au plus $2 \min\{D, k\}$ arêtes de G. Cela prend alors $\mathcal{O}(n_i \min\{D, k\})$ messages pour construire $\mathcal{T}_i$, où n_i est le nombre de nœud de couleur i. En sommant pour

les k arbres logiques, cela fait donc $\mathcal{O}(\sum_i n_i \min\{D,k\}) = \mathcal{O}(n \min\{D,k\})$ messages.

2. Un nœud v envoie l'identifiant de son plus proche landmark l_v et son étiquette dans T_{l_v} à w, nœud le plus proche de couleur $i = h(v)$. Le nœud w diffuse cette étiquette à ces voisins dans le réseau logique $\mathcal{T}_i$. Finalement, tout nœud u de couleur i recevra ces étiquettes et construira sa table $\mathcal{I}_u$. Le coût de diffusion de ces étiquettes est de $\mathcal{O}(\min\{D, k\log k\} + n/k \cdot \min\{D,k\})$ message. En effet, il y a $\mathcal{O}(n/k)$ nœuds dans $\mathcal{T}_i$ connectés par des chemins d'au plus $\min\{D,k\}$ arêtes.

Ainsi, pour construire toutes les tables $\mathcal{I}_u$, et completer la phase 3.1, nous avons besoin de $\mathcal{O}(n \cdot (\min\{D,k\} + n/k \cdot \min\{D,k\})) = \mathcal{O}\big(n^2/k \cdot \min\{D,k\}\big)$ messages. A la fin de l'étape 3.1, l'algorithme de routage permet de router de toute source à toute destination avec un étirement d'au plus 7.

Phase 3.2 - Amélioration de l'étirement par échanges locaux d'étiquettes de routage (table $\mathcal{I}_u$) Dans cette phase, les nœuds échangent des étiquettes de routage localement afin de garantir que l'algorithme de routage produit des routes d'étirement au plus 5. Nous nous intéressons au cas ou l'étirement de s à t peut être 7, c'est a dire quand $s \in \mathcal{D}_t$ et $t \notin \mathcal{D}_s$. Pour obtenir un étirement de 5 dans ce cas, le représentant w du nœud s peut apprendre l'étiquette de routage induisant le meilleur chemin de w à s parmi les arbres T_{l_s} et T_{l_t}, avec l_s et l_t les landmarks respectifs de s et t.

Pendant cette phase chaque nœud t demande à tout nœud s de lui envoyer l'identifiant de son plus proche landmark, l_s. Le nœud t renverra suite à cela on étiquette de routage dans l'arbre T_{l_s} au nœud w (via s le nœud s). L'étape 3.2 consomme $\mathcal{O}(n \cdot k\log k \cdot \min\{D, k\log k\})$ messages.

3.5 Phase 1: Construction d'arbres de plus court chemin couvrants ou tronqués et élection d'un landmark (table $\mathcal{D}_u$)

Cette phase a pour objectif de construire les tables de routage-direct. Pour parvenir à cela, tout nœud $v \in V$ va initier un algorithme proche d'un Bellman-Ford distribué (algorithme 1). L'arbre enraciné en v, couvre progressivement les nœuds ayant besoin d'ajouter v dans leurs table de routage-direct.

- L'algorithme 2 spécifie les conditions sous lesquelles un nœud u ajoute dans sa table $\mathcal{D}_u$ un nœud v et donc les conditions sous lesquelles un nœud u s'ajoute dans l'arbre enraciné en v. Cet algorithme spécifie également les circonstances entraînant la modification d'entrées dans $\mathcal{D}_u$.

- La construction de ces arbres est synchronisée en utilisant un convergecast. Tout nœud u rejetant une amélioration qui lui est proposée par un nœud w, renvoie un message d'acquittement au nœud w. À l'inverse, si le nœud u accepte la mise à jour de route proposée par w il n'enverra un message d'acquittement que lorsqu'il aura proposé cette nouvelle route à tous ses voisins, et que ceux-ci auront acquitté cette dernière.

- L'algorithme permet également d'élire le landmark d'identifiant minimum $l_{\min}$.

3.5.1 Structures et notations pour la phase 1

La table de routage-direct, $\mathcal{D}_u$, a des nœuds comme clefs. Les clefs de cette table sont destinées à être les nœuds vers lesquels u connaît un plus court chemin : Les nœuds proches de u, appelés boule de voisinage de u et notés $\mathrm{B}(u)$ ainsi que les landmarks L. Pour toute clef $v \in \mathcal{D}_u$, notée $\mathcal{D}_u[v]$, les valeurs suivantes sont stockées :

- $\mathrm{nexthop}$ est un nœud, destiné à être un voisin du nœud u sur un plus court chemin de u à v ;
- d est une distance, destinée à être la distance entre u et v ;
- hd est une hop-distance, destinée à être la hop-distance entre u et v ;
- c est une couleur, destinée à être la couleur du nœud v ;
- valid est un compteur, destiné à compter le nombre d'acquittements reçu durant différentes phases de l'algorithme ;
- size est un entier, destiné à être le nombre de nœuds du sous-arbre enraciné en u pour l'arbre de v ;
- $\mathrm{largestChild}$ est un nœud, destiné à être l'enfant du nœud u dans l'arbre de v ayant le poids maximal parmi ses enfants ;
- $\mathrm{largestSize}$ est un entier, destiné à être le poids du sous-arbre enraciné en $\mathrm{largestChild}$ dans l'arbre de v ;
- label est une étiquette de routage, destinée à être l'étiquette de routage de u dans l'arbre enraciné en v.

La table de représentant, $\mathcal{M}_u$, a des couleurs comme clefs. Pour toute clef $i \in \mathcal{M}_u$, les valeurs suivantes sont stockées :

- node est un nœud, destiné à être le nœud de couleur i le plus proche de u ;
- d est une distance, destinée à être la distance entre u et le nœud stocké dans $\mathcal{M}_u[i].\mathrm{node}$;
- hd est une hop-distance, destinée à être la hop-distance entre u et le nœud stocké dans $\mathcal{M}_u[i].\mathrm{node}$.

Comme le graphe considéré est pondéré, il est nécessaire de comparer des métriques incluant à la fois la distance classique et la hop-distance, l'ordre suivant est introduit pour comparer ces métriques :

Définition 3.1. *L'opérateur binaire $\prec$ est l'ordre lexicographique sur les paires de nombre réels. Autrement dit, pour tout couples de réels (x, y) et (x', y') :*

$$(x, y) \prec (x', y') \equiv (x < x') \vee (x = x' \wedge y < y')$$

La boule de voisinage d'un nœud est une sous-partie logique de sa table de routage-direct calculée en utilisant la table des représentants :

Définition 3.2. *Étant donné un nœud u, nous définissons la boule de voisinage de $u \in V$, $\mathrm{B}(u)$, comme étant l'ensemble des nœuds plus proches que le représentant le plus éloigné de u. La distance au représentant le plus éloigné correspond au rayon, $r(u)$, de $\mathrm{B}(u)$, il est égal à*

$$r(u) = \max_{j \in \{1, \cdots, k\}} (\mathcal{M}_u[j].d, \mathcal{M}_u[j].hd)$$

La boule de voisinage se définie donc comme :

$$\mathrm{B}(u) = \left\{ v \in \mathcal{D}_u \mid (\mathcal{D}_u[v].d, \mathcal{D}_u[v].hd) \prec r(u) \right\}$$

Cette définition implique bien évidement que tout représentant $v \in \mathcal{M}_u$ appartient à $\mathrm{B}(u)$.

3.5.2 Algorithmes de la phase 1

Lorsqu'un nœud $u \in V$ se réveille, il initialise ses tables $\mathcal{D}_u$ et $\mathcal{M}_u$. Il ajoute une entrée de clef u dans ces deux tables avec une distance à $(0, 0)$ avec un nexthop à null, de plus si u est un landmark il se considère pour commencer comme étant le landmark $l_{\min}$ et met donc $\mathrm{lmin}_u \leftarrow u$, sinon il met cette variable à null. Enfin le nœud u envoie un message à tous ses voisins : $\mathrm{SEND}_{\Gamma(u)}(\mathrm{mBellmanFord}(u, 0, 0, c(u)))$.

Lorsqu'un message $\mathrm{mBellmanFord}(v, d, hd, i)$ est reçu par un nœud u d'un voisin w, cela signifie intuitivement qu'il existe un nœud v de couleur i accessible depuis w par un chemin de longueur d composé de hd arêtes.

Algorithm 1: Réception par un nœud u d'un message $\mathrm{mBellmanFord}(v, d, hd, i)$ provenant d'un nœud w

```
   /* Backup of the current values                                          */
 1 oldnh ← null
 2 if v ∈ D_u then
 3 |   oldnh ← D_u[v].nexthop
 4 |   oldd ← D_u[v].d
 5 |   oldhd ← D_u[v].hd
 6 end
   /* Update D_u[v] if the new values are better than the current ones     */
 7 didUpdate ← updateIfBetter(v, d + ω(u, v), hd + 1, i)
 8 if didUpdate then                                   /* If interested:  */
   |   /* Send an ACK to the previous nexthop if it changed and if the ACK
   |      has not been already sent to it                                  */
 9 |   if (oldnh ≠ null) ∧ (oldnh ≠ D_u[v].nexthop) ∧ (D_u[v].valid < degree(u)) then
10 |   |   SEND_oldnh(mAckBellmanFord(v, oldd − ω(u, oldnh), oldhd − 1, lmin_u))
11 |   end
   |   /* Propagate the information about v                               */
12 |   D_u[v].valid ← 0
13 |   SEND_Γ(u)(mBellmanFord(v, D_u[v].d, D_u[v].hd, i))
14 else                                  /* If not interested:  send an ACK */
15 |   SEND_w(mAckBellmanFord(v, d, hd, lmin_u))
16 end
```

La procédure $\mathrm{updateIfBetter}(v, d, hd, i)$,exécutée par un nœud u, met à jour les tables $\mathcal{D}_u$ et $\mathcal{M}_u$ et supprime possiblement de $\mathcal{D}_u$ certains nœuds obsolètes. Cette procédure met à jour l'entrée $\mathcal{D}_u[v]$ si les paramètres proposés sont meilleurs que ceux stockés dans $\mathcal{D}_u[v]$. De plus cette procédure a une valeur de retour de True si et

seulement si une mise à jour de $\mathcal{D}_u[v]$ a effectivement eut lieu.

Algorithm 2: Procedure $\mathrm{updateIfBetter}(v, d, hd, i)$ exécutée au nœud u

```
1  if ((v ∉ D_u) ∨ ((d, hd) ≺ (D_u[v].d, D_u[v].hd))) ∧
     ((i = 1) ∨ (i ∉ M_u) ∨ ((d, hd) ≺ max_{j∈{1,···,k}}(M_u[j].d, M_u[j].hd))) then
2      D_u ← D_u ∪ {v}
3      D_u[v].nexthop ← v
4      D_u[v].d ← d
5      D_u[v].hd ← hd
6      D_u[v].c ← i
7      Update M_u[i] if better
8      foreach w ∈ B(u) ∪ {w' ∈ D_u | D_u[w'].c = 1} do
9          if D_u[w].valid < degree(u) then
10             w' ← D_u[w].nexthop
11             SEND_{w'}(mAckBellmanFord(v, D_u[w].d − ω(u, w'), D_u[w].hd − 1, lmin_u))
12         end
13         D_u ← D_u \ {w}
14     end
15     return True
16 else
17     return False
18 end
```

Par la suite nous utiliserons l'arbre orienté induit par les entrées des tables de routage-direct pour une destination v donnée :

Définition 3.3. *Définissons* $\mathrm{InvTree}_v$ *comme la composante connexe contenant le nœud* $v \in V$ *dans le graphe ayant comme ensemble de nœud* V *et comme ensemble d'arêtes l'ensemble des paires* $\{u, u'\}$ *tels que* $\mathcal{D}_u[v].\mathrm{nexthop} = u'$ *ou* $\mathcal{D}_{u'}[v].\mathrm{nexthop} = u$.

Ces arbres convergeront vers des arbres de plus court chemin.

Les messages de type $\mathrm{mAckBellmanFord}(v, d, hd, l)$ remontent des feuilles de l'arbre $\mathrm{InvTree}_v$ vers v, ils transportent l'identifiant l du landmark de plus petit identifiant du sous-arbre du nœud émettant le message. Lorsque v reçoit le dernier acquittement il connaît l'identifiant de $l_{\min}$. La distance dans le message permet au récepteur de confirmer que l'acquittement reçu correspond bien à la dernière distance annoncée.

Algorithm 3: Réception par un nœud u d'un message d'acquittement $\mathrm{mAckBellmanFord}(v, d, hd, l)$ provenant d'un nœud w

if $(v \in \mathcal{D}) \wedge (d = \mathcal{D}_u[v].d) \wedge (hd = \mathcal{D}_u[v].hd)$ **then** `/* Si l'acquittement correspond à la dernière distance annoncée */`
  $\mathrm{lmin}_u \leftarrow \min\{\mathrm{lmin}_u, l\}$
  $\mathcal{D}_u[v].\mathrm{valid} \leftarrow \mathcal{D}_u[v].\mathrm{valid} + 1$
  if $\mathcal{D}_u[v].\mathrm{valid} = \mathrm{degree}(u)$ **then**
    if $u \neq v$ **then**
      $v' \leftarrow \mathcal{D}_u[v].\mathrm{nexthop}$
      $\mathrm{SEND}_{v'}(\mathrm{mAckBellmanFord}(v, \mathcal{D}_u[v].d - \omega(u, v'), \mathcal{D}_u[v].hd - 1, \mathrm{lmin}_u))$
    else if $u = lmin_u$ **then**
      $\mathrm{SEND}_{\Gamma(u)}(\mathrm{mLeader}(u))$
    else
      if $\mathrm{globalValid}_u = \mathrm{degree}(u)$ **then**
        $v' \leftarrow \mathcal{D}_u[\mathrm{lmin}_u].\mathrm{nexthop}$
        $\mathrm{SEND}_{v'}(\mathrm{mAckLeader}())$
      end
    end
  end
end

Les deux algorithmes suivants permettent de prévenir tous les nœuds du graphe que tous les arbres ont convergé. Par la même occasion, les nœuds recevront l'identifiant du landmark $l_{\min}$.

Algorithm 4: Réception par un nœud u d'un message $\mathrm{mLeader}(l)$ provenant d'un nœud w

if $w = \mathcal{D}_u[l].\mathrm{nexthop}$ **then**
  $\mathrm{lmin}_u \leftarrow l$
  $\mathrm{SEND}_{\Gamma(u)}(\mathrm{mLeader}(l))$
else
  $\mathrm{SEND}_w(\mathrm{mAckLeader}())$
end

Algorithm 5: Réception par un nœud u d'un message d'acquittement $\mathrm{mAckLeader}()$ provenant d'un nœud w

$\mathrm{globalValid}_u \leftarrow \mathrm{globalValid}_u + 1$
if $\mathrm{globalValid}_u = \mathrm{degree}(u)$ **then**
  if $u = \mathrm{lmin}_u$ **then**
    $\mathrm{globalValid}_u \leftarrow 0$
    $\mathrm{SEND}_{\Gamma(u)}(\mathrm{mStartLabel}())$
    $\mathcal{D}_u[u].\mathrm{size} \leftarrow 1$
    $\mathcal{D}_u[u].\mathrm{largestChild} \leftarrow \mathrm{null}$
    $\mathcal{D}_u[u].\mathrm{largestSize} \leftarrow 0$
    $\mathcal{D}_u[u].\mathrm{valid} \leftarrow 0$
    $\mathrm{SEND}_{\Gamma(u)}(\mathrm{mSizes}(u))$
  else if $\mathcal{D}_u[u].\mathrm{valid} = \mathrm{degree}(u)$ **then**
    $v' \leftarrow \mathcal{D}_u[\mathrm{lmin}_u].\mathrm{nexthop}$
    $\mathrm{SEND}_{v'}(\mathrm{mAckLeader}())$
  end
end

3.5.3 Analyse de la phase 1, indépendamment de la coloration

Dans les lemmes qui suivent, nous utiliserons la notion d'accepter une annonce, définie comme suit :

Définition 3.4. *On dit qu'un nœud u* accepte *un message* $\mathrm{mBellmanFord}(v, d, hd, i)$*, ou accepte un triplet (v, d, hd), si u modifie sa table de routage-direct $\mathcal{D}_u$ à jour suite à la réception de ce message, ou à la considération de ce triplet.*

Les lemmes suivants sont valables quelque soit la coloration des nœuds. Observons pour commencer la manière dont les messages/triplets sont traités à leurs réception par un nœud u :

Lemme 3.4. *Un nœud n'accepte jamais un triplet (v, d, hd) si le triplet (v, d', hd') précédemment accepté pour le nœud v satisfait $(d', hd') \prec (d, hd)$.*

Preuve. D'une part, un nœud u n'accepte jamais un triplet (v, d, hd) si v appartient déjà a $\mathcal{D}_u$ et qu'il stocke une distance $(\mathcal{D}_u[v].d, \mathcal{D}_u[v].hd)$ strictement meilleure celle proposée $((d, hd))$. D'autre part, si v n'appartient pas à $\mathcal{D}_u$, cela signifie alors que v a été supprimé de $\mathcal{D}_u$ depuis l'acceptation d'un triplet (v, d', hd'). La seule raison possible à cela est que le nœud v, avec son ancienne distance (d, hd), était en dehors de $\mathrm{B}(u)$ a un instant passé quelconque. Cependant, le rayon de $\mathrm{B}(u)$, $r(u)$ n'augmente jamais. Cela indique que u doit accepter le triplet (v, d, hd) car la distance (d, hd) est plus grande que (d', hd'), et donc plus grande que le rayon. ∎

Lemme 3.5. *Il existe un temps t_{BF} à partir duquel aucun message* $\mathrm{mBellmanFord}$ *ou* $\mathrm{mAckBellmanFord}$ *n'est envoyé ou reçu dans l'ensemble du réseau.*

Preuve. Remarquons premièrement que l'émission d'un message $\mathrm{mBellmanFord}(v, d, hd, i)$ est soit due à l'initialisation d'un nœud v (auquel cas $d = hd = 0$) ou en conséquence de la réception d'un autre message $\mathrm{mBellmanFord}(v, d', hd', i)$ (voir algorithme 1). Donc, tout message $\mathrm{mBellmanFord}(v, d, hd, i)$ envoyé par un nœud w peut être associé:

- un de ces voisins u;
- la chaîne de messages initiée par v ayant amené ce message;
- et, par extension, le chemin de v à w utilisé par ces messages.

Clairement, par définition de l'algorithme, ce chemin a une longueur d et une hop-longueur de hd. D'après le lemme 3.4, un nœud u n'accepte donc jamais de message associé à un chemin passant par u. Ce qui implique qu'un nœud u n'accepte que des messages dont le chemin associé est simple. Pour résumer, une chaîne de messages $\mathrm{mBellmanFord}$ sont de hop-longueur au plus n, ce qui signifie que le nombre total de messages $\mathrm{mBellmanFord}$ envoyé durant la totalité de l'exécution est fini (borné par n!). D'ailleurs, comme il y a au plus un message $\mathrm{mAckBellmanFord}$ envoyé pour chaque message $\mathrm{mBellmanFord}$ précédemment envoyé, il y a également un nombre fini de messages $\mathrm{mAckBellmanFord}$, ce qui conclue la preuve. ∎

Dans un premier temps, l'objectif de tout nœud u est de rendre sa boule *complète* (pour cela il acceptera tout nouveau nœud dont il entend parler) :

Définition 3.5 (Boule de voisinage complete)**.** *La boule de voisinage $\mathrm{B}(u)$ d'un nœud $u \in V$ est dite* complète *si elle contient au moins un nœud de chaque couleur, y compris la couleur* 1.

Puis lorsque sa boule est complète, le nœud u ajoutera à sa boules tous les nœuds à distance inférieure à son rayon et ainsi obtenir une boule *ronde* :

Définition 3.6 (Boule de voisinage ronde)**.** *La boule de voisinage* $\mathrm{B}(u)$ *d'un nœud* $u \in V$ *est dite* ronde *si, pour tout nœud* $v \in \mathrm{B}(u)$*, tout autre nœud* $w \in V$ *tel que* $(d(u,w), hd(u,w)) \prec (d(u,v), hd(u,v))$ *appartient également à* $\mathrm{B}(u)$.

Pour garantir que l'étirement des routes obtenues est celui espéré, il est nécessaire que les tables de routage reflètent la topologie du graphe, et notamment les distance, correctement :

Définition 3.7 (Table de routage-direct correcte)**.** *Une entrée pour un nœud* v *dans* $\mathcal{D}_u$ *est dite* correcte *si :*

- *la distance* $\mathcal{D}_u[v].d$ *est égale à la distance dans le graphe* $d(u,v)$.
- *la hop-distance* $\mathcal{D}_u[v].hd$ *est égale à la hop-distance dans le graphe* $hd(u,v)$
- *et le nœud next-hop* $nh = B_u[v].\mathrm{nexthop}$ *est un voisin de* u *sur un plus court chemin vers* v.

La table de routage-directe d'un nœud est dite correcte si toutes les entrées *de cette table sont correctes.*

Enfin, pour garantir que le routage vers un nœud de la boule de u est possible sans faire de routage à la source, il faut également que les boules avoisinantes de B_u soient dans un état cohérent :

Définition 3.8 (Tables de routage-direct monotones)**.** *L'ensemble des tables de routage-direct est dit* monotone*, si, pour tout nœud* $u \in V$ *et pour toute destination* $v \in \mathcal{D}_u$*, le next-hop vers la destination possède également le nœud* v *dans sa propre table de routage-direct.*

Pour la même raison, il est important que les tables de représentants aient une certaine cohérence :

Définition 3.9 (Tables de représentant monotones)**.** *Les tables de représentant sont dites monotones si pour tout nœud* $u \in V$ *et pour toute couleur* $i \in [1,k]$*, on a* $\mathcal{M}_u[i].\mathrm{node} = w$ *avec* $u' = \mathcal{D}_u[w].\mathrm{nexthop}$ *et* $\mathcal{M}_{u'}[i].\mathrm{node} = w$.

Avant toute chose attardons nous sur un exemple de construction de boule de voisinage pour écarter toute incompréhension sur ce calcul.

Exemple de construction de boule de voisinage Voici un exemple de construction de boule de voisinage dans le modèle de communication $\mathcal{LOCAL}$. Cet exemple montre toutes les configurations qui peuvent être observés durant la construction des boules de voisinage (voir figure 3.3) :

1. Dans les deux premières rondes, la boule $\mathrm{B}(u)$ n'est pas complète, le nœud u ajoute tous les nœuds qu'il apprend ;

2. À partir de la troisième ronde, $\mathrm{B}(u)$ est complète. Durant cette ronde, le nœud u trouve également un nœud `blanc` plus proche que le nœud v (à distance $6 < 11$). À cet instant, le nœud u peut alors tenter de réduire sa boule, le représentant le plus éloigné étant ce nouveau nœud `blanc`. Le nœud u supprime donc de $\mathrm{B}(u)$ (et, *a fortiori* de $\mathcal{D}_u$) tous les nœuds à distance supérieure à 6, soit uniquement le nœud v.

3. Enfin, un nouveau chemin vers le nœud v est appris par u. Ce nouveau chemin propose une route de longueur inférieure à 6, le nœud v sera donc à nouveau ajouté à $\mathcal{D}_u$. À cet instant, la distances (*distance*,*hop-distance*) aux représentants des différentes couleurs sont :

   - couleur "`hachure`" : $(1, 1)$
   - couleur "`noir`" : $(1, 1)$
   - couleur "`gris`" : $(2, 2)$
   - couleur "`blanc`" : $(5, 5)$

   Le rayon de la boule est donc de 5, tout nœud à distance supérieure à 5 et appartenant à $\mathrm{B}(u)$ sera supprimé de $\mathrm{B}(u)$.

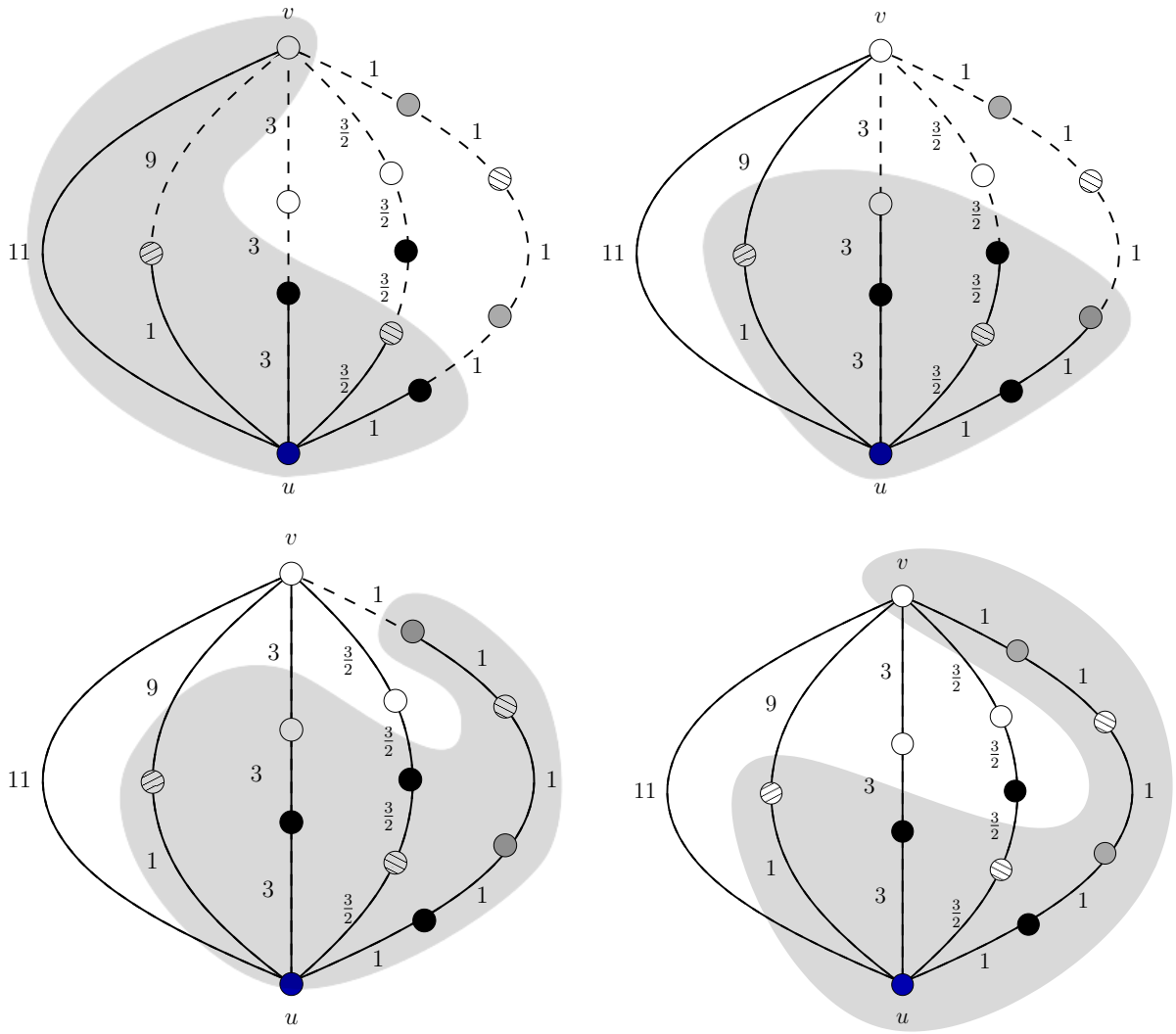

Figure 3.3: Exemple de construction synchrone de boule de voisinage durant laquelle le nœud v est ajouté 2 fois à la boule $\mathrm{B}(u)$ du nœud u (une première fois à distance 11 et une seconde à distance 5). Les nœuds appartenant à la boule de u à un instant donné sont ceux appartenant à la zone grise.

Commençons à présent à analyser la correction de cet algorithme.

Lemme 3.6. *Au temps t_{BF}, les tables de routage-direct sont correctes et monotones, de plus, toute les boules de voisinage sont rondes et complètes. Enfin, les tables de représentant sont également monotones.*

Preuve. Cette preuve est basée sur les propositions suivantes :

> **Proposition 3.4.** *Au temps t_{BF}, l'ensemble des tables de routage-direct sont monotones, il en est de même pour les tables de représentant.*
>
> **Preuve.** Observons les nœuds $u \in V$ et $w \in \Gamma(u)$ tels que il existe un nœud $v \in \mathcal{D}_u$ et $\mathcal{D}_u[v].\mathrm{nexthop} = w$ et $v \notin \mathcal{D}_w$. Alors, comme $\mathcal{D}_u[v].\mathrm{nexthop} = w$, il existe un temps t auquel w a envoyé un message $\mathrm{mBellmanFord}(v, -, -, -)$

et a donc v dans sa table $\mathcal{D}_w$. De plus, il existe un temps $t' > t$ tel que w a supprimé v de $\mathcal{D}_w$. Si w a supprimé v, alors, d'après l'algorithme 2 lignes 8 à 13, $\mathcal{D}_w$ contient pour toute couleur i au moins un nœud v_i de couleur i tel que v_i est strictement plus proche que v de w. Le nœud v_i est également plus proche que v de u. Comme tout nœud v_i a été proposé par u via un message $\mathrm{mBellmanFord}(v_i, -, -, i)$, le nœud u devrait également avoir supprimé le nœud v de $\mathcal{D}_u$ étant donné qu'il connaît au moins un nœud de chaque couleur strictement plus proche que ce dernier. Il y a donc une contradiction et l'ensemble des tables de routage-direct est donc monotone. De cette conclusion, et comme la table représentant est est construite en utilisant uniquement les tables de routage-direct, nous pouvons conclure que la table représentant est également monotone.

□

Proposition 3.5. *Au temps t_{BF}, tout nœud $u \in V$ a sa table de routage-direct, $\mathcal{D}_u$, complète.*

Preuve. Considérons qu'il existe un ensemble de nœuds $U_i \subset V$ pour lequel tout nœud $u \in U_i$ a $\mathcal{D}_u$ incomplète et manquant un nœud de couleur i. Notons $u_{i,\min}$ le nœud de l'ensemble U_i étant le plus proche d'un nœud de couleur i. Par définition, le nœud $u_{i,\min}$ a parmi ses voisins un nœud $w \notin U_i$ plus proche que lui d'un nœud de couleur i. Ce nœud w a donc envoyé un message $\mathrm{mBellmanFord}(-, -, -, i)$ à u. En effet, $\mathrm{updateIfBetter}(-, -, -, i)$ est nécessairement vérifié la première fois que w reçoit une entrée pour la couleur i, il envoie donc à cet instant un message $\mathrm{mBellmanFord}(-, -, -, i)$ à tous ses voisins. (algorithme 1 lignes 8 et 13). Pour la même raison, $u_{i,\min}$ devrait avoir accepté le message $\mathrm{mBellmanFord}(-, -, -, i)$ envoyé par w, et donc, devrait avoir une entrée de couleur i dans sa table de routage-direct. Il y a donc une contradiction est donc $U_i = \varnothing$ pour toute couleur i. □

Proposition 3.6. *Au temps t_{BF}, tout nœud $u \in V$ a sa table de routage-direct $\mathcal{D}_u$, correcte.*

Preuve. Considérons le plus proche couple de nœuds $u \in V$ et $v \in \mathcal{D}_u$ tels que les informations stockées au nœud u concernant v sont incorrectes. Si u est le plus proche nœud incorrect pour le nœud v, alors il existe un nœud $u' \in \Gamma(u)$ appartenant à un plus court chemin de u à v et stockant des informations correctes concernant le nœud v (il est important de remarquer qu'un nœud stocke nécessairement des informations correctes à son propre sujet)

Si u stocke un triplet (v, d, hd) alors il a accepté ce triplet et l'a donc envoyé via un message $\mathrm{mBellmanFord}(v, d, hd, -)$ à ses voisins. Le nœud u a donc reçu un triplet $(v, d + \omega(u, u'), hd + 1)$. Comme le nœud u' est sur un plus court chemin de u à v, nous avons $d + \omega(u, u') = d(u, v)$ et $hd + 1 = hd(u, v)$. Donc, si u stocke une information incorrecte à propos du nœud v il va mettre son entrée à jour. Pour conclure cela il est tout de même nécessaire d'observer la propriété évidente suivante, aucun message ne contient une distance plus petite que la distance réelle dans le graphe. Il y a une contradiction, toute entrée est donc correcte après stabilisation. Il en va bien évidement de même pour la table de routage-direct. □

Proposition 3.7. *Au temps t_{BF}, tout nœud $u \in V$ a sa table de routage-direct $\mathcal{D}_u$, ronde.*

Preuve. Durant la première phase de construction des tables de routage-direct, tout nœud $u \in V$ ajoute tout nœud nouveau qui lui est proposé. Puis, une fois que la table $\mathcal{D}_u$ est complète, le nœud u supprime les nœuds non-représentant se trouvant plus loin que le représentant $w_{\max}$ le plus éloigné de u, de plus, il ajoute tout nœud strictement plus proche que $w_{\max}$ de u. Donc, si u n'est pas ronde cela signifie qu'il existe des nœuds pour lesquels u n'a reçu aucun message $\mathrm{mBellmanFord}$. Par conséquent, en utilisant le même type d'arguments que dans la preuve de la proposition 3.6 il est possible de montrer que $\mathcal{D}_u$ est nécessairement ronde.

□

∎

Nous souhaitons que la terminaison de construction des arbres soit détectée par la racine de l'arbre, le lemme et le corollaire suivants s'intéressent à cette détection :

Lemme 3.7. *Pour tout nœud $v \in V$, tout nœud $u \in \mathrm{InvTree}_v$ au temps t_{BF} finit par recevoir $\mathrm{degree}(u)$ messages $\mathrm{mAckBellmanFord}(v, d, hd, -)$, avec d et hd les distance et hop-distance entre u et v.*

Preuve. Fixons v et considérons l'arbre $\mathrm{InvTree}_v$ au temps t_{BF}. Pour prouver ce lemme, il est suffisant de prouver, par induction ascendante dans l'arbre, que tout nœud u de l'arbre $\mathrm{InvTree}_v$ envoie effectivement un message $\mathrm{mAckBellmanFord}(v, d, hd, -)$ a son parent w, avec d et hd les distance et hop-distance entre v et w. ∎

Corollaire 3.2. *Pour tout nœud v, il existe un temps t_v auquel v reçoit son δ-ième message $\mathrm{mAckBellmanFord}(v, 0, 0, -)$, avec $\delta = \mathrm{degree}(v)$.*

Admettons $l_{\min}$ comme étant le landmark d'identifiant minimum dans G. Pour détecter la terminaison de construction des boules de voisinage nous devons avoir un garantie sur l'état de celles-ci au moment où $l_{\min}$ reçoit le dernier acquittement :

Lemme 3.8. *Il existe un temps t_1 auquel $l_{\min}$ reçoit son δ-ième et dernier message $\mathrm{mAckLeader}$, avec $\delta = \mathrm{degree}(l_{\min})$. De plus, à cet instant t_1, tout nœud u vérifie $\mathrm{lmin}_u = l_{\min}$ et toutes les tables de routage-direct et représentant sont dors et déjà dans le même état qu'à l'instant t_{BF} et resterons dans cet état.*

Preuve. Considérons un nœud $v \in V$, et un nœud u quelconque appartenant à $\mathrm{InvTree}_v$ a l'instant t_{BF}. Considérons l'acceptation du triplet (v, d, hd), le dernier triplet concernant v accepté par u, et le chemin causal $u_1 = v, u_2, \cdots, u_{i-1}, u_i = u$. Comme toutes les entrées des tables de routage-direct sont correctes au temps t_{BF}, et (v, d, hd) est le dernier triplet concernant v accepté par u, alors, pour tout $1 \leqslant j \leqslant i$, l'entrée $\mathcal{D}_{u_j}[v]$ est conservée inchangée après que la susdite chaîne de messages ait atteint u_j. Cela signifie que que le nœud v reçoit tous ses messages $\mathrm{mAckBellmanFord}(v, 0, 0, -)$ par un convergecast de messages $\mathrm{mAckBellmanFord}$ en provenance d'au moins tous les nœuds de $\mathrm{InvTree}_v$ (au temps t_{BF}). Par conséquent, au temps $\max_{v \in V} t_v$, tout nœud u stocke (au moins) tous les nœuds qui, à terme, appartiendrons à $\mathcal{D}_u$. Par conséquent, toutes les tables de routage-direct sont déjà

dans leurs états finaux à cet instant (toutes les suppressions de nœuds se font localement et donc en temps 0). Notons à présent, que pour tout landmark l, les message de convergecast $\mathrm{mAckBellmanFord}$ concernent tout le graphe. C'est pourquoi, tout landmark l va avoir $\mathrm{lmin}_l = l_{\min}$ au plus tard au temps t_l. Dès l'instant $t_{l_{\min}}$, le nœud $l_{\min}$ va initier la diffusion (via des messages $\mathrm{mLeader}$) suivie d'un convergecast (via des messages $\mathrm{mAckLeader}$) dans l'arbre $\mathrm{InvTree}_{l_{\min}}$, qui couvre tout le graphe. D'après le corrolaire 3.2, $l_{\min}$ va, à terme, collecter $\mathrm{degree}(l_{\min})$ messages $\mathrm{mAckLeader}$, le dernier à arrivant à un temps t_1. D'une part, grâce à la diffusion, tout nœud u vérifie $\mathrm{lmin}_u = l_{\min}$ au temps t_1 (et pour toujours). D'autre part, grâce au convergecast, durant lequel chaque nœud v attend le temps t_v pour envoyer son accusé, nous avons $t_1 \geqslant \max_{v \in V} t_v$. ∎

Lemme 3.9. *Le temps t_1 est d'au plus $4D$.*

Preuve. La complexité en temps de l'algorithme Bellman-Ford modifié a évidement la même complexité en temps que l'algorithme classique. Tous les nœuds ont donc leurs tables de routage-direct dans leur état final après au plus D unités de temps, donc plus aucun message $\mathrm{mBellmanFord}$ ne circule après D unités de temps. L'algorithme de convergecast effectué par les messages $\mathrm{mAckBellmanFord}$ commence au plus tard lorsqu'aucun message $\mathrm{mBellmanFord}$ ne circule et cet algorithme a une complexité en temps de D. Après $2D$ unités de temps ces deux algorithmes sont terminés. Les deux derniers algorithmes sont une diffusion et un autre convergecast, la complexité en temps totale de la phase 1 est donc de $4D$. ∎

Après, au plus, $4D$ unités de temps après le réveil du nœud $l_{\min}$, pour tout nœud u, les éléments de la table $\mathcal{D}_u$ sont stabilisés et les champs $\mathrm{nexthop}, d, hd, c$ sont également stables. De plus, nous avons dès lors $\mathrm{lmin}_u = l_{\min}$.

3.5.4 Mémoire de travail dans le modèle de coloration aléatoire

Si les couleurs des nœud sont choisies de manière aléatoire uniforme, certaines propriétés sur la taille des tables et sur le nombre de nœuds d'une couleur donnée apparaissent. Cela ayant un impact direct sur les performances mémoire de l'algorithme $\mathrm{DCR}(k)$.

Commençons par quelques rappels :

Lemme 3.10 (Le problème du collectionneur de coupons [Fel74])**.** *Considérons une sélection aléatoire avec remise dans une collection de k coupons différents, chaque coupon ayant la même probabilité d'être collecté. Notons Tr le nombre de sélections requises pour collecter chaque coupon au moins une fois. L'espérance de la variable aléatoire Tr est de :*

$$\mathbb{E}(Tr) = k \cdot H_k = k \ln k + \gamma k + \frac{1}{2} + \mathcal{O}(1), \quad \textit{if } k \to \infty$$

avec H_k le nombre harmonique de rang k et $\gamma \approx 0.5772156649$ est la constante d'Euler–Mascheroni. Il est possible de borner la queue de la distribution de Tr par :

$$\Pr[T > \beta k \ln k] \leqslant k^{-\beta+1}$$

Par la suite, nous supposerons que k est polynomial en n, plus précisément $k = n^\alpha$ avec $0 < \alpha < 1$. Posons V_u comme étant l'ensemble de nœud ordonnés selon leurs distance à u. L'ensemble V_u est *complet* si il contient au moins un représentant de chaque couleur. Si V_u est complet, alors une fonction, dite de *réduction*, qui consiste

à donner un préfixe (V'_u) minimal de V_u tel que V'_u contient au moins un représentant par couleur. Le *rayon* de la réduction r_u est la distance entre u et le dernier nœud du préfixe V'_u.

Lemme 3.11 (Taille des boules de voisinage)**.** *Notons V_u^t un ensemble arbitraire de t nœuds construit par un processus impédant de la coloration. Sous l'hypothèse de coloration aléatoire, pour tout t, la boule de voisinage $B_u(t)$ de tout nœud $u \in V$ définie comme la réduction de V_u^t a une taille inférieure à $(\beta+1)k\ln k$ avec grande probabilité. Nous appelons grande probabilité $1-\frac{1}{n^2}$ avec $\beta \geqslant \frac{2+\alpha}{\alpha}$.*

Preuve. Rappelons que l'adversaire considéré est oublieux, il ne connaît donc pas le choix de couleurs (aléatoire) fait par les nœuds. D'après le lemme 3.10, l'ensemble V_u^t se trouve être complet lorsque t est plus petit que $\beta k\ln k$ avec probabilité $1-k^{-\beta+1}$. Avant cet instant $\mathrm{B}(u) = V_u$. Prouvons que, pour tout t, $|B_u(t)| < (\beta+1)k\ln k$ avec grande probabilité. En raison de la propriété de minimalité, remarquons que le nombre de nœud dans $\mathrm{B}(u)$ à distance r_u est au plus de k à tout moment. Un nouveau nœud est inséré dans $\mathrm{B}(u)$ si et seulement si sa distance à u est strictement plus petite que r_u. Supposons que $|\mathrm{B}(u)| > (\beta+1)k\ln k$. Cela signifie que $\mathrm{B}(u)$ contient au moins $(\beta+1)k\ln k - k > \beta k\ln k$ nœuds à distance inférieure à r_u, et ces nœuds ne définissent pas un ensemble complet. D'après le lemme 3.10, cet événement survient avec une probabilité inférieure à $k^{\beta+1} = n^{\alpha(-\beta+1)} = \mathcal{O}\big(n^{-2}\big)$ si $\beta > \frac{2+\alpha}{\alpha}$, ∎

Par conséquent, pour tout temps, la mémoire de travail requise pour stocker $\mathrm{B}(u)$ pour tout nœud u est inférieure à $(\beta+1)k\ln k$.

Lemme 3.12 (Nombre de nœuds d'une couleur donnée)**.** *Sous l'hypothèse de coloration aléatoire, si $n \geqslant (24\ln n)^{\frac{1}{1-\alpha}}$ alors le nombre de nœuds pour toute couleur $i \in [1,k]$ le nombre de nœuds ayant cette couleur est compris entre $\frac{n}{2k}$ et $\frac{3n}{2k}$ avec une probabilité supérieure à $1-2/n^2$.*

Preuve. Considérons les variables aléatoires suivantes $X_1,\ldots,X_n$ définies par $\Pr(X_i=1) = 1/k$. Pour une couleur donnée i, la valeur de X_j vaut 1 si le nœud j est de couleur i, 0 sinon. Les variables X_j étant indépendantes nous pouvons utiliser la borne de Chernoff pour borner la somme des X_j. Si $X = \sum_{i=1}^{n} X_i$ et μ est l'espérance de X, alors pour $0<\delta<1$, la borne de Chernoff implique que :

$$\Pr(|X-\mu| \geqslant \delta\mu) \leqslant 2e^{-\frac{\delta^2\mu}{3}}$$

Posons $\delta = 1/2$. Pour obtenir $\frac{\delta^2 n^{1-\alpha}}{3} \geqslant 2\log n$ il suffit d'avoir $n \geqslant (24\log n)^{\frac{1}{1-\alpha}}$. ∎

Lemme 3.13. *Sous l'hypothèse de coloration aléatoire, à tout instant durant l'exécution et pour tout nœud $u \in V$, le nombre d'entrées dans $\mathcal{D}_u$ est de $\mathcal{O}\big(n/k + \frac{2+\alpha}{\alpha}k\log k\big)$ avec grande probabilité.*

Preuve. La taille de la table de routage-direct d'un nœud est basé sur trois choses, la taille de $\mathcal{M}_u$ (qui a une taille exactement k), le nombre de nœuds landmark et la taille de la boule de voisinage B_u.
D'après le lemme 3.12, il y a $\Theta(n/k)$ landmarks avec probabilité $1-1/n^2$ et d'après le lemme 3.11, la taille des boules de voisinage est de $\mathcal{O}\big(\frac{2+\alpha}{\alpha}k\log k\big)$. ∎

3.5.5 Modèle synchrone et coloration aléatoire

Cette section traite la question du coût de communication requis pour construire les tables de routage-indirect sous les hypothèses de coloration aléatoire des nœuds. Informellement, le nombre total de message est lié au nombre de changement d'état de chaque table $\mathcal{D}_\cdot$.

L'analyse d'un algorithme asynchrone dans un environnement parfaitement synchrone nécessite de poser quelques définitions sur la manière d'exécuter cet algorithme, nous supposons que le processus suivant est appliqué :

Définition 3.10 (Simulation d'algorithme synchrone)**.** *Tout algorithme asynchrone peut simuler une exécution synchrone en greffant la méthode de simulation constituée des règles suivantes :*

- *il y a une horloge de synchronisation globale;*
- *le délai de transmission des messages est plus petit que l'intervalle de temps entre deux tops d'horloge;*
- *entre deux tops d'horloge consécutifs, tout nœud u stocke et ordonne les messages reçus selon leurs distances avec u annoncées par ces messages;*
- *à chaque top d'horloge, les nœuds exécutent l'algorithme asynchrone en consommant les messages par distance croissante, de plus tout calcul local est fait instantanément.*

Le lemme suivant sera utilisé pour caractériser le nombre de changements d'états possible pour une route donnée découverte. En effet, dans le cas pondéré, il est possible même lors d'une exécution synchrone de trouver des routes moins bonnes plus tôt. La figure 3.4 montre un exemple dans lequel les chemins seront découverts par un nœud dans l'ordre inverse de leur qualité en terme de ratio longueur du chemin trouvé par rapport à la distance réelle dans le graphe.

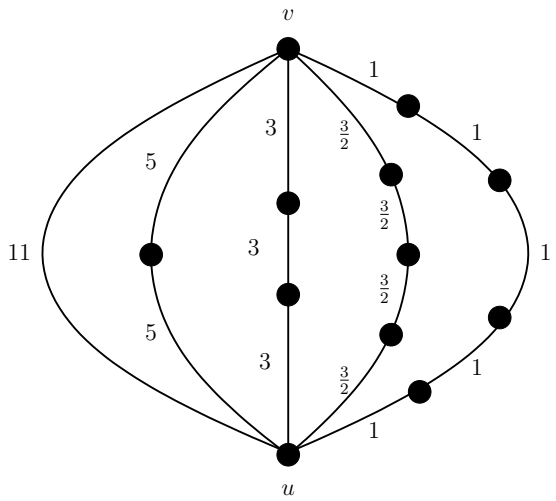

Figure 3.4: Les chemins sont appris par le nœud u de la gauche vers la droite, la longueur totale des chemins ainsi appris est strictement décroissante (dans l'ordre $(11, 10, 9, 6, 5)$) alors que la hop-longueur est strictement croissante (dans l'ordre$(1, 2, 3, 4, 5)$). Dans ce graphe une exécution synchrone de notre algorithme implique donc 5 changements d'état du nœud u pour la destination v.

Lemme 3.14. *Étant donné deux chemins de longueurs respectives d et d', et de hop-longueurs respectives hd et hd'. Si $d \geqslant d'$, but $hd \leqslant hd' \leqslant D$, alors $hd' - hd \leqslant \xi - 1$.*

Preuve. Pour $W_{\min}$ et $W_{\max}$ le plus petit et le plus gros poids d'arêtes du graphe, nous avons :

$$\begin{aligned} hd &\geqslant d/W_{\max} \\ &\geqslant d'/W_{\max} \\ &\geqslant (hd' \cdot W_{\min})/W_{\max} \\ &\geqslant hd'/W \end{aligned}$$

Par conséquent :

$$hd' - hd \leqslant hd'(1 - 1/W) \leqslant D(1 - 1/W)$$

Ce qui ce traduit finalement par $hd' - hd \leqslant \xi - 1$. ∎

Lemme 3.15 (Coût de communication)**.** *Le nombre de messages envoyés durant la phase 1 est au plus de $\mathcal{O}(\xi m(k \log k + n/k))$ avec grande probabilité.*

La preuve s'appuie sur la figure 3.5.

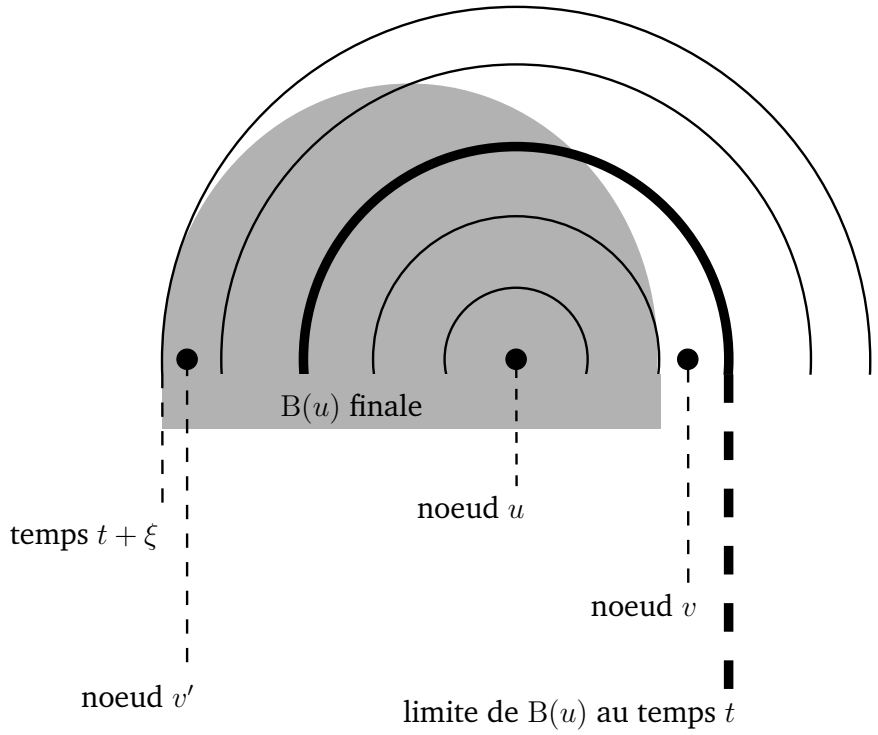

Figure 3.5: Étapes de la construction de la boule du nœud u.

Preuve. Quatre types de messages sont envoyés durant cette phase. D'après le lemme 3.8, seul le nœud $l_{\min}$ va initier la diffusion consistant de messages $\mathrm{mLeader}$. Le nombre total de messages de ce type est donc d'au plus m. Par conséquent, le nombre total de messages de type $\mathrm{mAckLeader}$ est également borné par m. De la même façon, le nombre de messages de type $\mathrm{mAckBellmanFord}$ est borné supérieurement par le nombre de messages de type $\mathrm{mBellmanFord}$. Nous donnons par la suite une borne supérieure sur le nombre de messages de ce dernier type.

Considérons un nœud arbitraire u, et prouvons que u accepte $\mathcal{O}(\xi(k \log k + n/k))$ messages $\mathrm{mBellmanFord}$ durant l'exécution, avec grande probabilité. Comme tout message accepté génère l'émission de $\mathrm{degree}(v)$ messages, cette borne permettra de déduire la borne supérieur proposée dans le lemme.

Commençons par remarquer que les triplets (v, d, hd) sont reçu par ordre croissant de hd, à cause du modèle synchrone, mais également par ordre croissant de d pour v et hd fixés (d'après la définition du modèle d'algorithme synchrone simulé). Le lemme 3.4 implique qu'un nœud u accepte au plus un triplet concernant un nœud v par hop-distance. Le lemme 3.14 implique quant à lui que tout nœud v ne peut pas faire partie d'un triplet accepté par le nœud u plus de ξ fois.

Les triplets acceptés par le nœud u peuvent être séparés en deux groupes :

(1) ceux qui ont été acceptés avant que la boule de voisinage du nœud u soit complète,

(2) et les autres, acceptés dans le but de rendre la boule ronde (attention la boule est ronde en terme de distances, ce qui explique que la boule finale sur la

figure 3.5 ne parait pas ronde bien qu'elle le soit).

Comme les communications sont synchrones, les triplets du premier groupe annoncent des hop-distances plus petites que celle du second.

Notons t le temps auquel $\mathrm{B}(u)$ devient complète. Voici l'analyse du nombre messages de chacun de ces deux groupes :

(1) D'après le lemme 3.11, à l'instant t, la boule de voisinage $\mathrm{B}(u)$ contient au plus $\frac{\alpha+2}{\alpha} k \log k$ nœuds avec probabilité $1 - n^{-2}$. Et donc, d'après le lemme 3.14, le nombre de triplets acceptés avant l'instant t est d'au plus $\mathcal{O}(\xi \cdot k \log k)$ avec grande probabilité.

(2) À partir du temps t, les triplets acceptés ne concernant pas un landmark contiennent une distance inférieure au rayon de la boule de u au temps t. (le rayon devient fini au temps t et ne décroit plus après cet instant.) Ainsi, d'après le lemme 3.14, des triplets sont encore acceptés par u pendant au plus ξ rondes synchrones, du temps t au temps $t + \xi$. Autrement dit, un nœud v' accepté après l'instant t est au plus à distance ξ de tout nœud v accepté avant l'instant t. Donc le nœud v' est au plus à distance $t + \xi$ de u (voir figure 3.5).

Pour toute ronde $t < t' \leqslant t + \xi$. Comme les triplets sont considérés par le nœud u par ordre de distance croissante, le nœud u considérera dans la ronde t' au plus un triplet par couleur, n'ayant en effet, aucun intérêt à en consulter plus, ayant déjà à ce stade tous les meilleurs représentants de chaque couleur à distance t' (remarquons tout de même que cette remarque ne s'applique pas aux nœud landmarks qui seront systématiquement considérés). D'après le lemme 3.11, le nœud u va accepter durant la ronde t' au plus $\mathcal{O}(k \log k)$ triplets ne concernant pas des nœuds landmark, avec grande probabilité. Le nœud u va donc envoyer $\mathcal{O}(\mathrm{degre}(u) \cdot \xi k \log k)$ messages $\mathrm{mBellmanFord}$ durant cette phase pour construire sa boule de voisinage.

En ce qui concerne les entrées de $\mathcal{D}_u$ pour les landmarks, d'après le lemme 3.12, il y a au plus $3n/k$ landmarks avec grande probabilité. Donc, d'après la remarque sur le nombre de triplets acceptés, tout nœud landmark sera accepté au plus ξ fois. Le nombre total de messages $\mathrm{mBellmanFord}$ émis par u pour les nœuds landmarks est donc d'au plus $\mathcal{O}(\mathrm{degre}(u) \cdot \xi \cdot n/k)$, avec grande probabilité. En sommant sur l'ensemble des nœuds $u \in V$, le nombre total de messages pour la phase 1 est donc de :

$$\begin{aligned} \mathsf{Msg_s} &= \sum_{u \in V} \mathcal{O}(\mathrm{degre}(u) \cdot \xi k \log k) + \mathcal{O}(\mathrm{degre}(u) \cdot \xi \cdot n/k) \\ &= \mathcal{O}(\xi m (k \log k + n/k)) \end{aligned}$$

∎

3.6 Phase 2: Calcul des étiquettes de routage dans les arbres (table $\mathcal{D}_u$)

L'objectif de cette phase est de construire, pour tous nœuds $u \in V$ et $v \in \mathcal{D}_u$, l'étiquette de routage de u dans $\mathrm{InvTree}_v$. Pour faire cela, nous utiliserons les fonctions de calcul d'étiquettes locales proposées par le lemme 3.16 (section 3.6.1) combiné avec des diffusions et des convergecasts qui seront utilisées pour :

- calculer le poids de l'ensemble des sous-arbres de $\mathrm{InvTree}_v$;
- puis calculer les étiquettes de routage de l'ensemble des nœuds de $\mathrm{InvTree}_v$.

L'utilisation d'étiquettes de routage permet de router dans des arbres en plus court chemin sans que les nœuds ne connaissent leurs enfants dans ces différents arbres, cela permettra dans notre cas d'obtenir des entrées dans la table $\mathcal{I}$ de taille poly-logarithmiques. Dès qu'un nœud connaît le poids de l'ensemble des sous-arbres de ses enfants, ainsi que l'étiquette de son parent dans l'arbre il est capable de calculer localement sa propre étiquette. La première section donne le détail des calculs effectués localement par les nœuds et de la connaissance requises pour ce calcul.

3.6.1 Fonctions de calcul d'étiquettes de routage

Considérons un arbre de plus court chemin T de G enraciné en un nœud r. Notons que dans T le chemin entre toute paire de nœuds est inférieur à $2D$. Tout nœud peut calculer une étiquette de routage de $\mathcal{O}(\min\{D, \log n\} \cdot \log n)$ bits tel que le routage dans T peut être réalisé par plus court chemin en utilisant uniquement ces étiquettes et une entête de message de la même taille. L'algorithme de calcul des étiquettes est adapté de l'algorithme décrit dans [FG01b] qui permet de calculer des étiquettes de routage de la même taille. Cependant, la solution proposée dans [FG01b] ne peut pas être appliquée directement dans un environnement distribué, et tout particulièrement dans le cas présent où une complexité en temps de $\mathcal{O}(D)$ est recherchée. En effet le calcul originel utilise une numérotation des nœuds suivant l'ordre DFS, or un parcours en profondeur prend un temps linéaire. Par conséquent, quelques changements sont apportés à cet algorithme de routage pour éviter ces problèmes.

Dans le but de construire l'étiquette de routage $\ell(u, T)$ d'un nœud u dans T, il est nécessaire de déterminer le poids du nœud u dans T (son nombre de descendants dans T), ainsi que son *fils le plus lourd*, i.e. le fils ayant le plus gros poids dans T. Le chemin compact path^*_u de la racine au nœud u peut alors être calculé. Ce chemin compact est la séquence d'identifiants de nœuds de r à u dans T dans laquelle tout identifiant correspondant à un fils lourd est remplacé par une étoile $*$. L'étiquette de routage $\ell(u, T)$ est composée de

(1) le chemin cpath_u qui est path^*_u dans lequel toute séquence d'étoile est remplacée par sa propre longueur ;

(2) un bit-set b_u qui permet de déterminer si un élément de cpath_u est soit un nœud soit une longueur de séquence d'étoiles ;

(3) l'identifiant de son fils le plus lourd. Un exemple d'une telle étiquette de routage est donné dans la table 3.2.

path in T	$u_0 = r$	u_1	u_2	u_3	u_4	$u_5 = u$
path^*_u	u_0	u_1	*	*	u_4	*
cpath_u	u_0	u_1	2		u_4	1
b_u	1	1	0		1	0

Table 3.2: Un exemple simple de $\ell(u, T)$ après calcul en considérant que les fils lourds de u_1, u_2 et u_4 sont respectivement u_2, u_3 et u_5. Les fils lourds des autres nœuds étant des nœuds n'appartenant pas à $\{u_1, u_2, u_3, u_4, u_5\}$.

Ces étiquettes de routage peuvent être calculées récursivement en partant de la racine r vers les feuilles de T en utilisant les deux fonctions suivantes :

- La fonction $\text{rootLabel}(u, v)$ renvoie l'étiquette de routage d'un nœud u dans un arbre dans lequel u est la racine et v est son fils le plus lourd.
- La fonction $\text{computeLabel}(u, \texttt{label}, v)$ renvoie l'étiquette de routage d'un nœud u dans un arbre tel que le parent de u a l'étiquette `label` et le fils lourd de u est v.

Le calcul distribué de ces étiquettes est fait lors de la phase 2 et utilise ces deux primitives. La phase 2 est décrite en section 3.6.

L'algorithme de routage au nœud u pour la destination v est effectué comme suit. Le nœud u utilise $\ell(u, T)$ et $\ell(v, T)$ pour déterminer le prochain saut vers v. Pour faire court, en utilisant ces étiquettes, le nœud u peut déterminer la position approximative de v dans T. Pour faire cela, u doit d'une certaine manière trouver le plus long préfixe commun de $\ell(u, T)$ et $\ell(v, T)$. Cela requiert en fait deux calculs : le nœud u doit trouver le plus long préfixe commun des deux bit-set b_u et b_v et le plus long préfixe commun de cpath_u et cpath_v. Une fois ces deux opération effectuées, u peut déterminer si v est un de ces descendants ou non. Si v n'est pas un descendant, u se contente d'envoyer le message à son père. Dans le cas contraire, en utilisant le premier élément divergeant de b_v (par rapport à b_u) le nœud u peut déterminer si le prochain saut vers v est son fils le plus lourd ou non. S'il l'est, le nœud u connaît l'identifiant de son fils le plus lourd et peut donc router vers celui-ci. En revanche, s'il ne l'est pas, u doit dans ce cas utiliser l'étiquette de routage de v dans T ($\ell(v, T)$), contenue dans l'entête du paquet. Ainsi, le nœud u peut router vers n'importe quel nœud v dans T en utilisant uniquement $\ell(u, T)$ et $\ell(v, T)$. Dans le schéma de routage distribué, le mécanisme de routage ci-dessus sera formalisé par la primitive suivante :

- La fonction $\text{decodeNexthop}(\texttt{label}_1, \texttt{label}_2)$ permet de calculer le prochain saut vers le nœud d'étiquette $\texttt{label}_1$ en utilisant l'étiquette du nœud local $\texttt{label}_2$.

Le lemme suivant résume les résultats de cette section :

Lemme 3.16. *Étant donné un arbre pondéré T de hauteur h enraciné en un nœud r, les fonctions* rootLabel *et* computeLabel *permettent de calculer récursivement depuis la racine vers les feuilles, l'étiquette de routage $\ell(u, T)$ pour tout nœud u, avec les prorpriétés suivantes :*

- *La taille de $\ell(u, T)$ est de $\mathcal{O}(\min\{h, \log n\} \cdot \log n)$ bits.*

- *La fonction* $\mathrm{decodeNexthop}$*, en utilisant deux étiquettes de routage* $\texttt{label}_1$ *et* $\texttt{label}_2$ *comme paramètres, calcule en temps constant le prochain saut vers le nœud d'étiquette de routage* $\texttt{label}_1$*lors du routage depuis le nœud d'étiquette* $\texttt{label}_2$.

3.6.2 Structures de données de la phase 2

L'ensemble de clef-valeur $\mathrm{LogRoot}_u$ a une couleur comme clef. Pour toute clef i, les valeurs suivantes sont stockées :

- node est un nœud, destiné à être la racine de l'arbre logique de couleur i.
- label est une étiquette de routage, destinée à être l'étiquette de routage du nœud stocké dans $\mathrm{LogRoot}_u[v].\mathrm{node}$ dans l'arbre du landmark $l_{\min}$.

3.6.3 Algorithmes distribués de calcul d'étiquettes de routage

Les algorithmes distribués de cette phase ne présentant pas de difficultés ou d'originalités particulière, ils seront détaillés en annexe A.1. Le principe de cette phase est le suivant :

1. Le landmark $l_{\min}$ effectue une diffusion dans $\mathrm{InvTree}_{l_{\min}}$ pour prévenir tous les nœuds que la phase 2 peut commencer.

   L'algorithme 15 détaille ces opérations.

2. Tout nœud v étant averti de cela, entame une diffusion dans $\mathrm{InvTree}_v$ suivi d'une remonté d'acquittements pour calculer les poids des différents sous-arbres de $\mathrm{InvTree}_v$.

   Les algorithmes 16 et 17 détaillent ces opérations.

3. Suite à la détection de terminaison des calculs des poids dans son arbre, le nœud v initie le calcul des étiquettes. Ce calcul se fait encore une fois par une diffusion dans $\mathrm{InvTree}_v$.

   L'algorithme 18 détaille ces opérations, la technique utilisée est classique et appelée *calcul de fonction globale* [ABP90].

4. Dès qu'un nœud u détecte que toutes ses étiquettes sont calculées il envoie un acquittement à $l_{\min}$ dans $\mathrm{InvTree}_{l_{\min}}$. Le nœud u calcule une étiquette par entrée dans $\mathcal{D}_u$, il est donc simple pour lui de détecter la fin de ce calcul.

   L'algorithme 19 détaille ces opérations.

5. Enfin, dès que le landmark $l_{\min}$ a reçu tous ses acquittements il prévient tous les nœuds que la phase 2 est terminée par une diffusion dans $\mathrm{InvTree}_{l_{\min}}$.

   L'algorithme 20 détaille ces opérations. Notons tout de même un détail important de cet algorithme, dès qu'un nœud u reçoit ce dernier type de message :

   - Il envoie un message :

     $$\mathrm{mLogicalParent}(u, \mathcal{D}_u[\mathrm{lmin}_u].\mathrm{label}, c(u))$$

     à son parent dans l'arbre $\mathrm{InvTree}_{l_{\min}}$, ce qui permet d'initier la construction de l'arbre logique de la phase 3.1 (section 3.7).

- Ainsi que le message :

  $$\mathrm{mLogicalContact}(v, u, \mathrm{closestLandmark}_u, \mathcal{D}_u[\mathrm{closestLandmark}_u].\mathrm{label})$$

  au représentant de $v' = \mathcal{M}_u[\mathrm{h}(u)].\mathrm{node}$, initiant ainsi l'envoi de l'étiquette de routage de u aux nœuds de couleur $\mathrm{h}(u)$ (section 3.7).
- Enfin, le nœud u envoie, le message :

  $$\mathrm{mLocalBroadcast}(v, u, \mathcal{D}_u[v].\mathrm{label})$$

  à tout nœud $v' \in \mathrm{B}(u)$, ce qui permet d'initier la phase 3.2 (section 3.8).

Par la suite nous utiliserons le temps auquel finit de cette phase :

Définition 3.11. *Notons $t_2 > t_1$ le temps auquel le dernier message de cette phase est reçu.*

3.6.4 Analyses de la phase 2, indépendamment du modèle de coloration

Les lemmes de cette section sont valables quelque soit la coloration des nœuds du graphe. Le lemme suivant est une conséquence directe du lemme 3.16 :

Lemme 3.17. *Au temps t_2, tout nœud v stocke pour tout $u \in \mathcal{D}_v$ une étiquette de routage $\mathcal{D}_v[u].\mathrm{label}$ qui encode un plus court chemin dans $\mathrm{InvTree}_u$.*

Preuve. D'après [ABP90], la fonction globale permettant de calculer les poids des sous-arbres est correcte. Donc, d'après le lemme 3.16 et par récurrence depuis la racine vers les feuilles, il est aisé de montrer que les étiquettes calculées encodent des plus court chemins dans les arbres. ∎

Lemme 3.18. *La complexité en temps de la phase 2 est de* $\mathsf{Time}(\textsc{Phase2}) = 6D$.

Preuve. Les algorithmes utilisés dans cette phase sont uniquement des diffusions et des convergecasts. Plus précisément, sont fait en série, trois diffusions et trois convergecats, le temps total de cette phase est donc de $6D$. ∎

3.6.5 Sous l'hypothèse de coloration aléatoire, phase 2

Le lemme suivant n'est valable que sous l'hypothèse où les nœuds obtiennent leurs couleurs par un choix aléatoire uniforme dans $[1, k]$.

Lemme 3.19. *Le coût de communication de cette phase est de $\mathcal{O}(m(k \log k + n/k))$ messages.*

Preuve. Cette preuve ne cache pas de réelle difficulté, elle somme simplement les coût de diffusion dans les différents arbres considérés :

- la diffusion et le convergecast dans l'arbre $\mathrm{InvTree}_{l_{\min}}$ requirent tous les deux $\mathcal{O}(m)$ messages. Ces deux algorithme sont exécutés une seule fois (algorithmes 15 et 19), plus tard $l_{\min}$ diffuse également la détection de terminaison de la phase 2 dans $\mathrm{InvTree}_{l_{\min}}$ (algorithme 20);

- tout nœud u initie également une diffusion/convergecast dans son propre arbre $\mathrm{InvTree}_u$ dans l'objectif de calculer les poids des sous-arbres de $\mathrm{InvTree}_u$ (algorithmes 16 et 17), il initie également le calcul des étiquettes de routage en diffusant des étiquettes de routage dans $\mathrm{InvTree}_u$ (Algorithm 18);

Toute diffusion et convergecast dans un arbre $\mathrm{InvTree}_u$ a un coût de $\mathcal{O}\big(\sum_{v\in\mathrm{InvTree}_u}\mathrm{degre}(v)\big)$ messages. Tout nœud effectue un nombre constant de ces deux algorithmes, donc le nombre total de messages envoyés dans la phase 2 est :

$$\begin{aligned}
\mathsf{Msg_s}(\textsc{Phase2}) &= \sum_{u\in V}\mathcal{O}\left(\sum_{v\in\mathrm{InvTree}_u}\mathrm{degre}(v)\right)\\
&= \sum_{u\in V\setminus L}\mathcal{O}\left(\sum_{v\in\mathrm{InvTree}_u}\mathrm{degre}(v)\right)+\sum_{l\in L}\mathcal{O}\left(\sum_{v'\in\mathrm{InvTree}_l}\mathrm{degre}(v')\right)\\
&= \sum_{u\in V\setminus L}\mathcal{O}(m\cdot|\mathrm{B}(u)|)+\mathcal{O}\Big(m\cdot\frac{n}{k}\Big)\\
&= \mathcal{O}(m\cdot(k\log k))+\mathcal{O}\Big(m\cdot\frac{n}{k}\Big)=\mathcal{O}(m(k\log k+n/k))
\end{aligned}$$

∎

3.7 Phase 3.1: Construction d'arbres logiques et diffusions dans ces arbres (table $\mathcal{I}_u$)

Cette phase a pour objectif, pour toute couleur $i\in[1,k]$, d'échanger les labels de routage concernant les nœud destination dont la valeur hachée est égale à i entre les nœuds de couleur i :

1. Pour arriver à effectuer ces échanges de manière économe en nombre de messages, le premier objectif est de construire la structure permettant cet échange, autrement dit, construire, pour toute couleur $i\in[1,k]$, un arbre logique $\mathcal{T}_i$. Les nœuds de $\mathcal{T}_i$ sont les nœuds de l'ensemble V_i, i.e. l'ensemble des nœuds de couleur i. Cet arbre logique sera, plus tard, utilisé par les nœuds i pour se communiquer des informations de routage indirect vers les nœuds de valeur hachée i.

   Sans entrer dans les détails, une arête entre deux v et v' dans $\mathcal{T}_i$ représente un chemin de v à v' dans $\mathcal{T}_i$ sans aucun nœud intermédiaire de couleur i. L'arbre logique $\mathcal{T}_i$ a une racine dont l'identifiant sera stocké par le nœud $l_{\min}$ dans $\mathrm{LogRoot}_{l_{\min}}[i].\mathrm{node}$. Pour tout nœuds $u,v\in V_i$, il existe une arête logique (u,v) ssi. :

   - u est le premier ancêtre du nœud v de couleur $c(v)$ (ou inversement)
   - ou u et v n'ont pas d'ancêtre de couleur $c(v)$ et u (ou v) est la racine logique $\mathrm{LogRoot}_{c(v)}$ de l'arbre $\mathcal{T}_{c(v)}$ stockée en $l_{\min}$.

   Une arête logique $\{u,v\}$ est stockée sous forme de deux arcs logiques, (u,v) et (v,u) respectivement stockés dans les tables $\mathrm{LogNeighbor}_u$ et $\mathrm{LogNeighbor}_v$ des nœuds u et v. L'arc (u,v) est l'étiquette logique de v dans $\mathrm{InvTree}_{l_{\min}}$, i.e. $\mathrm{LogNeighbor}_u[v].\mathrm{label}$ qui représente cet arc logique en mémoire est destiné à être égal à $\mathcal{D}_v[l_{\min}].\mathrm{label}$. Comme habituellement, l'ensemble des clefs de

$\mathrm{LogNeighbor}_u$ sera noté $\mathrm{LogNeighbor}_u$. Comme le montre la figure 3.6, les algorithmes 6 et 7 visent à construire les arêtes logique entre nœuds d'une couleur donnée i.

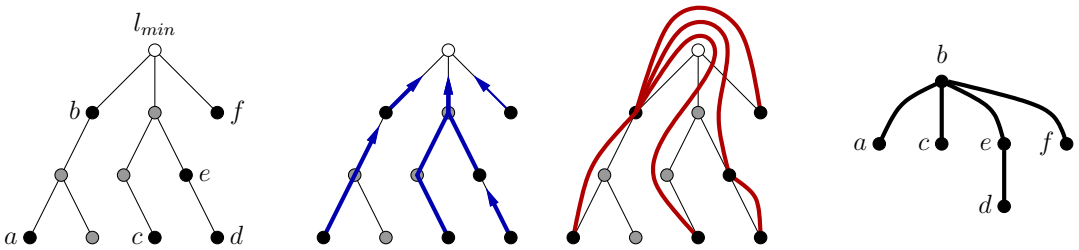

Figure 3.6: Construction de l'arbre logique $\mathcal{T}_i$ ($\mathrm{logicalRoot}_i = b$).

2. Dès lors qu'un arbre logique $\mathcal{T}_i$ est construit, les nœuds appartenant à celui-ci vont démarrer l'échange d'informations avec leurs voisins logiques :
   - concernant les étiquettes de routage qu'ils connaissent déjà (algorithme 9) ;
   - et concernant les nouvelles étiquettes de routage qu'ils apprennent par la suite (algorithme 8).

3.7.1 Structures de données, phase 3.1

La table de routage-indirect, $\mathcal{I}_u$, a des nœuds comme clefs. Elles sont destinées à être les nœuds dont la valeur hachée est égale à la couleur du nœud u. Pour toute clef $v \in \mathcal{I}_u$, les valeurs suivantes sont stockées

- $\mathrm{closestLandmark}$ est un nœud, destiné à être le plus proche landmark de v ;
- $\mathrm{closestLabel}$ est une étiquette de routage, destinée à être l'étiquette de routage de v dans l'arbre du nœud stocké dans $\mathcal{I}_u[v].\mathrm{closestLandmark}$;
- $\mathrm{bestLandmark}$ est un nœud, destiné à être le landmark qui permet d'obtenir la plus courte route de u à v dans l'arbre de ce landmark ;
- $\mathrm{bestLabel}$ est une étiquette de routage, destinée à être l'étiquette de routage du nœud v dans l'arbre stocké dans $\mathcal{I}_u[v].\mathrm{bestLandmark}$.

Algorithm 6: Réception par un nœud u d'un message $\mathrm{mLogicalParent}(v, \texttt{label}, i)$ provenant d'un nœud w

if $u \neq l_{\min}$ **then**
  if $c(u) \neq i$ **then**
    $v' \leftarrow \mathcal{D}_u[\mathrm{lmin}_u].\mathrm{nexthop}$
    $\mathrm{SEND}_{v'}(\mathrm{mLogicalParent}(v, \texttt{label}, i))$
  else
    $\mathrm{LogNeighbor}_u \leftarrow \mathrm{LogNeighbor}_u \cup \{v\}$
    $\mathrm{LogNeighbor}_u[v].\mathrm{label} \leftarrow \texttt{label}$
    $v' \leftarrow \mathrm{decodeNexthop}(\texttt{label}, \mathcal{D}_u[\mathrm{lmin}_u].\mathrm{label})$
    $\mathrm{SEND}_{v'}(\mathrm{mLogicalChild}(v, \texttt{label}, u, \mathcal{D}_u[\mathrm{lmin}_u].\mathrm{label}))$
    foreach $x \in \mathcal{I}_u$ *such that* $\mathcal{I}_u[x].\mathrm{closestLandmark} \neq \mathrm{null}$ **do**
      $\mathrm{SEND}_{v'}(\mathrm{mLogicalDiffusion}(v, \texttt{label}, x, \mathcal{I}_u[v].\mathrm{closestLandmark}, \mathcal{I}_u[v].\mathrm{closestLabel}, u))$
    end
  end
else
  if $\mathrm{LogRoot}_u[i].\mathrm{node} = \mathrm{null}$ **then**
    $\mathrm{LogRoot}_u[i].\mathrm{node} \leftarrow v$
    $\mathrm{LogRoot}_u[i].\mathrm{label} \leftarrow \texttt{label}$
  else
    $v' \leftarrow \mathrm{decodeNexthop}(\texttt{label}, \mathcal{D}_u[\mathrm{lmin}_u].\mathrm{label})$
    $\mathrm{SEND}_{v'}(\mathrm{mLogicalChild}(v, \texttt{label}, \mathrm{LogRoot}_u[i].\mathrm{node}, \mathrm{LogRoot}_u[i].\mathrm{label}))$
    $v' \leftarrow \mathrm{decodeNexthop}(\mathrm{LogRoot}_u[i].\mathrm{label}, \mathcal{D}_u[\mathrm{lmin}_u].\mathrm{label})$
    $\mathrm{SEND}_{v'}(\mathrm{mLogicalChild}(\mathrm{LogRoot}_u[i].\mathrm{node}, \mathrm{LogRoot}_u[i].\mathrm{label}, v, \texttt{label}))$
  end
end

Algorithm 7: Réception par un nœud u d'un message mLogicalChild($v_1, \texttt{label}_1, v_2, \texttt{label}_2$) provenant d'un nœud w

if $u \neq v_1$ **then**
  $v' \leftarrow$ decodeNexthop($\texttt{label}_1, \mathcal{D}_u[\text{lmin}_u]$.label)
  SEND$_{v'}$(mLogicalChild($v_1, \texttt{label}_1, v_2, \texttt{label}_2$))
else
  LogNeighbor$_u \leftarrow$ LogNeighbor$_u \cup \{v_2\}$
  LogNeighbor$_u[v_2]$.label $\leftarrow \texttt{label}_2$
  $v' \leftarrow$ decodeNexthop($\texttt{label}_2, \mathcal{D}_u[\text{lmin}_u]$.label)
  foreach $x \in \mathcal{I}_u$ *such that* $\mathcal{I}_u[x]$.closestLandmark $\neq$ null **do**
    SEND$_{v'}$(mLogicalDiffusion($v_2, \texttt{label}_2, x, \mathcal{I}_u[v]$.closestLandmark, $\mathcal{I}_u[v]$.closestLabel, u))
  end
end

Algorithm 8: Réception par un nœud u d'un message mLogicalContact(manager, target, landmark, targetLabel) provenant d'un nœud w

if $u \neq$ manager **then**
  $v' \leftarrow \mathcal{D}_u$[manager].nexthop
  SEND$_{v'}$(mLogicalContact(manager, target, landmark, targetLabel))
else
  $\mathcal{I}_u \leftarrow \mathcal{I}_u \cup \{\texttt{target}\}$
  $\mathcal{I}_u$[target].closestLandmark $\leftarrow$ landmark
  $\mathcal{I}_u$[target].closestLabel $\leftarrow$ targetLabel
  Update $\mathcal{I}_u$[target].bestLandmark and $\mathcal{I}_u$[target].bestLabel if needed
  foreach $v \in$ LogNeighbor$_u$ **do**
    $v' \leftarrow$ decodeNexthop(LogNeighbor$_u[v]$.label, $\mathcal{D}_u[\text{lmin}_u]$.label)
    SEND$_{v'}$(mLogicalDiffusion(v, LogNeighbor$_u[v]$.label, target, landmark, targetLabel, u))
  end
end

Algorithm 9: Réception par un nœud u d'un message mLogicalDiffusion(head, headLabel, target, landmark, targetLabel, tail) provenant d'un nœud w

if $u \neq$ head **then**
  $v' \leftarrow$ decodeNexthop(headLabel, $\mathcal{D}_u[\text{lmin}_u]$.label)
  SEND$_{v'}$(mLogicalDiffusion(head, headLabel, target, landmark, targetLabel, tail))
else
  $\mathcal{I}_u \leftarrow \mathcal{I}_u \cup \{\texttt{target}\}$
  $\mathcal{I}_u$[target].closestLandmark $\leftarrow$ landmark
  $\mathcal{I}_u$[target].closestLabel $\leftarrow$ targetLabel
  Update $\mathcal{I}_u$[target].bestLandmark and $\mathcal{I}_u$[target].bestLabel if needed
  foreach $v \in$ LogNeighbor$_u \setminus \{\texttt{tail}\}$ **do**
    $v' \leftarrow$ decodeNexthop(LogNeighbor$_u[v]$.label, $\mathcal{D}_u[\text{lmin}_u]$.label)
    SEND$_{v'}$(mLogicalDiffusion(v, LogNeighbor$_u[v]$.label, target, landmark, targetLabel, u))
  end
end

Définition 3.12. *Le temps $t_{3.1}$ est la fin de la phase 3.1.*

Il est assez rapide de remarquer que seuls les nœuds de couleur i participent à la construction de $\mathcal{T}_i$, en effet, un nœud u ajoute une entrée dans sa table $\mathrm{LogNeighbor}_u$ seulement si $c(u) = i$ (voir lignes 2 and 6 de l'algorithme 6). On peut ainsi définir simplement :

Définition 3.13. *Le graphe logique pour une couleur $i \in [1, k]$ est induit par la table* $\mathrm{LogNeighbor}$ *des nœuds de couleur i.*

Définition 3.14. *Pour une couleur donnée i, l'ensemble $V_i^{\varnothing}$ est constitué des nœuds appartenant à V_i et n'ayant pas d'ancêtre de couleur i dans l'arbre $T_{l_{\min}}$*

Remarquons que cet ensemble correspond également à l'ensemble des racines potentielles de l'arbre $\mathcal{T}_i$, néanmoins seul l'exécution pourra déterminer lequel de ces nœud sera considéré comme racine.

Lemme 3.20. *Le graphe logique induit par les structures les tables* $\mathrm{LogNeighbor}$ *pour une couleur donnée est effectivement un arbre de plus court chemin qui couvre le graphe.*

Preuve. Sans perte de généralité, considérons que le nœud $r \in V_i^{\varnothing}$ est le nœud dont le message $\mathrm{mLogicalParent}(r, \mathcal{D}_r[l_{\min}].\mathrm{label}, c(r))$ est arrivé en premier au nœud $l_{\min}$. Dans le cas particulier où i est la couleur 1 (la couleur des nœuds landmark), alors les algorithmes 18 et 19 fixent respectivement au lignes 13 et 11 $r = l_{\min}$. Donc, $\mathrm{LogRoot}_{l_{\min}}[i].\mathrm{node} = r$, lorsque tout message suivant est reçu par le nœud $l_{\min}$. Remarquons également que le nœud r n'aura aucun parent logique attaché (lignes 16-17 de l'algorithme 6). Tout nœud $v \in V_i \setminus \{r\}$ va avoir exactement un parent attaché à lui, soit r si $v \in V_i^{\varnothing}$, (lignes 19-22 de de l'algorithme 6) soit son premier ancêtre de couleur i dans $\mathrm{InvTree}_{l_{\min}}$ (lignes 6-9 de l'algorithme 6) Par conséquent, tout nœud excepté le nœud r, a, à la fin de cette phase, exactement un parent logique. De plus, tout les nœuds partagent un même ancêtre commun $l_{\min}$. Ainsi, le graphe induit par les parents logique est un arbre. De plus, l'algorithme 7 ajoute simplement un lien symétrique à tous les lien de parenté crées, le graphe logique induit par les arêtes logiques est donc un arbre. ∎

Lemme 3.21. *Au temps $t_{3.1}$, pour tout nœuds u et v, si $c(u) = \mathrm{h}(v)$, alors $\mathcal{I}_u[v].\mathrm{closestLandmark}$ et $\mathcal{I}_u[v].\mathrm{closestLabel}$ ont des valeurs correctes, i.e. le nœud $\mathcal{I}_u[v].\mathrm{closestLandmark} \in \mathrm{B}(v)$ et $\mathcal{I}_u[v].\mathrm{closestLabel}$ est un chemin compressé de v à $\mathcal{I}_u[v].\mathrm{closestLandmark}$.*

Preuve. Comme l'algorithme 8 est appelé par tout nœud v avant la fin de la phase 2 (donc avant le commencement de la phase 3.1), le représentant w du nœud v pour la couleur $\mathrm{h}(v)$ apprend $\mathrm{closestLandmark}_v$ et $\mathcal{D}_v[\mathrm{closestLandmark}_v].\mathrm{label}$ via un message $\mathrm{mLogicalContact}$ (ligne 11 de l'algorithme 20). De plus, comme $\mathrm{closestLandmark}_v$ est affecté après le temps t_1, après lequel $\mathcal{D}_v$ est correct, le nœud $\mathrm{closestLandmark}_v$ appartient à $\mathrm{B}(v)$ (ligne 2 de l'algorithme 20). En outre, d'après le lemme 3.17 tout nœud v a dans $\mathcal{D}_v[\mathrm{closestLandmark}_v].\mathrm{label}$ un chemin compressé de v à $\mathrm{closestLandmark}_v$ dans $\mathrm{InvTree}_{\mathrm{closestLandmark}_v}$. Le nœud w (représentant de v pour $\mathrm{h}(v)$) envoie cette information dans $\mathcal{T}_{\mathrm{h}(v)}$ (algorithme 9). D'après le lemme 3.20, $\mathcal{T}_{\mathrm{h}(v)}$ est un arbre qui couvre l'ensemble des nœuds $V_{\mathrm{h}(v)}$, donc tout nœud $u \in V_{\mathrm{h}(v)}$ reçoit $\mathrm{closestLandmark}_v$ et $\mathcal{D}_v[\mathrm{closestLandmark}_v].\mathrm{label}$ dans un message $\mathrm{mLogicalDiffusion}$. Le nœud u aura donc uniquement des entrées correctes qui viendront s'ajouter à sa table $\mathcal{I}_u[v]$ (lignes 5-7 de l'algorithme 9). ∎

Lemme 3.22. *La complexité en temps de la phase* 3.1 *est de* $\mathsf{Time}(\textsc{Phase}3.1) = 4D$.

Preuve. La distance entre tout nœud et le landmark $l_{\min}$ est borné supérieurement par D comme $\mathrm{InvTree}_{l_{\min}}$ est un arbre de plus court chemin. Donc les algorithmes 6 et 7 prennent un temps D au maximum.
La diffusion de l'étiquette de routage d'un nœud v est faite en deux parties (algorithmes 8 et 9) :

1. Premièrement, le nœud v envoie son étiquette de routage à son représentant u pour la couleur $\mathrm{h}(v)$, cela est fait en parallèle de la construction de $\mathcal{T}_{\mathrm{h}(v)}$. Cet algorithme a une complexité en temps de D, en effet $u \in \mathrm{B}(v)$, donc les messages sont envoyés par des plus court chemin en utilisant les tables de routage-direct.

2. Deuxièmement, cette étiquette de routage est diffusée dans $\mathcal{T}_{\mathrm{h}(v)}$, cette seconde partie commence uniquement lorsque le nœud u est ajouté à $\mathcal{T}_{\mathrm{h}(v)}$ et donc, dans le pire cas est effectuée en séquence avec des trois autres algorithmes. La complexité en temps de cette deuxième partie est proportionnelle au diamètre de l'arbre $\mathcal{T}_{\mathrm{h}(v)}$ qui est borné supérieurement par le diamètre de $\mathrm{InvTree}_{l_{\min}}$, et requiert donc $2D$ unités de temps.

La complexité totale de ces deux parties est donc de $4D$ ∎

Le lemme suivant sera utilisé pour prouver les bornes sur l'étirement maximum des routes dans la section 3.10.1 :

Lemme 3.23. *Pour tout nœud* $u \in V$ *et tout nœud destination* v *avec une valeur hachée égale à* $c(u)$*, le label* $\mathcal{I}_u[v].\mathrm{bestLabel}$ *encode un chemin qui est au moins aussi court que le chemin encodé par* $\mathcal{I}_u[v].\mathrm{closestLabel}$.

Preuve. La variable $\mathcal{I}_u[v].\mathrm{bestLabel}$ est modifiée si une meilleure option est proposée au nœud u. Comme cette variable est originalement fixée à $\mathcal{I}_u[v].\mathrm{closestLabel}$, donc $\mathcal{I}_u[v].\mathrm{bestLabel}$ encode un chemin qui est au moins aussi court que celui encodé par l'étiquette $\mathcal{I}_u[v].\mathrm{closestLabel}$. ∎

3.7.2 Sous hypothèse de coloration aléatoire, phase 3.1

Le lemme suivant n'est valable que lorsque les couleurs des nœuds sont choisies de manière aléatoire uniforme.

Lemme 3.24 (Longueur d'un arc logique)**.** *Notons* X_u *le nombre d'arêtes entre* u *et son premier ancêtre de couleur* $c(u)$ *dans* $\mathrm{InvTree}_{l_{\min}}$*. L'espérance de* X_u *est inférieure à* $2 \cdot \min\{D, k\}$.

Preuve. L'arbre $\mathrm{InvTree}_{l_{\min}}$ a une profondeur entre $D/2$ et D. Si $u \in V_i \setminus V_i^{\varnothing}$ alors le nombre d'arêtes menant à son parent logique est inférieur à la profondeur de $\mathrm{InvTree}_{l_{\min}}$. Si $u \in V_c^{\varnothing}$, $X_u \leqslant 2D$. Prennons maintenant $k < 2D$ et considérons la variable aléatoire Z dont la distribution est une loi géométrique de paramètre $p = 1/k$. Son espérance est k.

Supposons temporairement que le nœud $l_{\min}$ est multi-coloré. Notons $Y_u(i)$ le nombre d'arêtes entre u (de couleur arbitraire) et son plus proche ancêtre de couleur

i dans $\mathrm{InvTree}_{l_{\min}}$. Pour i et u donné, $Y_u(c) = \min\{Z, p(u)\}$ avec $p(u)$ la profondeur de de u dans $\mathrm{InvTree}_{l_{\min}}$, notons que :

$$\mathbb{E}\left(Y_u(c)\right) = \min\{\mathbb{E}\left(Z\right), p(u)\} \leqslant \min\{k, D\}$$

Par définition $X_u = Y_u(c)$ si $u \in V_c \setminus V_c^{\varnothing}$ et $X_u = Y_u(c) + Y_{\mathrm{logicalRoot}_c}(c)$ si $u \in V_c^{\varnothing} \setminus \{\mathrm{logicalRoot}_c\}$. Il s'ensuit que $\mathbb{E}\left(X_u\right) \leqslant 2\min\{k, D\}$.

∎

La complexité de communication induit par la consruction des arbres logiques est proportionel à la longueur des chemins empruntés par les messages de construction. Il s'ensuit que :

Lemme 3.25 (Coût de communication des arbres logiques)**.** *Le coût de communication total requis pour la construction de tous les arbres $\mathcal{T}_i$ est de* $\mathsf{Msg_s}(\textsc{ArbresLogiques}) = \mathcal{O}(n\min\{D, k\})$ *messages.*

Preuve. Pendant la construction d'un arbre $\mathcal{T}_i$, tout message échangé d'un nœud u vers son parent logique v utilise seulement des arêtes de $\mathrm{InvTree}_{l_{\min}}$. Plus précisément, si $\mathrm{LogRoot}_{l_{\min}}[i].\mathrm{node} = v$ alors :

- le nœud v apprend l'arête logique menant à u par le chemin $u \rightsquigarrow l_{\min} \rightsquigarrow v$ car $l_{\min}$ stocke l'étiquette de routage de v dans $T_{l_{\min}}$ et u connaît un plus court chemin vers $l_{\min}$;
- et le nœud u apprend l'arête logique menant à v par le chemin $v \rightsquigarrow l_{\min} \rightsquigarrow u$, en effet, le nœud v connait un plus court chemin vers $l_{\min}$ et le nœud u transporte son étiquette de routage dans $\mathrm{InvTree}_{l_{\min}}$ jusqu'au nœud $l_{\min}$, ce dernier peut donc lui renvoyer un message sans avoir à stocker cette étiquette.

Si v est un ancêtre de u dans $\mathrm{InvTree}_{l_{\min}}$ alors, les nœuds u et v échangent leurs étiquettes de routage par le chemin $u \rightsquigarrow v$ dans $\mathrm{InvTree}_{l_{\min}}$. Comme indiqué dans le lemme 3.24 tout ces chemins ont une espérance de longueur de $\mathcal{O}(\min\{k, D\})$. Tout arbre logique $\mathcal{T}_i$ contient en moyenne $\frac{n}{k}$ nœuds. L'espérance du coût de communication de la construction de l'ensemble des arbres logique est de $\mathcal{O}(n \cdot \min\{k, D\})$. ∎

Lemme 3.26 (Coût de communication de la construction des tables de routage-indirect)**.** *Le nombre total de messages utilisés pour échanger l'ensemble des étiquettes de routage pour construire $\mathcal{I}_u$ pour tout nœud $u \in V$ est de :*

$$\mathsf{Msg_s}(\textsc{Diffusion}) = \begin{cases} \mathcal{O}\left(\frac{n^2}{k} \cdot \min\{D, k\}\right) & \textit{en moyenne} \\ \mathcal{O}\left(\frac{n^2}{k} \cdot \min\{D, k\log k\}\right) & \textit{avec grande probabilité} \end{cases}$$

Preuve. Pour une destination donnée v de couleur i, le nœud v envoie son étiquette de routage dans l'arbre de son plus proche landmark à son représentant u pour la couleur $h(v)$, puis u diffuse cette étiquette dans $\mathcal{T}_i$:

1. La hop-distance entre v et u est, d'après le lemme 3.11, d'au plus $k\log k$ avec grande probabilité. De plus, le rayon de $\mathrm{B}(v)$ ne peut pas être plus grand que D. Le premier message, dans $\mathrm{B}(v)$, traverse au plus $\min\{D, k\log k\}$ arêtes avec grande probabilité. Ce coût est dominé par le coût de diffusion du l'item suivant.

2. D'après le lemme 3.24, tout message de diffusion entre deux voisins logique de $\mathcal{T}_i$ traverse au plus $2 \cdot \min\{D, k\}$ arêtes en espérance. De plus, d'après le lemme 3.11 la distance entre deux voisins logiques peut également être bornée avec grande probabilité par $\min\{D, k \log k\}$. En outre, d'après le lemme 3.12, le nombre de nœuds composant un arbre logique est de $\Theta(n/k)$ avec grande probabilité. D'après ces observation nous pouvons déduire que, pour tout nœud v, le coût de communication requis pour construire l'ensemble des entrées $\{\mathcal{I}_u[v] \mid u \in V \setminus \{v\}\}$ est :

   a) en espérance de $\mathcal{O}(n/k \cdot \min\{D, k\})$;

   b) avec grande probabilité de $\mathcal{O}(n/k \cdot \min\{D, k \log k\})$.

En sommant sur l'ensemble des nœuds $v \in V$ destination, l'espérance du coût de communication est de $\mathcal{O}\left(\frac{n^2}{k} \cdot \min\{D, k\}\right)$ en moyenne et de $\mathcal{O}\left(\frac{n^2}{k} \cdot \min\{D, k \log k\}\right)$ avec grande probabilité. ∎

D'après les lemmes 3.25 et 3.26 nous dérivons la borne générale sur le coût de communication de cette phase :

Corollaire 3.3. *Le coût de communication de la phase 3.1 est de :*

$$\mathsf{Msg_s}(\textsc{Phase}3.1) = \begin{cases} \mathcal{O}\left(\frac{n^2}{k} \cdot \min\{D, k\}\right) & \textit{en moyenne} \\ \mathcal{O}\left(\frac{n^2}{k} \cdot \min\{D, k \log k\}\right) & \textit{avec grande probabilité} \end{cases}$$

À la fin de cette phase le routage est possible entre toute paire de nœuds, cependant l'étirement n'est pas borné par 5. Aucune preuve n'est explicitement donnée, mais à cet instant l'étirement est tout de même borné par 7, la preuve du lemme 3.29 donne les indications qui permettraient de prouver cette borne. La phase suivante permet d'obtenir un étirement borné par 5.

3.8 Phase 3.2: Amélioration de l'étirement par échanges locaux d'étiquettes de routage (amélioration des entrées de la table $\mathcal{I}_u$)

L'objectif de cette phase est d'améliorer les entrées des tables indirectes (notamment les étiquettes de routage) dans le but de passer d'un étirement maximum 7 des routes à un étirement 5. Le cas pathologique dans lequel l'étirement peut atteindre 7 est décrit dans la figure 3.7. La chose importante à remarquer sur ce cas de figure est que $\mathtt{s} \in \mathrm{B}(\mathtt{t})$ et $\mathtt{t} \notin \mathrm{B}(\mathtt{s})$ ce qui peut impliquer un étirement 7 lors du routage de `s` à `t` via le landmark $l_{\mathtt{t}}$, plus proche landmark de `t`. Pour atteindre l'étirement objectif, il est nécessaire de récupérer pendant cette dernière phase, pour tout nœud s tel que $s \in \mathrm{B}(t)$ et $t \notin \mathrm{B}(s)$ l'étiquette de routage $\mathcal{D}_t[l_s].\mathrm{label}$ du nœud t dans l'arbre de landmark $\mathrm{InvTree}_{l_s}$. Pour faire cela, tout nœud t va demander à tout nœud s de $\mathrm{B}(t)$ s'il se trouve dans le cas pathologique décrit ci-dessus (voir également l'algorithme 10), puis le nœud s s'il se trouve dans ce cas, va répondre à cette demande en renvoyant à t l'identifiant de son plus proche landmark $\mathrm{closestLandmark}_s = l_s$ (algorithme 11). Le nœud t va ensuite envoyer un nouveau message à s, contenant son étiquette de routage $\mathcal{D}_t[l_s].\mathrm{label}$ dans l'arbre de landmark $\mathrm{InvTree}_{l_s}$ (algorithme 12), enfin, le nœud s fera suivre cette étiquette de routage à son représentant pour la couleur $h(t)$, noté w. Le nœud w stockera alors l'étiquette de routage reçu si elle produit un chemin plus court que celle qu'il stocke déjà pour t (w connaît au moins l'étiquette calculée pour t durant la phase 3.1).

Le lemme 3.29 montre que dans ce cas de figure, après convergence de la phase 3.2 le routage de `s` à `t` via le représentant `manager` produit un étirement inférieur à 5.

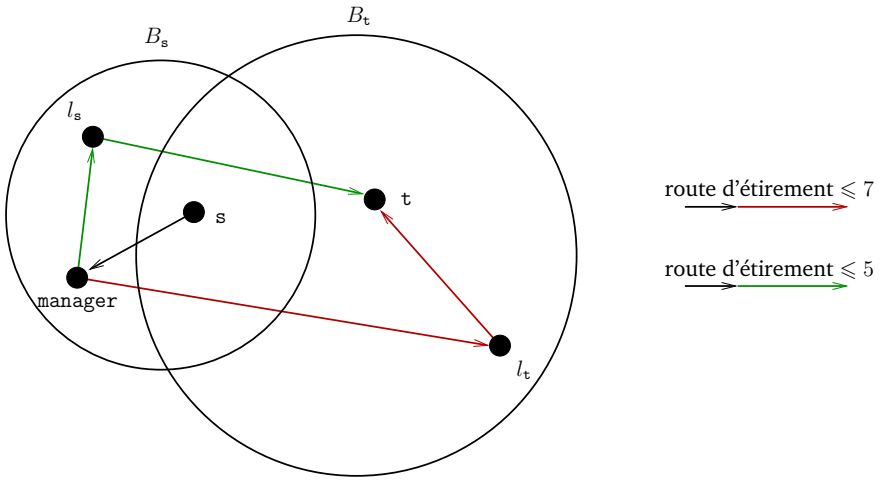

Figure 3.7: Cas dans lequel l'étirement $\leqslant 7$ et la route alternative permettant d'obtenir un étirement de $\leqslant 5$.

3.8.1 Algorithmes de la phase 3.2

Algorithm 10: Réception par un nœud u d'un message $\mathrm{mLocalBroadcast}(\mathtt{s}, \mathtt{t}, \mathtt{label})$ provenant d'un nœud w

```
if u ≠ s then
    v' ← D_u[t].nexthop
    SEND_{v'}(mLocalBroadcast(t, s, label))
else
    if t ∉ D_u then
        v' ← decodeNexthop(label, D_u[t].label)
        SEND_{v'}(mLocalAnswer(s, label, u, closestLandmark_u))
    end
end
```

Remarque : L'algorithme 10 ne peut pas être fait par une simple diffusion car la frontière d'une boule de voisinage $\mathrm{B}(u)$ ne peut être déterminée que par le nœud u lui même. C'est pourquoi, le nœud $\mathtt{t}$ doit envoyer des messages individuels aux différents nœuds de $\mathrm{B}_{\mathtt{t}}$.

Algorithm 11: Réception par un nœud u d'un message $\mathrm{mLocalAnswer}(\mathtt{t}, \mathtt{label}, \mathtt{s}, \mathtt{manager})$ provenant d'un nœud w

```
if u ≠ t then
    v' ← decodeNexthop(label, D_u[t].label)
    SEND_{v'}(mLocalAnswer(s, label, t, manager))
else
    v' ← D_u[t].nexthop
    SEND_{v'}(mLocalUpdate(s, u, D_u[manager].label))
end
```

Algorithm 12: Réception par un nœud u d'un message $\mathrm{mLocalUpdate}(\mathtt{s}, \mathtt{t}, \mathtt{label})$ provenant d'un nœud w

1 **if** $u \neq \mathtt{s}$ **then**
2 | $v' \leftarrow \mathcal{D}_u[\mathtt{s}].\mathrm{nexthop}$
3 | $\mathrm{SEND}_{v'}(\mathrm{mLocalUpdate}(\mathtt{s}, \mathtt{t}, \mathtt{label}))$
4 **else**
5 | $\mathtt{manager} \leftarrow \mathcal{M}_u[\mathrm{h}(\mathtt{t})].\mathrm{node}$
6 | $v' \leftarrow \mathcal{D}_u[\mathtt{manager}].\mathrm{nexthop}$
7 | $\mathrm{SEND}_{v'}(\mathrm{mLocalRelay}(\mathtt{manager}, \mathtt{s}, \mathrm{closestLandmark}_u, \mathtt{label}))$
8 **end**

Algorithm 13: Réception par un nœud u d'un message $\mathrm{mLocalRelay}(\mathtt{manager}, \mathtt{t}, l_{\mathtt{s}}, \mathtt{label})$ provenant d'un nœud w

1 **if** $u \neq \mathtt{manager}$ **then**
2 | $v' \leftarrow \mathcal{D}_u[\mathtt{manager}].\mathrm{nexthop}$
3 | $\mathrm{SEND}_{v'}(\mathrm{mLocalRelay}(\mathtt{manager}, \mathtt{t}, l_{\mathtt{s}}, \mathtt{label}))$
4 **else**
5 | Update $\mathcal{I}_u[\mathtt{t}].\mathrm{bestLandmark}$ and $\mathcal{I}_u[\mathtt{t}].\mathrm{bestLabel}$ if better to reach $\mathtt{t}$ via $l_{\mathtt{s}}$
6 **end**

3.8.2 Analyses de la phase 3.2

D'après les lemmes 3.8 et 3.17 durant cette phase les états des tables de routage-direct ne dépendent pas du temps, car celles-ci restent inchangées après le temps t_2 et que la phase courante s'effectue après le temps t_2. De plus, toute table $\mathcal{D}_u$ est correcte et stocke, pour tout nœud qu'il représente (tout nœud v tel que $h(v) = c(u)$), une étiquette de routage dans $\mathrm{InvTree}_{l_v}$, avec l_v plus proche landmark de v. (lemmes 3.21, 3.17 et théorème 3.17)

Définition 3.15. *Le temps $t_{3.2}$ est le temps auquel la phase 3.2 se termine, i.e. lorsque tous les nœuds $t \in V$ ont envoyé* mLocalBroadcast *à tous les nœuds $u \in B_t$ et plus aucun message de type* mLocalBroadcast*,* mLocalAnswer*,* mLocalUpdate *ou* mLocalRelay *n'est reçu.*

Le lemme suivant ne sera utilisé que plus tard, pour prouver que l'étirement de l'algorithme après convergence est de 5 (voir lemme 3.29) :

Lemme 3.27. *Au temps $t_{3.2}$, pour toute destination $t \in V$, tout nœud $s \in \mathrm{B}(t)$ tel que $t \notin \mathrm{B}(s)$ a un représentant $w \in B(s)$ qui stocke une étiquette de routage qui produit une route au moins aussi courte que celle produite par $\mathcal{D}_t[l_s].label$.*

Preuve. Pour une destination donnée $t \in V$, supposons qu'il existe un nœud $s \in \mathrm{B}(t)$ tel que le nœud w, représentant de s pour la couleur $h(t)$, stocke un label ℓ qui produit une route plus longue que celle qui serait produit s'il stockait $\mathcal{D}_t[l_s].label$. Cela peut arriver si et seulement si :

- le test fait à la ligne 5 de l'algorithm 10 est évalué à False ce qui est impossible car $s \in \mathrm{B}(t)$;
- ou, $\mathcal{D}_t[l_s].label$ n'a jamais reçu w car une étiquette de routage n'est remplacée que par une étiquette produisant une route plus courte (algorithm 13, ligne 5).

Cela implique donc que l'un des quatre messages n'a pas pu être routé vers sa destination, ce qui est impossible car ces quatre algorithmes n'utilisent que les tables de routage-direct et les étiquettes de routage dans l'arbre $\mathrm{InvTree}_t$. ∎

3.8.3 Sous hypothèse de coloration aléatoire, phase 2

Le lemme suivant n'est valable que si les couleurs des nœuds on été tirées de manière aléatoire uniforme dans $[1, k]$.

Lemme 3.28. *La phase* 3.2 *a une complexité en temps de* $4 \cdot \min\{D, k \log k\}$ *et un coût de communication de* $\mathcal{O}(nk \log k \cdot \min\{D, k \log k\})$ *avec grande probabilité.*

Preuve. Comme tous les messages de cette phase sont envoyés dans des boules de voisinage (soit celle de `s` soit celle de `t`), ils traversent au plus $\min\{D, k \log k\}$ arêtes. Les quatre algorithmes s'exécutent séquentiellement. La complexité en temps de la phase 3.2 est donc de $4 \min\{D, k \log k\}$.

Pour tout nœud $t \in V$ un message est envoyé à tout nœud s appartenant à $\mathrm{B}(t)$ par un plus court chemin dans la boule de $\mathrm{B}(t)$, puis renvoyé en t via un plus court chemin dans l'arbre $\mathrm{InvTree}_t$, puis à nouveau vers s dans $\mathrm{B}(t)$ et enfin de s vers son nœud représentant pour la couleur $h(t)$ dans $\mathrm{B}(s)$. Le nœud t envoie avec grande probabilité moins de $k \log k$ messages de ce type (un par nœud de sa boule). Dans le pire cas, tous les nœuds appartenant à $\mathrm{B}(t)$ ne contiennent pas t dans leurs boules et demandent donc tous l'étiquette de t dans leur plus proche arbre de landmark. Ainsi, tout chemin dans une boule a une longueur d'au plus $\min\{D, k \log k\}$, donc le coût de communication total de la phase 3.2 est de $\mathcal{O}(n \cdot (k \log k) \cdot \min\{D, k \log k\})$ avec grande probabilité.

∎

3.9 Algorithme de routage détaillé

Cette section décrit plus en détail l'algorithme de routage déjà décrit dans la section 3.4.1. Un message de contenu `content` devant être envoyé à un nœud `dest` est tout d'abord formaté comme suit : $\langle$`dest`, null, null, `content`$\rangle$ avant d'être traités par

l'algorithme suivant :

Algorithm 14: Algorithme de routage pour un nœud $u \neq \texttt{dest}$ et une requête $\langle \texttt{dest}, v, \texttt{label}, \texttt{content} \rangle$

```
1  if dest ∈ D_u then                          /* Routage dans la boule de voisinage */
2  |  v' ← D_u[dest].nexthop
3  |  SEND_{v'}(⟨dest, v, label, content⟩)
4  else if label ≠ null then                   /* Routage par un arbre en utilisant une
   étiquette */
5  |  v' ← decodeNexthop(label, D_u[v].label)
6  |  SEND_{v'}(⟨dest, v, label, content⟩)
7  else
8  |  i ← h(dest)
9  |  if c(u) ≠ i then                                  /* Router vers le manager */
10 |  |  x ← M_u[i].node
11 |  |  v' ← D_u[x].nexthop
12 |  |  SEND_{v'}(⟨dest, v, label, content⟩)
13 |  else      /* Routage indirect depuis le manager vers le nœud dest */
   |  |  /* Met à jour le représentant                                    */
14 |  |  x ← I_u[dest].bestLandmark
15 |  |  xlabel ← I_u[dest].bestLabel
   |  |  /* Routage via un arbre de landmark                             */
16 |  |  v' ← decodeNexthop(xlabel, D_u[x].label)
17 |  |  SEND_{v'}(⟨dest, x, xlabel, content⟩)
18 |  end
19 end
```

3.10 Analyses globales

3.10.1 Analyse de l'étirement

Observons l'étirement de l'algorithme de routage $\mathrm{DCR}(k)$ après convergence du schéma de routage, au temps $t_{\mathrm{end}} = \max\{t_{3.1}, t_{3.2}\}$.

Lemme 3.29. *Au temps t_{end}, toute route produite par l'algorithme $\mathrm{DCR}(k)$ est d'au plus 5.*

Preuve. Considérons un nœud $s \in V$ et une destination quelconque $t \in V$. Notons w le nœud représentant du nœud s pour la couleur $h(t)$, ainsi que $d = d(s,t)$, et $\mathcal{M}_s[i].\mathrm{node} = w$. Si le nœud t appartient à $\mathcal{D}_s$ alors le routage se fait par un plus court chemin comme les tables de routage sont monotones et correctes (lemme 3.6).

Observons l'étirement de la route induite par la requête de routage de s vers t lorsque $t \notin \mathcal{D}_s$. Comme les tables de routage-direct sont correctes, nous avons $\mathrm{closestLandmark}_s \in \mathrm{B}(s)$ et $\mathrm{closestLandmark}_t \in \mathrm{B}(t)$, ces landmarks seront notés respectivement l_s et l_t.

(1) Si $s \in \mathcal{D}_t$ alors w stocke l'étiquette de routage de t dans l'arbre de landmark du nœud l_s, i.e. $\mathcal{I}_w[t].\mathrm{bestLandmark} = \mathcal{D}_t[l_s].\mathrm{label}$ (lemme 3.27). Donc, d'après l'algorithme 14 on peut remarquer que la route induite par cette requête est $s \rightsquigarrow w$ dans $\mathrm{B}(s)$, puis $w \rightsquigarrow t$ dans $\mathrm{InvTree}_{l_s}$. Or cette route est plus courte que :

$$s \rightsquigarrow w \rightsquigarrow l_s \rightsquigarrow t$$

Le routage depuis le nœud $l_s = \mathcal{I}_w[t].\mathrm{bestLandmark}$ vers t est fait en utilisant l'étiquette $\mathcal{I}_w[t].\mathrm{closestLabel}$. Comme une route produite par cette étiquette de routage est un plus court chemin dans $\mathrm{InvTree}_{l_s}$ et les routes dans $\mathcal{D}_u$ sont également des plus courts chemins, alors la longueur de la route produite pour la requête de routage de s à t est inférieure à :

$$d(s,w) + d(w,l_s) + d(l_s,t)$$

Comme, l_s et w appartient à $\mathrm{B}(s)$, si $t \notin B_s$, alors :

$$\begin{cases} d(s,l_s) \leqslant d \\ d(w,s) \leqslant d \end{cases}$$

Donc :

$$\begin{cases} d(w,l_s) \leqslant d(w,s) + d(s,l_s) \leqslant 2d \\ d(l_s,t) \leqslant d(l_s,s) + d \leqslant 2d \end{cases}$$

ce qui permet de conclure que la route produite si le nœud w stocke l'étiquette de routage $\mathcal{D}_t[l_s].\mathrm{label}$ a un étirement inférieur à 5. Si le nœud w stocke une meilleure alternative à cette étiquette l'étirement n'en est qu'amélioré.

(2) Si $s \notin \mathcal{D}_t$ alors, d'après le lemme 3.23, l'étiquette de routage stockée au nœud w est au moins aussi bonne que l'étiquette $\mathcal{I}_w[t].\mathrm{closestLandmark} = \mathcal{D}_t[l_t].\mathrm{label}$. Considérons que le nœud w n'a pas stocké de meilleure option, et observons l'étirement produit s'il stocke $\mathcal{D}_t[l_t].\mathrm{label}$. La route alors induite lors du routage de s à t est $s \rightsquigarrow w$ dans $\mathrm{B}(s)$, puis $w \rightsquigarrow t$ dans l'arbre enraciné en l_t (d'après l'algorithme 14). Cette route est au moins aussi courte que la route :

$$s \rightsquigarrow w \rightsquigarrow l_t \rightsquigarrow t$$

la route induite est donc de longueur inférieure à :

$$d(s,w) + d(w,l_t) + d(l_t,t)$$

de plus, comme $s \notin \mathcal{D}_t$ (ce qui implique que $s \notin \mathrm{B}(t)$) et par conséquent, comme nous avons également $d(t,l_t) \leqslant d$, :

$$d(w,l_t) \leqslant d(w,s) + d + d(t,l_t) \leqslant 3d$$

ce qui mène finalement à une route induite de longueur :

$$d(s,w) + d(w,l_t) + d(l_t,t) \leqslant d + 3d + d \leqslant 5d$$

Finalement, d'après le lemme 3.27 nous pouvons affirmer que l'étirement de toute route après le temps t_{end} est d'au plus 5. ∎

La suite de la section montre simplement en combinant les lemmes et théorèmes des sections 3.5, 3.6, 3.7, 3.8 et 3.10.1, que les théorèmes 3.3 et 3.4 sont vrais.

3.10.2 Preuve du théorème 3.3

Convergence et correction. La correction du schéma de routage se déduit des lemmes suivants :

- la phase 1 converge en $4D$ unités de temps (lemme 3.9);
- la phase 2 converge en $6D$ unités de temps (lemme 3.18);
- la phase 3.1 converge en $4D$ unités de temps (lemme 3.22);
- la phase 3.2 converge en $4D$ unités de temps (lemme 3.28).

Les phases 3.2 et 3.1 sont faites en parallèle, donc la complexité totale en temps de l'algorithme DCR(k) est de $14D$.

Entêtes et taille des entrées. Notre algorithme est composé de n constructions d'arbres de plus court chemin (généralement tronqués) en utilisant une variante de Bellman-Ford distribué, et d'un nombre constant de phase utilisant un des arbres précédemment construit. Par conséquent, le temps d'exécution est de $\mathcal{O}(D)$ et le coût de communication total est $\mathcal{O}(n)$ fois celui qu'un Bellman-Ford distribué utiliserait dans un même graphe, i.e. $\mathcal{O}(mn)$ messages. Enfin, tout messages contient un nombre constant de :

1. d'identifiants de nœuds : $\mathcal{O}(\log n)$ bits;
2. de hop-distances : $\mathcal{O}(\log D)$ bits;
3. de distances : $\mathcal{O}(\log n \cdot \log W)$ bits;
4. de couleur s: $\mathcal{O}(\log n)$ bits;
5. de poids de sous-arbres : $\mathcal{O}(\log n)$ bits;
6. et des étiquettes de routage : $\mathcal{O}(\min\{D, \log n\} \log n))$ bits (lemme 3.16).

Étirement des routes. L'étirement est de 5 après la fin des phases 3.1 et 3.2 comme le prouve le lemme 3.29. □

3.10.3 Preuve du théorème 3.4

Mémoire de travail. La coloration aléatoire des nœuds du graphe implique certaines propriétés sur la taille de chaque structure. Avec grande probabilité :

1. pour toute couleur i, le nombre de nœuds de couleur i est $|V_i| = \mathcal{O}\left(\frac{n}{k}\right)$; (lemme 3.12);
2. les tables de routage-direct, pour tout nœud $u \in V$ et à tout instant, ont une taille d'au plus $|\mathcal{D}_u| = \mathcal{O}(n/k + k \log k)$ (lemme 3.13).

Après la fin de la phase 1, aucune entrée n'est supprimée d'une structure, donc la mémoire de travail requise à tout moment est borné par la taille finale des structures. Directement, d'après le lemme 3.12, on peut affirmer que la mémoire de travail pour tout nœud $u \in V$ est avec grande probabilité $\mathcal{O}\left(\frac{n}{k}\right)$, avec la répartition suivante :

3. $\mathcal{O}\left(\frac{n}{k}\right)$ pour les tables de routage-indirect, une entrée est stockée par nœud de valeur hachée égale à $c(u)$;

4. $\mathcal{O}(\frac{n}{k})$ pour les voisins logiques de u dans $\mathcal{T}_{c(u)}$, comme u a au plus $|V_{c(u)}|$ voisins ;
5. notons également que la phase 3.2 n'ajoute aucune entrée, elle ne fait que modifier les étiquettes de routage existantes dans les tables de routage-indirect calculées durant la phase 3.1.

Il s'ensuit que le nombre total d'entrées pour l'algorithme DCR(k) est de $\mathcal{O}(\frac{n}{k} + k \log k)$ par nœud avec grande probabilité.

Coût de communication Observons la complexité de communication de chacune des phases :

1. le calcul des tables de routage-direct a un coût de communication de $\mathcal{O}(\xi m(k \log k + n/k))$ avec grande probabilité (lemme 3.15);
2. le calcul des étiquettes de routage a un coût de communication total de $\mathcal{O}(m(k \log k + n/k))$ (lemme 3.19) ;
3. le calcul des graphes logique et la diffusion dans ceux-ci ont un coût de communication de et $\mathcal{O}\left(\frac{n^2}{k} \cdot \min\{D, k\}\right)$ en moyenne (ou $\mathcal{O}\left(\frac{n^2}{k} \cdot \min\{D, k \log k\}\right)$ avec grande probabilité)(corollaire 3.3) ;
4. la phase permettant l'optimisation de l'étirement a un coût de communication de $\mathcal{O}(nk \log k \cdot \min\{D, k \log k\})$ avec grande probabilité (lemme 3.28).

La complexité de communication total est donc en moyenne de :

$$\mathcal{O}\left(\xi m(k \log k + n/k) + \frac{n^2}{k} \cdot \min\{D, k\}\right) \text{ messages}$$

ou, avec grande probabilité de :

$$\mathcal{O}\left(\xi m(k \log k + n/k) + \frac{n^2}{k} \cdot \min\{D, k \log k\}\right) \text{ messages.}$$

□

3.11 Optimisations possible de l'algorithme DCR.

Cette section présente quelques améliorations simples qui permettraient d'améliorer le comportement de l'algorithme de routage en pratique, en d'autres termes, ces améliorations n'influent pas sur les performances (bornes données) théoriques.

3.11.1 Réduire l'étirement dans certains cas

Dans le cas présenté en figure 3.7, lorsque $s \in \mathrm{B}(t)$ et $t \notin \mathrm{B}(s)$ il est possible de router directement de s à t, obtenant ainsi sur ce cas de figure un étirement de 3. Pour cela il est nécessaire de modifier les entrées de la table de routage-indirect, en autorisant de stocker des routes induites par la composition de deux étiquettes de routage. Le routage depuis le nœud w, représentant de s pour la couleur $h(t)$, peut se faire par la route $w \rightsquigarrow s \rightsquigarrow t$, pour cela il suffit que w stocke à la fois l'étiquette de s dans l'arbre $\mathrm{InvTree}_w$ et l'étiquette de t dans l'arbre $\mathrm{InvTree}_s$.

3.11.2 Réduire la congestion en élisant un landmark par couleur

Afin d'obtenir une meilleure congestion à la fois pour les messages de contrôle et les messages de routage, il est possible d'utiliser, dans la phase 3.1, plusieurs landmark élus au lieu d'un seul. Par exemple, il serait possible d'avoir un landmark élu par couleur. Élire plusieurs landmark n'impliquerait pas de surcoût durant la phase 1 et permettrait de réduire le nombre de messages aux alentours du nœud $l_{\min}$. La complexité de communication globale de cet algorithme serait exactement la même pour la borne avec grande probabilité.

Question Pourrait-on avoir des garanties sur la congestion ?

3.11.3 Réduire l'étirement lors des routages indirects

Lors du routage d'un nœud u vers un nœud v dans un arbre $\mathrm{InvTree}_r$ il serait possible de trouver des raccourcis, en utilisant des arêtes du graphe n'appartenant pas à l'arbre. Plus précisément, lorsqu'un nœud w appartenant au plus court chemin de u à v dans $\mathrm{InvTree}_r$, détecte que dans l'étiquette de v dans $\mathrm{InvTree}_r$ qu'il reçoit, se trouve un nœud w' appartenant à sa boule de voisinage, il peut orienter le routage vers ce nœud w'. Comme le routage dans la boule de w vers w' se fait par un plus court chemin dans le graphe, ce chemin est au moins aussi court que le chemin de w à w' dans $\mathrm{InvTree}_r$ et donc ce second chemin est un raccourcis potentiel.

3.12 Conclusion

Les questions principales qu'amène ce chapitre sont les suivantes :

Est-il possible d'atteindre un compromis optimal mémoire/étirement avec un coût de communication meilleur que tout algorithme de plus court chemin ? Il est probable que l'algorithme décrit dans [AGM+08] ne soit pas adaptable avec son étirement original de 3. En effet, ce dernier utilise une technique qui parait très difficile à adapter de manière distribué en conservant un coût de communication de $\mathcal{O}(n^2)$ et ce même pour des graphes sans-échelle. Pour arriver au même compromis que celui proposé dans [AGM+08], il faudrait sacrifier l'indépendance des noms, ce qui permettrait de considérer le schéma de Thorup et Zwick proposé dans [TZ01] qui lui, en revanche, serait simple à adapter en utilisant les techniques décrites dans ce chapitre. Cependant, sacrifier l'indépendance des noms, n'est pas une solution satisfaisante comme nous l'avons déjà vu. C'est pourquoi il est probable qu'il soit nécessaire de concevoir un algorithme nouveau pour atteindre ce compromis optimal.

Est-il possible d'adapter simplement cet algorithme pour le rendre auto-stabilisant ? Cette question est évidement la plus importante dans l'optique d'une utilisation en pratique de cet algorithme. Les algorithmes proches d'algorithmes Bellman-Ford distribués décrits dans ce chapitre ne sont pas auto-stabilisant. Néanmoins, le chapitre 5 propose une implémentation auto-stabilisante dans le modèle à mémoire partagée. Cette partie de l'algorithme étant la plus sensible cela donne bon espoir de pouvoir proposer une version auto-stabilisante de l'algorithme DCR.

Simulations de Routage distribué 4

Sommaire

Les bornes inférieures relatives au routage données dans le chapitre précédent font une totale abstraction de la famille de graphe considérée et sont données pour le modèle synchrone $\mathcal{LOCAL}$. Les questions posées dans ce chapitre sont simples :

Q1 : *Si la famille de graphe considérée est restreinte aux graphes sans-échelle, existe-t-il des algorithmes dédiés proposant de meilleurs compromis mémoire/étirement en pratique ?*

Q2 : *Quelles peuvent être les performances de ces algorithmes dédiés sur des graphes n'ayant pas les mêmes propriétés ?*

Q3 : *Les algorithmes universels tel que* DCR*, présenté dans le chapitre 3, sont-il significativement moins performants que des algorithmes dédiés aux graphes sans-échelle ?*

Q4 : *Quel est l'influence de l'asynchronisme sur le coût de communication ?*

Ce chapitre présente une étude expérimentale d'algorithmes distribués avec indépendance des noms. Tous ces algorithmes sont compactes. Certain ont une mémoire sous-linéaire garantie par nœud, d'autres ont quant-à eux une mémoire sous-linéaire en moyenne. L'objectif de cette étude est de comparer ces différents algorithmes sur les performances tels que l'étirement des routes, la mémoire et le nombre de messages nécessaire pour construire les tables de routage.

Nous commençons par donner le modèle de simulation. Puis nous détaillerons les algorithmes considérés en donnant également une analyse très brève de leurs complexités. Enfin, nous finirons par donner les résultats obtenus durant les expérimentations.

4.1 Modèle de simulation

4.1.1 Le simulateur DRMSIM

Toutes les expérimentations présentées dans ce chapitre ont été effectuées à l'aide du simulateur DRMSIM [DRM13], développé à INRIA Sofia-Antipolis.

Spécificités du modèle de communication, passage de messages. Le simulateur DRMSIM est basé sur un simulateur d'événements discrets nommé MASCSIM. DRMSIM permet de simuler le fonctionnement d'un réseau de routeurs distribués. Chaque routeur a une mémoire propre et échange des messages avec les autres routeurs via les événements discrets gérés par MASCSIM. Ce dernier s'occupe de la redistribution des messages aux routeurs concernés par les différents messages.

Spécificités du modèle de temps. Tout message émis par un nœud u à destination d'un voisin v est ajouté dans une liste. Cette liste est ordonnée par temps d'arrivée aux différents destinataires. La latence des liens est donc simulée à l'émission par le nœud u, qui calcule le temps d'arrivée du message destiné à v et l'insère dans la liste d'événements. C'est pour cette raison que nous utilisons des liens à latence fixées et ainsi respecter la propriété FIFO (*First In First Out*). En effet, la propriété FIFO est considéré comme une hypothèse standard dans le domaine du calcul distribué. Le modèle de latence considéré pour les expérimentations est le modèle Δ-borné présenté dans la section 2.3.2. La valeur de Δ sera un paramètre d'entrée du simulateur et sera spécifiée pour chacune des expériences. Remarquons également que lorsque nous parlerons de modèle synchrone ou $\mathcal{LOCAL}$, le modèle sera plus exactement le modèle 1-borné (pour les expérimentations uniquement). Comme déjà précisé en section 2.3.2, ces deux modèles ne sont pas tout à fait équivalents. Par exemple les bornes supérieures données dans le chapitre 3 ne s'appliquent pas directement dans ce modèle, mais permettent néanmoins de se faire une bonne idée des bornes dans le modèle 1-borné. Les complexités théoriques seront toujours données dans le modèle $\mathcal{LOCAL}$.

4.2 Résumé des résultats

Nous considérons dans ce chapitre que tous les graphes son non-pondérés. Les algorithmes comparés dans sont :

- **Universels :** DCR et LO;
- **Dédiés aux graphes sans-échelle :** HDLBR et CLUSTER.

Nous nous utiliserons également comme comme complexités de références celles d'un algorithme de plus court chemin de type vecteur de chemin, i.e. DVECTOR (cet algorithme étant un équivalent simplifié de BGP, nous utiliserons par moment le nom BGP pour désigner DVECTOR). Ainsi qu'un algorithme de plus court chemin optimal qui atteint les bornes inférieures du routage de plus court chemin. L'algorithme HDLBR est une version distribuée de l'algorithme introduit par Tang *et al.* dans [TZLL12]. Les algorithmes LO et CLUSTER sont originaux et présentés dans ce chapitre. Les descriptions de ces algorithmes sont précisées dans la section 4.3. Ces algorithmes sont décrits dans le modèle distribué et sont tous basés sur le squelette suivant :

Définition 4.1 (Squelette des algorithmes de routage)**.** *Tout comme dans* DCR*, la boule de voisinage d'un nœud $u \in V$ sera noté $B(u)$ et l'ensemble des landmark sera noté L. Pour router d'un nœud u vers un nœud v :*

1. *Si $v \in B(u)$ alors le routage est direct, i.e. u sait router vers v en plus court chemin via $B(u)$.*
2. *Sinon le routage se fait en utilisant un ou plusieurs arbres de landmarks.*

Dans tous les algorithmes présentés par la suite, tout nœud u devra être capable de router dans sa boule $B(u)$ et vers un sous ensemble $L_u \subseteq L$. Le plus proche landmark d'un nœud u sera appelé l_u par convention. L'ensemble des L et la notion de boules de voisinage variera légèrement selon les algorithmes. En ce qui concerne les boules de voisinage il y aura deux cas, soit la boule doit contenir exactement un landmark, soit les nœuds sont colorés et la boule contiendra au moins un nœud de chaque couleur. Le routage d'un nœud u vers un nœud $v \in B(u)$ ne sera jamais décrit car la technique utilisée est toujours la même. Le routage se fait par un plus court chemin dans $B(u)$ en utilisant le procédé décrit dans la section 3.4.1 du chapitre chapitre 3.

4.2.1 Complexités théoriques des algorithmes

Complexités théoriques. La table 4.1 présente les complexités d'étirement des routes, de communication et de mémoire de travail des différents algorithmes qui seront comparés dans la section 4.5 dans le modèle $\mathcal{LOCAL}$. L'étirement est donné sous la forme (s_m, s_a) avec s_m l'étirement multiplicatif et s_a l'étirement additif. Toutes les complexités sont données dans le cas général puis pour un graphe sans-échelle G. Notons D_L et $D_{L'}$ les distances maximales entre deux landmarks dans G ainsi que $\mathcal{B}$ et $\mathcal{B}'$ la taille maximale d'une d'une boule de voisinage respectivement dans les algorithmes HDLBR et CLUSTER. Remarquons que pour le modèle de graphe RPLG, $\mathcal{B}$ et $\mathcal{B}'$ seront prouvés être de $\mathcal{O}(\sqrt{n})$.

Cas général

Algorithme	Étirement	Mémoire de travail		Coût de communication
		Moyenne	Maximum	
DVECTOR	$(1,0)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\Theta(nm)$
DCR	$(5,0)$	$\tilde{\mathcal{O}}(\sqrt{n})$	$\tilde{\mathcal{O}}(\sqrt{n})$	$\tilde{\mathcal{O}}\left(m\sqrt{n}+n^{3/2}\cdot\min\{D,\sqrt{n}\}\right)$
LO	$(3,0)$	$\tilde{\mathcal{O}}(\sqrt{n})$	n	$\tilde{\mathcal{O}}\left(m\sqrt{n}+n^{3/2}\cdot\min\{D,\sqrt{n}\}\right)$
HDLBR	$(2,2D_L)$	$\mathcal{O}(\sqrt{n})+\mathcal{B}$	n	$\mathcal{O}(m\cdot(\sqrt{n}+\mathcal{B}))$
CLUSTER	$(3,4D_{L'})$	$1+\mathcal{B}'$	$\mathcal{O}(\sqrt{n})+\mathcal{B}'$	$\mathcal{O}(m\cdot(1+\mathcal{B}')+n\min\{D,\sqrt{n}\})$

Graphes sans échelle

Algorithme	Étirement	Mémoire de travail		Coût de communication	Référence
		Moyenne	Maximum		
DVECTOR	$(1,0)$	$\mathcal{O}(n)$	$\tilde{\mathcal{O}}(n)$	$\Theta(n^2)$	[RLH06]
DCR	$(5,0)$	$\tilde{\mathcal{O}}(\sqrt{n})$	$\tilde{\mathcal{O}}(\sqrt{n})$	$\tilde{\mathcal{O}}\left(n^{3/2}\right)$	Chapitre 3
LO	$(3,0)$	$\tilde{\mathcal{O}}(\sqrt{n})$	n	$\tilde{\mathcal{O}}\left(n^{3/2}\right)$	Ce chapitre
HDLBR	$(2,2D_L)$	$\mathcal{O}(\sqrt{n})$	n	$\tilde{\mathcal{O}}\left(n^{3/2}\right)$	[TZLL12]
CLUSTER	$(3,4D_{L'})$	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(\sqrt{n})$	$\tilde{\mathcal{O}}\left(n^{3/2}\right)$	Ce chapitre

Table 4.1: Complexités des algorithmes de routages présentés dans ce chapitre en tout généralité dans la première table puis pour des graphes sans-échelle, avec $m = \tilde{\mathcal{O}}(n)$ et $D = \mathcal{O}(\log n)$, dans la seconde. Rappelons que les graphes sont considéré non-pondérés.

4.2.2 Résumé des résultats expérimentaux.

Nous verrons dans ce chapitre que :

- Tous les algorithmes sont efficaces en moyenne en terme d'étirement.
- Cependant les algorithmes dédiés peuvent produire pour quelques routage des route d'étirement supérieurs au diamètre du graphe (pour un UDG carré par notamment).
- Les algorithmes dédiés sont vraiment très efficaces sur les graphe sans-échelle en terme de mémoire et de coût de communication. En pratique $\mathcal{B}$ et $\mathcal{B}'$ sont très inférieures à $\sqrt{n}$. La taille des boules est plutôt de l'ordre de 10 pour des graphes de 5000 nœuds pour les graphes sans-échelle.
- DCR et DCR7 ont des étirements très similaires, DCR ayant un coût de communication plus élevé il est intéressant d'utiliser DCR uniquement si l'étirement maximum est un critère important.

4.3 Les algorithmes : description et analyse théorique

Dans cette section nous présentons surtout les algorithmes de routage. Les détails de construction distribué ne seront donnés que lorsque les techniques utilisées diffèrent de celles présentées dans le chapitre 3. Par exemple, toute les constructions d'arbres

de landmarks sont similaire et ne seront pas détaillées, il en est de même pour les constructions de boules de voisinages.

De la même façon les détails des preuves concernant ces constructions ne seront pas donnés. De manière générale le résultat important est que le coût de communication pour la construction de boules de voisinage de taille X est de $\mathcal{O}(mX)$ dans le modèle $\mathcal{LOCAL}$ pour un graphe non-pondéré. Ce résultat s'obtient facilement en dérivant la preuve du lemme 3.15.

4.3.1 DCR (et DCR7)

L'algorithme DCR, est celui présenté dans le chapitre 3. Nous étudierons également l'algorithme DCR7 qui est l'algorithme DCR amputé de l'amélioration de l'étirement fait lors de la phase 3.2 (section 3.8). Voici tout de même un résumé rapide des idées principales.

Préliminaires. Tout nœud du graphe tire une couleur dans $[1, k]$ de manière aléatoire uniforme.

Ensemble des landmarks. Les landmarks sont choisis de manière aléatoire uniforme parmi V. De manière indépendante, chaque nœud se définit comme landmark avec une probabilité $1/|L|$.

Boules de voisinage. La boule d'un nœud u est définie comme étant l'ensemble minimal constitué des nœuds les plus proches de u et contenant au moins un nœud de chaque couleur.

Routage. Le routage d'un nœud u vers un nœud v se fait via un nœud w proche ayant sa couleur égale à la valeur hachée de l'identifiant de v. Puis le routage s'effectue par le landmark sélectionné par w. Le landmark sélectionné est l_v, i.e. le landmark le plus proche de v dans le cas d'DCR7. Dans le cas DCR le routage se fait en utilisant comme intermédiaire le landmark produisant la route la plus courte parmi : l_u et l_v. Ces algorithmes sont résumés en figure 4.1.

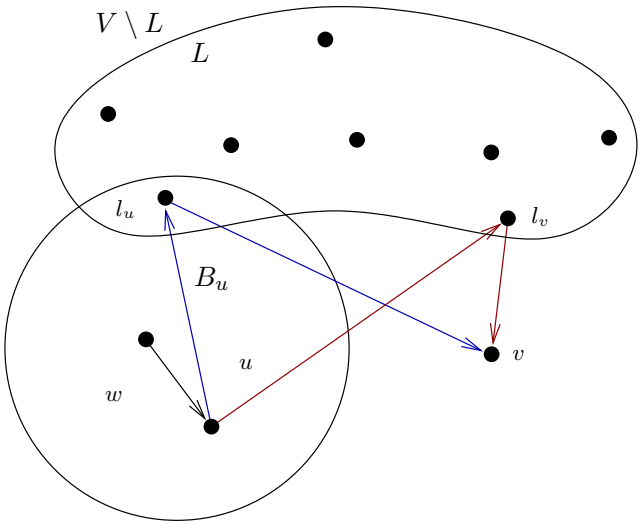

Figure 4.1: Routage du nœud u vers le nœud v. L'algorithme DCR utilise le meilleur des deux chemins (le rouge ou le bleu) tandis que DCR7 utilise uniquement le chemin bleu, via l_u.

Garanties théoriques. Les garanties détaillées pour ces algorithmes sont données par le théorème 3.3, dans le chapitre 3, le lemme suivant résume ces résultats :

Lemme 4.1. *Dans le modèle $\mathcal{LOCAL}$, les algorithmes* DCR7 *et* DCR *ont une complexité de communication de* $\mathsf{Msg_s} = \mathcal{O}\Big(mk \log k + \frac{n^2}{k} \cdot \min\{D, k \log k\}\Big)$*, une mémoire de travail de* $\mathsf{Mem} = \mathcal{O}\big(k \log k + \frac{n}{k}\big)$ *et des étirements respectifs de* 7 *et* 5.

Dans les graphes sans-échelle et pour $k = \sqrt{n}$ ce lemme peut être énoncé sous une forme plus simple

Lemme 4.2. *Dans le modèle $\mathcal{LOCAL}$, pour un graphe dans lequel $m = \tilde{\mathcal{O}}(n)$ et $D = \mathcal{O}(\log n)$, les algorithmes* DCR7$(\sqrt{n})$ *et* DCR$(\sqrt{n})$ *ont une complexité de communication de* $\mathsf{Msg_s} = \tilde{\mathcal{O}}\big(n^{3/2}\big)$*, une mémoire de travail de* $\mathsf{Mem} = \tilde{\mathcal{O}}(\sqrt{n})$ *et des étirements respectifs de* 7 *et* 5.

4.3.2 LO (Landmarks omniscients)

Cet algorithme pourrait être considéré comme une adaptation naïve de l'algorithme centralisé et étiqueté, présenté dans [TZ01], vers le modèle distribué et avec indépendance des noms. Cette deuxième considération aura pour impact d'obtenir une mémoire pour les landmarks de n entrées. Remarquons qu'en considérant le modèle étiqueté il serait possible d'implémenter le même algorithme en garantissant que tous les nœuds aient une mémoire compacte, $\mathcal{O}(\sqrt{n})$ entrées. L'algorithme dans le modèle étiqueté sera également présenté dans cette section (pour information) mais ne sera pas utilisé dans ce chapitre.

Ensemble des landmarks. Les landmarks sont répartis de manière aléatoire uniforme parmi V. De manière distribuée, chaque nœud se définit comme landmark avec une probabilité $1/|L|$.

Boules de voisinage. La boule $B(u)$ d'un nœud u est constituée du nœud l_u ainsi que de l'ensemble des nœuds dont la distance à u est strictement inférieure à $d(u, l_u)$.

Routage. Dans le modèle avec indépendance des noms, les nœuds de l'ensemble L doivent stocker une route vers tout nœud du graphe, i.e. un chemin complet pour toute destination. De plus, tout nœud u appartenant à $V \setminus L$ devra être capable de router vers le nœud le plus proche appartenant à L noté l_u. Le routage d'un nœud u à un nœud v est donc composé de deux plus courts chemins $u \rightsquigarrow l_u \rightsquigarrow v$. Le routage suivant cet algorithme est résumé en figure 4.2.

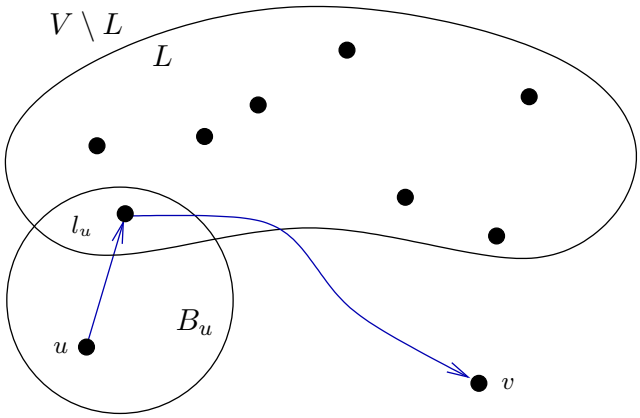

Figure 4.2: Routage du nœud u vers le nœud v avec l'algorithme LO

Garanties théoriques. Les $|L|$ nœuds landmark connaissent tous les nœuds du graphe, leur mémoire est donc de n. Les autres nœuds connaissent les nœuds de leur boule de voisinage uniquement. Or, pour tout nœud u, la boule $B(u)$ contient seulement des nœuds plus proches que l_u, le landmark de u. Les boules ont un nombre d'entrées de $\mathcal{O}(n/|L| \cdot \log n)$ (en utilisant le même argument que pour les boules d'DCR, i.e. collectionneur de coupons). Et la mémoire de travail moyenne est donc de :

$$\overline{\mathsf{Mem}} = \mathcal{O}\left(\frac{n|L| + (n - |L|)n/|L|}{n}\right) = \mathcal{O}\left(|L| + \frac{n}{|L|}\right)$$

On remarque ainsi que que $|L| = \sqrt{n}$ minimise la mémoire de travail. L'algorithme construit $|L|$ arbres de plus court chemin et n boules de voisinage contenant les $\mathcal{O}(n/|L| \cdot \log n)$ plus proches nœuds de u. En utilisant les même arguments que dans la preuve du lemme 3.15 le coût de communication, dans le modèle $\mathcal{LOCAL}$ non pondéré, peut être prouvé de :

$$\mathsf{Msg} = \mathcal{O}(m(|L| + n/|L|))$$

L'analyse de l'étirement est exactement la même que pour [TZ01], et donne donc une étirement de $(3, 0)$. Cet étirement est obtenu car les routages hors de $B(u)$ sont faits via des chemins de la forme $u \rightsquigarrow l_v \rightsquigarrow v$. Or, ce chemin est plus court que $u \rightsquigarrow l_v \rightsquigarrow u \rightsquigarrow v$ et toutes ces portions de chemin ont une longueur inférieure ou égale à $d(u, v)$, l'étirement de ce chemin est donc de 3. Le lemme suivant résume cette analyse :

Lemme 4.3. *Si l'algorithme* LO *est paramétré avec* $|L| = \sqrt{n}$ *alors, pour tout graphe et dans le modèle* $\mathcal{LOCAL}$, $\mathsf{Mem} = \mathcal{O}(n)$, $\overline{\mathsf{Mem}} = \mathcal{O}(\sqrt{n})$ *et* $\mathsf{Msg} = \tilde{\mathcal{O}}(m\sqrt{n})$. *De plus l'étirement maximum d'une route est de* $(3, 0)$.

4.3.3 HDLBR

HDLBR est l'algorithme décrit dans [TZLL12]. Il est inspiré du principe de [TZ01] et sur la technique hachage utilisée dans DCR pour rendre cet algorithme indépendant des noms. HDLBR est un schéma de routage dédié aux graphes en loi de puissance, mais a des garanties de succès de routage quelque soit le graphe considéré.

4.3.3.1 Présentation de l'algorithme

Boules de voisinage. De même que dans LO, la boule $B(u)$ d'un nœud u est constituée du nœud l_u ainsi que de l'ensemble des nœuds dont la distance à u est strictement inférieure à $d(u, l_u)$.

Boules de voisinage inverses. Dans cet algorithme, tout nœud $v \in B(u)$ doit également savoir router vers u. Cet ensemble de destinations vers lesquels le nœud v devra savoir router est appelé *boule inverse de* v et est noté $B^{-1}(v)$.

Construction distribuée des boules de voisinage inverses. Cet ensemble sera construit de manière distribuée en utilisant une détection de terminaison des boules de voisinages (de la même manière que dans la phase 1 de l'algorithme DCR, i.e. section 3.5). Pour effectuer cette détection de terminaison rappelons qu'il est d'abord nécessaire d'élire un landmark d'identifiant minimum. Lorsque les boules sont détectées terminées, tout nœud u diffusera un message contenant son identifiant ainsi que le rayon de sa boule et la distance que le message a parcouru. En utilisant ces deux distances, tout nœud v recevant ce message pourra déterminer si $v \in B_u$. Si c'est le cas, alors v ajoute u à B_v^{-1}.

Fonction de hachage H. L'algorithme HDLBR utilise une fonction de hachage $H : V \to L$, équilibrée dans le sens où, pour tout landmark $l \in L$, au plus $\mathcal{O}(\sqrt{n} \log n)$ nœuds de V ont une valeur hachée égale à l.

Routage. Le routage d'un nœud u vers un nœud v se fait en trois étapes,

(1) le nœud u route vers $H(v) \in L$;

(2) Le nœud $H(v)$ route vers le plus proche landmark de v nommé par convention l_v ;

(3) Le nœud l_v appartient à $B^{-1}(l_v)$ par définition de $B(v)$. Le paquet pourra donc être routé de l_v vers v via les boules inverses, à commencer par $B^{-1}(v)$.

Le routage suivant cet algorithme est résumé en figure 4.3 .

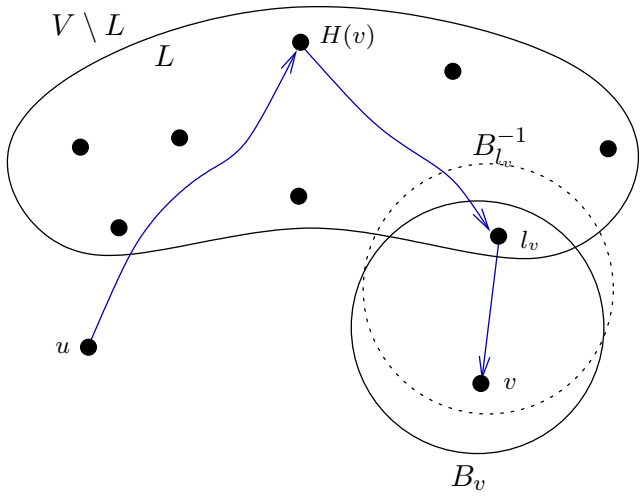

Figure 4.3: Routage du nœud u vers le nœud v avec l'algorithme HDLBR

Intuitions. Cet algorithme est particulièrement efficace en terme de mémoire de travail lorsque les boules de voisinages contiennent peu de nœuds. Or la taille de la boule de voisinage d'un nœud u est faible si les deux conditions suivantes sont réunies :

- le nœud u est proche d'un landmark, cela est lié à la couverture du graphe par les nœuds landmarks ;
- et le graphe a une expansion faible.

HDLBR est efficace en terme d'étirement si les landmarks sont proches les uns des autres, autrement dit si les nœuds de haut degrés sont très proches les uns des autres. Dans les graphes de terrain considérés, ces deux propriétés sont observées. Plus formellement, en étudiant celles-ci dans le modèle RPLG Tang *et al.* donnent des garanties théoriques sur les performances d'HDLBR pour l'étirement et la mémoire.

Rappelons que l'étirement est prouvé être de $(2, 2D_L)$ et la mémoire de $\mathcal{O}(\sqrt{n}) + \mathcal{B}$ pour cet algorithme. En utilisant les algorithmes de construction de boules et d'arbres décrits dans le chapitre 3 nous obtenons une complexité de communication de $\mathcal{O}(m|L|) + \mathcal{O}(m\mathcal{B})$ dans le modèle $\mathcal{LOCAL}$ (voir lemme 3.15 pour une idée de preuve). En fixant $|L| = \sqrt{n}$ la complexité de communication est de $\mathcal{O}(m(\sqrt{n} + \mathcal{B}))$. La section suivante présente une analyse précise pour le modèle de graphe RPLG, permettant de caractériser $\mathcal{B}$ et D_L. Ainsi, nous pourrons donner une borne ne dépendant que de n pour le coût de communication et la mémoire.

4.3.3.2 Garanties théoriques pour le modèle de graphes RPLG.

Comme le résume la table 4.1, les deux variables apparaissant dans les complexités sont :

(1) la taille des boules de voisinage $\mathcal{B}$

(2) la distance maximale entre deux nœuds landmarks D_L.

Garanties sur la valeur de $\mathcal{B}$. Sous le modèle RPLG, le volume d'un ensemble de nœud donné S est égal à la somme des poids des nœuds qui le composent

$$\mathrm{Vol}(S) = \sum_{i \in S} w_i$$

Comme le prouve Linyuan Lu dans sa thèse [Lu02], deux ensembles disjoints de volumes $\mathcal{V}$ et $\mathcal{V}'$ sont à distance au plus 1 l'un de l'autre avec forte probabilité si $\mathcal{V} \cdot \mathcal{V}' = \mathcal{O}(Vol(G))$. Le poids maximum d'un nœud dans le modèle RPLG avec la distribution considérée est : $e^{\alpha/\beta}$. Le poids des nœuds est, à une petite constante multiplicative près, égal à son degré. Donc ce nœud u_M de poids maximum appartient nécessairement à l'ensemble des landmarks. Remarquons, même si cela n'a pas d'importance pour la preuve suivante qu'il n'est pas forcément le nœud de degré maximum. Il est également important de noter que le volume d'un ensemble de S nœuds est d'au moins $|S|$ car tout les poids sont $\geqslant 1$. Dans [TZLL12] Tang *et al.* prouvent en utilisant ces propriétés que toute boule $B(u)$ contenant $\mathcal{O}\left(n^{\zeta+\epsilon}\right)$ nœuds aura, avec grande probabilité, le nœud l_u dans son voisinage si $\zeta \geqslant \beta$. En effet, si $\zeta \geqslant \beta$ le volume d'une boule multiplié par le volume du nœud de plus haut degré est

$$e^{\alpha/\beta} \cdot e^{\zeta+\epsilon} \geqslant c \cdot n$$

avec c constante et ϵ suffisamment petit. Ainsi, la taille des boules est avec grande probabilité de l'ordre de $n^{\mathcal{O}(\zeta)}$. Plus précisément, pour une valeur de $\beta = 2$ la taille des boules est $\mathcal{B} = \mathcal{O}(\sqrt{n})$ avec grande probabilité.

Garanties sur la valeur de D_L. Dans la proposition 4.1 de [CL02] il est prouvé qu'avec forte probabilité $(1 - 1/n^2)$ le diamètre D_L, du sous graphe induit par l'ensemble des $\sqrt{n}$ nœuds de plus haut degré, est constant.

Le lemme suivant résume les éléments de complexité de cette section :

Lemme 4.4. *Si l'algorithme* HDLBR *est paramétré avec $|L| = \sqrt{n}$ alors, pour tout graphe du modèle* RPLG *et pour le modèle $\mathcal{LOCAL}$,* $\mathsf{Mem} = \mathcal{O}(n)$, $\overline{\mathsf{Mem}} = \mathcal{O}(\sqrt{n})$ *et* $\mathsf{Msg} = \tilde{\mathcal{O}}(m\sqrt{n})$. *De plus l'étirement maximum d'une route est de* $(3, O(1))$.

4.3.4 CLUSTER (Landmarks connexes)

L'algorithme CLUSTER utilise les idées de HDLBR (construction de boules de voisinage et d'arbres) ainsi que la construction présentée en section 4.3.4.1 et la technique de routage compacte dans les arbres d'DCR, présentée en section 3.6.

Macroscopiquement cet algorithme va construire un cluster dans le "centre du graphe" au sein duquel tous les nœuds pourront router entre eux. Les nœuds du centre se partagent également la connaissance permettant de router vers tous les autres nœuds du graphe. À l'extérieur du centre, tout nœud est capable de router vers le centre, ainsi que vers les nœuds plus proche que le plus proche nœud du centre. Ce deuxième ensemble constitue sa boule de voisinage.

Commençons par présenter la construction du cluster, qui définit l'ensemble des landmark pour l'algorithme CLUSTER.

4.3.4.1 Attribuer des couleurs aux S plus proches nœuds d'un nœud donné.

Cette construction permet de construire une boule depuis son centre. Cela permet notamment de ne pas construire d'arbres depuis tous les nœuds du graphe comme nous avons pu le faire pour les boules de voisinage dans le chapitre 3.

Énoncé du problème. On considère un graphe $G = (V, E)$ pondéré et connexe. L'objectif est de construire de manière distribuée, dans un environnement asynchrone, une boule $C \subset V$ centrée en un sommet l. La notion de boule étant la même que celle donnée dans le chapitre 3. Les contraintes liées à la construction de C sont les suivantes :

- Le nœud l connaît C en entier et doit être capable de router vers tout nœud appartenant à C.
- Le graphe induit par C est connexe.
- L'ensemble C contient un nœud de chaque couleur.
- Toute entrée est encodée sur $\mathcal{O}(\log n)$ bits. Ce qui empêche donc l de stocker un chemin en entier pour tout nœud appartenant à C.
- $\forall u \in V \setminus C$, le nœud u a une mémoire de travail de $\mathcal{O}(1)$ entrées.
- $\forall u \in C$, le nœud u a une mémoire de travail de $\mathcal{O}\big(S + \frac{n}{S}\big)$ entrées.

Sous ces contraintes et pour $S = \mathcal{O}(\sqrt{n})$, le nombre d'entrées moyenne par nœud est de $\mathcal{O}(1)$. De manière plus générale, le nombre d'entrées total est $\mathcal{O}\big(S^2/n\big)$.

Structure de données. Tout nœud $u \in V$ stocke parent_u l'identifiant d'un nœud, destiné à être son parent dans un arbre de plus court chemin enraciné en l. De plus, si u appartient à la boule C, il stocke une table de routage C_u. Pour toute entrée $v \in C_u$, le nœud u stocke

- l'identifiant $C_u[v].\mathrm{nexthop}$ d'un nœud étant amené à être un voisin sur un plus court chemin vers v ;
- ainsi qu'une couleur de nœud $C_u[v].\mathrm{couleur}$ supposée être la couleur du nœud$v$.

De plus, tout nœud u stocke un bit d'information lui permettant de déterminer s'il appartient ou non à C.

Propriété requise pour le routage. Tout comme pour les boules de voisinage, l'ensemble des tables C doit former un ensemble monotone (voir lemme 3.8) pour garantir que le routage est possible depuis l vers tout nœud $u \in C$. La définition formelle de monotonie est la suivante :

$$\forall (u, v) \in C^2 : (v \in C_u \wedge C_u[v].\mathrm{nexthop} = w) \implies v \in C_w$$

De plus, pour garantir le routage de plus court chemin, il est nécessaire que les entrées soit correctes. Formellement, pour tout $(u, v) \in C^2$ avec $(v \in C_u \wedge C_u[v].\mathrm{nexthop} = w)$, l'entrée $C_u[v]$ est correcte si et seulement si w est un voisin de u et appartient à un plus court chemin de u à v.

Solution algorithmique. Une solution consiste à construire un arbre T_l enraciné en l dans lequel tout sommet u connaît son parent parent_u ainsi que sa distance à l notée d_u. Puis, le sommet l sélectionne un sous ensemble des nœuds couvert par cet arbre comme faisant partie de C. L'algorithme distribué peut se résumer comme suit :

1. Tout nœud u tel que $d_u \leqslant S$ fait remonter dans l'arbre T_l son identifiant et sa distance. Puis il envoie un message de synchronisation vers l. Ainsi, lorsque la racine a reçu un message de synchronisation de tout ses voisins elle sait qu'elle a reçu toutes les distances des nœuds pouvant potentiellement faire partie de C.

   Le nœud l conserve à tout moment la table C concernant les $x \leqslant S$ nœuds les plus proches de lui. En cas d'égalité de distance entre deux nœuds u et v, le nœud u est préféré au nœud v si $ID(u) < ID(v)$.

2. Dès que le nœud l a détecté la terminaison de la phase 1 il initie une diffusion qui permettra aux nœuds choisis par l de savoir qu'ils appartiennent à C. Le message M émis par l contient le rayon R de C ainsi que l'identifiant ID_{seuil} du nœud $\mathrm{seuil} \in C$ tel que $d_{\mathrm{seuil}} = R$ et $\forall w \in C | (d_w = R), (id(w) \geqslant id(\mathrm{seuil}))$.

   Remarque : Parmi les nœuds à distance R de l, les nœuds appartenant à C sont ceux d'identifiant minimaux. Donc, pour tout nœud $w' \in V \mid d_{w'} = R$ on a $id(w) < id(\mathrm{seuil})$.

   Considérons un nœud u recevant le message $M = (R, ID_{\mathrm{seuil}})$ d'un voisin w. Si le prédicat suivant est vérifié :

   $$P_{\mathrm{boule}}(u) \equiv [(d_u < R) \vee ((d_u = R) \wedge (id(u) \geqslant ID_{\mathrm{seuil}}))] \wedge u \notin C$$

   le nœud u se marque comme appartenant à C, envoie un accusé $A = (id(u))$ de cet ajout à parent_u et retransmet le message M à tout ses voisins.

3. Dès qu'un nœud w reçoit un un accusé $A = (id(u))$ l'avertissant de l'ajout de u à C d'un nœud u par un voisin v il ajoute dans sa table C_w une entrée $C_w[u]$ avec $C_w[u].\mathrm{nexthop} = v$ (pour le moment la couleur du nœud u est indéterminée). Puis il transmet le message A à son parent_w.

   Le nœud l est capable de détecter la terminaison de cette étape, en effet, lorsque tous les accusés d'ajouts à C sont parvenus à l on observe que $|C_l| = S$.

4. Dès que le nœud l apprend la route vers un nœud v il lui attribue une couleur arbitraire c parmi celle que l n'a pas encore attribué, c est choisie dans

   $$[1, S] \setminus \{ \bigcup_{w \in C_l} C_l[w].\mathrm{couleur} \}$$

   de plus, le nœud l met à jour l'entrée de v : $C_l[v].\mathrm{couleur} = c$. De plus, il route vers v pour le prévenir de la couleur qui lui est attribuée. Tout nœud w appartenant au chemin de routage l vers v met à jour son entrée couleur : $C_w[v].\mathrm{couleur} = c$.

Théorème 4.1 (Complexité de communication)**.** *Le coût de communication total de l'algorithme ci-dessus en nombre d'entrées de $\mathcal{O}(\log n)$ bits est de $\mathcal{O}(m + n \cdot \min\{D, S\})$. De plus, les messages ont une taille de $\mathcal{O}(\log n)$.*

Preuve. Voici le détail des complexités de chacune des étapes de cet algorithme :

0. La construction de l'arbre enraciné en L, comme déjà vu plusieurs fois à un coût de communication de Msg $= \mathcal{O}(m)$ messages de taille $\mathcal{O}(\log n)$.

1. La remontée dans l'arbre T_l d'identifiants sur une distance d'au maximum $\min\{D, S\}$ pour tous les nœuds, soit $n \cdot \min\{D, S\}$ messages.

2. Diffusion depuis l dans le graphe G jusqu'à une distance R d'un message contenant R et un l'identifiant d'un nœud. Soit un coût de communication de $\mathcal{O}(m)$.

3. Les accusés de réception peuvent être envoyés par des plus courts chemins vers l en utilisant les parents des nœuds dans T_l. Le coût de cette étape est donc celui de S routages vers un nœud à distance $\min\{D, S\}$, soit $\mathcal{O}(S \cdot \min\{D, S\})$.

4. Les plages de couleurs sont définies par deux entiers de $\mathcal{O}(\log S) = \mathcal{O}(\log n)$ bits. De plus les S messages sont routés dans C, or le rayon de C est de $\min\{D, S\}$, donc le coût de communication de cette phase est de $\mathcal{O}(S)$.

Enfin, les messages ne contiennent que des distances dans le graphe ainsi que des identifiants de nœuds et des couleurs, la tailles des messages est donc de $\mathcal{O}(\log n)$.∎

Théorème 4.2 (Complexité mémoire)**.** *Le nombre d'entrées dans la table d'un nœud est $\mathcal{O}(1 + S)$ s'il appartient à C et 1 s'il n'appartient pas à C. Le nombre moyen d'entrées par nœuds de $\mathcal{O}\left(1 + \frac{S^2}{n}\right)$. De plus la taille des entrées est de $\mathcal{O}(\log n)$.*

Preuve. Tout nœud $u \in V$ doit stocker un compteur allant jusqu'à $\mathrm{degre}(u)$ pour permettre la détection de terminaison de l'arbre T_l (accusés de réceptions). Tout nœud u appartenant à C sait, de plus, router vers un sous-ensemble de C, la taille de sa table de routage est donc de $\mathcal{O}(1 + S)$. À aucun moment un nœud de l'ensemble C ne stocke d'information concernant un nœud n'appartenant pas à sa table, donc sa mémoire de travail est également de $\mathcal{O}(1 + S)$. Si u n'appartient pas à C sa mémoire de travail est donc 1 entrée. En sommant sur tous les nœuds on obtient donc une mémoire totale de $\mathcal{O}\big(S^2 + (n - S)\big)$ soit :

$$\mathcal{O}\left(\frac{S^2 + (n - S)}{n}\right) = \mathcal{O}\left(1 + \frac{S^2}{n}\right) \text{ en moyenne.}$$

Enfin, chaque entrée a une taille de $\mathcal{O}(\log n)$ car elle est constituée d'un compteur allant jusqu'à n au maximum ainsi que d'un identifiant de nœud.

∎

Théorème 4.3 (Complexité en temps)**.** *Le temps de convergence de cet algorithme est de $\mathcal{O}(D)$ quelque soit le paramètre S.*

Preuve. Chacun des sous-algorithmes est soit une diffusion, soit un routage de plus court chemin chacun d'entre eux prend donc un temps D. Le temps total est donc $\mathcal{O}(D)$. ∎

4.3.4.2 Présentation de l'algorithme CLUSTER

Ensemble des landmark. L'ensemble des landmarks L est centré autour du nœud de plus haut degré de G nommé l. Cet ensemble L est construit en utilisant l'algorithme décrit en sous-section 4.3.4.1 avec une taille d'ensemble laissée en paramètre de CLUSTER et initié par l. L'ensemble L est donc connexe et contient les $|L|$ nœud les plus proches de l. Le nœud l peut alors router vers tout nœud $v \in L$. Il est de plus la racine d'un arbre de plus court chemin dans lequel tout nœud connaît v son étiquette de routage $\ell(v, T_l)$.

Boules de voisinage. De même que dans LO et HDLBR, la boule $B(u)$ d'un nœud u est constituée du nœud l_u ainsi que de l'ensemble des nœuds dont la distance à u est strictement inférieure à $d(u, l_u)$.

Routage. Tout nœud $u \in V$ sait router vers l. Le routage depuis un nœud $u \in V$ vers un nœud $v \in V$ est effectué ainsi :

- $u \rightsquigarrow l_u$: u contient un nœud l_u dans $B(u)$ tel que $l_u \in L \wedge \forall l_x \in L, d(u, l_u) \leqslant d(u, l_x)$.
- $l_u \rightsquigarrow l \rightsquigarrow H(v)$: cette partie du routage se fait en deux temps, premièrement en remontant dans l'arbre T_l depuis le nœud l_u vers le landmark l'_u premier ancêtre commun à l_u et $H(v)$ dans T_l. Le nœud l'_u étant un ancêtre de $H(v)$, il connaît une route vers ce dernier et peut donc router en plus court chemin vers lui. Remarquons qu'au pire, le premier ancêtre commun à l_u et $H(v)$ dans T_l est le landmark l lui même.
- $H(v) \rightsquigarrow l \rightsquigarrow v$: Le nœud $H(v)$ connaît l'étiquette de v dans T_l il est donc capable de router vers v dans T_l. Ce chemin, tout comme dans l'étape précédente, ne passera pas nécessairement par l mais par le premier ancêtre commun à $H(v)$ et v dans T_l.

L'étirement dépend donc, tout comme dans HDLBR, de la distance entre u et v ET du rayon du sous-graphe G_L induit par L. La figure 4.4 montre le routage utilisant le schéma CLUSTER, dans celle-ci les traits en pointillés représentent les arêtes du sous-graphe de G_L.

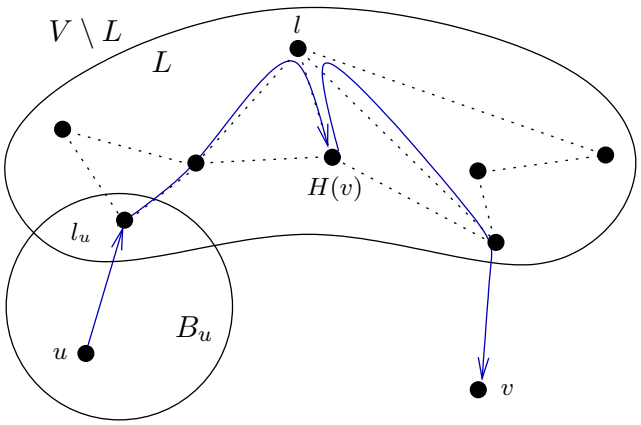

Figure 4.4: Routage du nœud u vers le nœud v avec l'algorithme CLUSTER

Intuitions - Efficacité sur les graphes RPLG. Notons L' l'ensemble des landmarks calculés par CLUSTER sur un graphe G. Ainsi que L l'ensemble des landamarks calculés par HDLBR sur ce même graphe. L'idée derrière CLUSTER est la même que pour HDLBR. Cependant, les résultat expérimentaux donnés dans [TZLL12] montrent que dans les graphes CAIDA et RPLG la quasi-totalité des entrées des tables de routages sont des entrées permettant de router vers les nœuds de L, autrement dit $\mathcal{B} << |L|$. Dans CLUSTER l'ensemble des landmark est intuitivement très similaire à celui de HDLBR, en effet, les nœuds centraux dans le graphe ont un fort degré et sont très fortement connectés, donc $\mathcal{B}'$ est très similaire à $\mathcal{B}$ et $\mathcal{B}' << |L'|$. De plus dans CLUSTER le sous-graphe induit par les nœuds L' est connexe, ce qui permet à tout nœud $u \notin L'$ de ne pas se soucier du routage dans L' et ainsi déléguer tout routage hors de $B(u)$ au nœud l_u, plus proche landmark de u. Tout nœud $u \notin L'$ économise ainsi $|L'|$ entrées en mémoire et sa mémoire de travail ne dépend que de $\mathcal{B}'$.

Analyse de la complexité mémoire. Les nœuds doivent construire l'ensemble centré en l, le landmark de plus haut degré en utilisant la technique décrite dans la section 4.3.4.1. Cette construction requiert une mémoire de $1 + |L|$ entrées pour les nœuds de L et 1 entrée pour les autres (théorème 4.2). De plus, tout nœud sait router vers l'ensemble des nœuds de sa boule, soit au maximum $\mathcal{B}'$ nœuds. Enfin les landmarks doivent, en plus, savoir router vers tous les nœuds de L. La mémoire totale des nœuds est donc de :

- $1 + \mathcal{B}'$ pour les nœuds n'appartenant pas à L ;
- et $1 + \mathcal{B}' + |L|$ pour les nœuds appartenant à L.

En moyenne, la mémoire des nœuds est donc de $1 + \mathcal{B}' + \frac{|L|^2}{n}$.

Analyse du coût de communication. De cette complexité mémoire peut directement se calculer le coût de communication, toujours en utilisant les même arguments que pour la preuve du lemme 3.15. Le coût de construction des boules est de $\mathcal{O}(m \cdot \mathcal{B}')$ le coût de construction du groupe de landmarks est de $\mathcal{O}(m + n \cdot \min\{D, |L|\})$ (voir théorème 4.1). Le coût de communication total est donc de $\mathcal{O}(m(1 + \mathcal{B}') + n \cdot \min\{D, |L|\})$.

Analyse de l'étirement. L'étirement dépend de la distance maximale entre deux nœuds landmarks, que nous notons $D_{L'}$. Pour router d'un nœud $u \in V$ vers un nœud $v \in V \setminus (B(u) \cup L)$, la route est de la forme :

$$u \rightsquigarrow l_u \rightsquigarrow H(v) \rightsquigarrow v$$

Or ce chemin est plus court que le chemin :

$$u \rightsquigarrow l_u \rightsquigarrow l \rightsquigarrow H(v) \rightsquigarrow l \rightsquigarrow l_u \rightsquigarrow u \rightsquigarrow v$$

Enfin, la longueur de ces différentes portions de routes sont bornées respectivement par :

$$u \underset{d(u,v)}{\rightsquigarrow} l_u \underset{D_{L'}}{\rightsquigarrow} l \underset{D_{L'}}{\rightsquigarrow} H(v) \underset{D_{L'}}{\rightsquigarrow} l \underset{D_{L'}}{\rightsquigarrow} l_u \underset{d(u,v)}{\rightsquigarrow} u \underset{d(u,v)}{\rightsquigarrow} v$$

La longueur totale de cette route est donc de $3d(u,v) + 4D_{L'}$ soit un étirement de $(3, 4D_{L'})$.

Le lemme suivant résume les complexités de cet algorithme :

Lemme 4.5. *L'algorithme* CLUSTER *dans le modèle $\mathcal{LOCAL}$, a les complexités suivantes* $\mathsf{Mem} = 1 + \mathcal{B}' + |L|$, $\overline{\mathsf{Mem}} = 1 + \mathcal{B}' + \frac{|L|^2}{n}$ *et* $\mathsf{Msg} = \mathcal{O}(m(1 + \mathcal{B}') + n \cdot \min\{D, |L|\})$. *De plus l'étirement maximum d'une route est de* $(3, 4D_{L'})$.

4.3.4.3 Garanties théoriques dans le modèle RPLG, analogies avec HDLBR

Dans cette section nous considérons le paramétrage $|L| = \sqrt{n}$.

Valeur de $\mathcal{B}'$ pour le modèle RPLG. Si l'ensemble L a une taille de $\sqrt{n}$, alors en utilisant exactement les même arguments que pour HDLBR la taille des boules de voisinages peut être bornée par $\mathcal{O}(\sqrt{n})$. Ainsi la mémoire de travail requise par un nœud est avec forte probabilité $\mathsf{Mem} = \overline{\mathsf{Mem}} = \mathcal{O}(\sqrt{n})$ au maximum et en moyenne.

Valeur de $D_{L'}$ pour le modèle RPLG. Rappelons que L' est l'ensemble des landmarks calculés par CLUSTER et L l'ensemble des landamarks calculés par HDLBR sur un même graphe. Supposons que ces deux algorithmes soient paramétrés de la même manière, i.e. $|L| = |L'|$. Tous les nœuds de L' sont à distance au plus $\mathrm{rayon}(L')$ du nœud l central. Le diamètre du sous-graphe induit pas L' est donc $\mathrm{rayon}(L') * 2$. Or le nœud l est le nœud de plus haut degré du graphe, il fait donc également parti de L. De plus, $|L| = |L'|$ donc

$$D_L < \mathrm{rayon}(L') \implies \exists v \in L : v \notin L' \wedge d(v, l) < \mathrm{rayon}(L')$$

cependant, L' contient tous les plus proches nœuds de l il y a donc une contradiction, car par définition de L' :

$$v \notin L' \implies d(v, l) \geqslant \mathrm{rayon}(L')$$

Donc $D_L \geqslant \mathrm{rayon}(L')$, c'est pourquoi $D_{L'} \leqslant 2D_L$, la configuration permettant d'avoir l'ensemble L minimal en terme de rayon est exposé en figure 4.5. Or le rayon de l'ensemble des landmark L est prouvé constant avec forte probabilité dans les graphes RPLG donc, avec cette même configuration, $D_{L'}$ est constant. L'étirement de ce schéma de routage est donc de $(3, \mathcal{O}(1))$ dans les graphes RPLG.

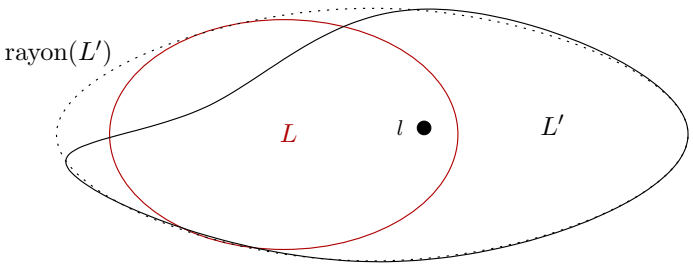

Figure 4.5: Plus faible diamètre obtenable pour L relativement à L'.

Compromis sur la taille des entrées. Dans HDLBR la taille des entrées est $\mathcal{O}(\log n)$, chaque entrée est composée d'identifiants de taille logarithmique. Dans CLUSTER les entrées stockées pour router hors de L' par les nœuds appartenant à L' sont des étiquettes de routage et ont donc une taille $\mathcal{O}\big(\log^2 n\big)$ (lemme 3.16).

Le lemme suivant résume les éléments de complexité de cette section, rappelons que le graphe a un diamètre poly-logarithmique et que $m = \tilde{\mathcal{O}}(n)$:

Lemme 4.6. *Si l'algorithme* CLUSTER *est paramétré avec* $|L| = \sqrt{n}$ *alors, avec grande probabilité, pour tout graphe* RPLG *et dans le modèle* $\mathcal{LOCAL}$, $\mathsf{Mem} = \overline{\mathsf{Mem}} = \mathcal{O}(\sqrt{n})$ *et* $\mathsf{Msg_s} = \mathcal{O}\big(n^{3/2}\big)$. *De plus l'étirement maximum d'une route est de* $(3, \mathcal{O}(1))$.

4.4 Les graphes et les paramétrages d'algorithmes utilisés

Nous allons pour les expérimentations considérer différents modèles de graphes petit monde : GLP, RPLG, CAIDA et DIMES. Ainsi que deux modèles de graphes aléatoires de diamètre plus important et avec des distributions des degrés suivant une loi de poisson : UDG carré et UDG rectangulaire. Nous considérons ces modèle car ils sont susceptibles d'être des cas dans lesquels les algorithmes spécialisés aux graphes sans-échelle (HDLBR ou CLUSTER) sont peu performants. Les détails des modèles de graphe sont décrits dans la section 2.1.3. La table 4.2 montre un résumé des propriétés des graphes généré. Remarquons enfin que pour les graphes générés fixerons $n = 5000$ car pour les UDG, au delà de cette taille, le nombre d'arêtes rend les simulation longues voire impossibles (RAM requise trop importante). Dans l'idéal il serait intéressant de pouvoir effectuer les simulations sur des graphes ayant de l'ordre de 17000 nœuds.

	Modèle de graphe					
	CAIDA	DIMES	RPLG	GLP	UDG	UDG rect.
#Nœuds n	17306	17144	5000	5000	5000	5000
#Arêtes m	35547	46621	11558	12771	23756	44719
Diamètre D	10	8	12	8	75	271

Table 4.2: Propriétés des graphes utilisés pour les expérimentations.

La distribution des degrés et des distances sont données pour les différents graphes respectivement en figures 4.6 et 4.7. Comme les graphes considérés ont des tailles différentes, toutes les métriques sont données en pourcentage des nœuds (ou des paires de nœuds).

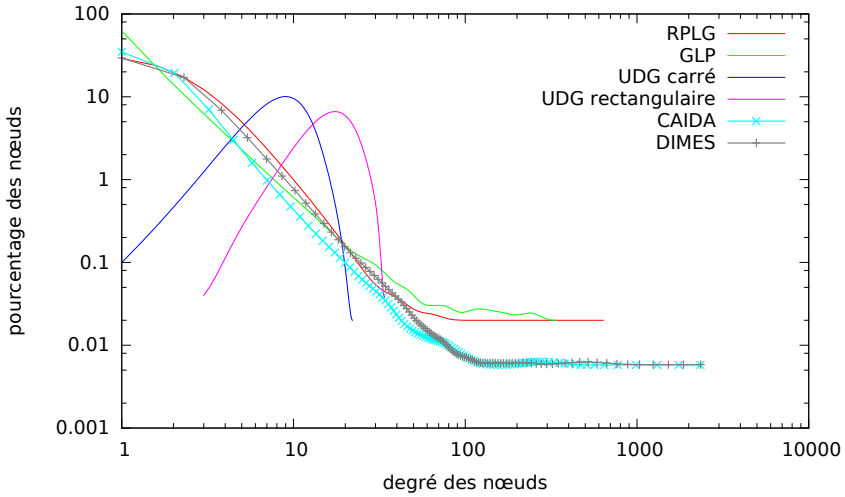

Figure 4.6: degrés

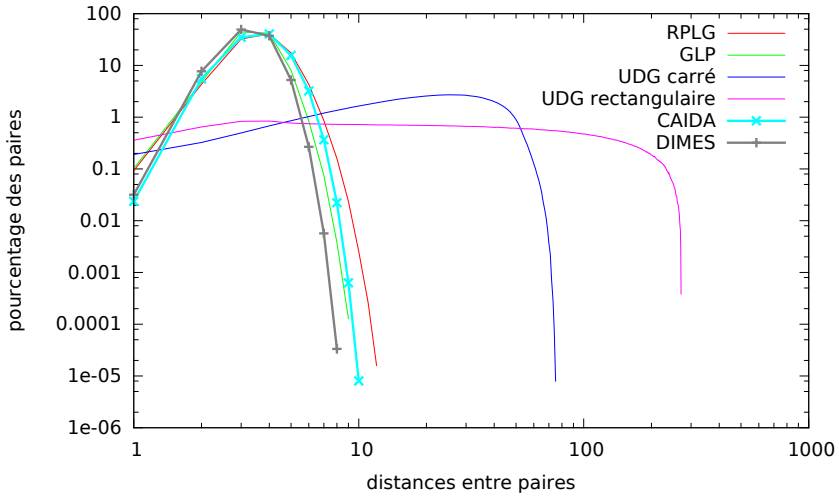

Figure 4.7: distances

La suite de cette section présente les paramétrages utilisés pour chacun de ces modèles.

Paramétrages. Les significations précises des paramètres ont été données dans la section 2.1.3 :

- **Graphe GLP** Le générateur de graphe GLP est implémenté dans *dipergraph*, une librairie utilisée par DRMSIM. Dans nos simulation, le paramétrage utilisé est toujours le suivant* :
  - nombre de nœuds initiaux $n_0 = 6$;

*Ce paramétrage est utilisé au sein du projet que nous intégrons pour obtenir des résultats comparables, de plus ce paramétrage permet d'approcher au mieux le graphe des AS.

  - nombre d'arêtes par pas $x = 1.15$;
  - importance donnée à l'attachement préférentiel, i.e. plus γ est grand, plus les nœuds de fort degré seront préférés pour la création d'arêtes. $\gamma = 0.4669$;
  - et enfin $p = 0.6753$.

  Le graphe généré a les propriétés structurelles suivantes : $n = 5000$, $m = 12771$ et $D = 8$.

- **Graphe RPLG** La valeur du paramètre de loi de puissance β sera fixée à 2.3 dans les expériences, valeur pour laquelle la distribution des degrés approche la distribution du graphe des AS. Le graphe généré a les propriétés structurelles suivantes : $n = 5000$, $m = 11558$, $D = 12$.

- **Graphes UDG** Le paramétrage de rayon utilisé est celui qui permet d'obtenir un graphe de degré moyen minimal tout en étant connexe. Comme nous l'avons déjà fait remarquer cette valeur est connue théoriquement :

$$r = c \cdot \sqrt{\frac{\ln n}{n}}$$

  avec c une constante supérieure à 1. La valeur de c a été déterminée empiriquement (en augmentant la valeur de r jusqu'à ce que le graphe généré soit connexe), pour information, la valeur de rayon utilisée sera de 0.041 pour les deux modèles suivants. Nous utiliserons pour les expérimentations un UDG classique (plan carré unitaire) ainsi qu'une variante sur un plan rectangulaire $[0, 1] \times [0, 1/\sqrt{n}]$:

  - **Carré.** Le graphe généré a les propriétés structurelles suivantes $n = 5000$, $m = 23756$ et $D = 75$.
  - **Rectangulaire.** L'objectif de ce modèle est d'obtenir un graphe avec un diamètre plus important tout en gardant l'aspect aléatoire du modèle UDG. Le graphe généré a les propriétés structurelles suivantes $n = 5000$, $m = 44719$ et $D = 271$.

- Nous utiliserons également deux cartes issues de données réelles[†] :

  - **Carte CAIDA [Cai]** La carte utilisée est celle obtenue en juin 2004 et a les caractéristiques suivantes : $n = 17306$, $m = 35547$, $D = 10$.
  - **Carte DIMES [Dim]** La carte utilisée est celle obtenue en 2007 et a les caractéristiques suivantes : $n = 17144$, $m = 46621$, $D = 8$.

Enfin, nous considérerons dans toutes les expériences, que le nombre de couleurs et le nombre de landmarks utilisés pour les différent paramétrages d'algorithmes sont de $k = |L| = \sqrt{n}$. Nous rappelons également que les graphes sont considérés non-pondérés.

[†]Les liens permettant d'obtenir ces cartes sont fournis dans la bibliographie

4.5 Résultats expérimentaux

La première section donne un premier regard sur les résultats, du point de vue des performances moyennes. Les sections suivantes proposent une analyse plus détaillée des résultats, en observant notamment les distributions pour les différentes mesures de performances et l'impact de l'asynchronisme sur le coût de communication.

4.5.1 Observation générales

Cette section présente un aperçu des résultats expérimentaux. Elle montre les moyennes obtenues pour les différentes combinaisons d'algorithme et de modèle de graphe pour l'étirement, la mémoire et le coût de communication.

Étirement. D'un point de vue macroscopique nous pouvons voir en figure 4.8 que l'étirement de tous ces algorithmes est toujours bon (toujours inférieur à 2). Cela signifie que le nombre de paquets que les nœuds retransmettraient dans un contexte d'utilisation réel serait au maximum deux fois plus élevé que pour un algorithme de plus court chemin tel que DVECTOR.

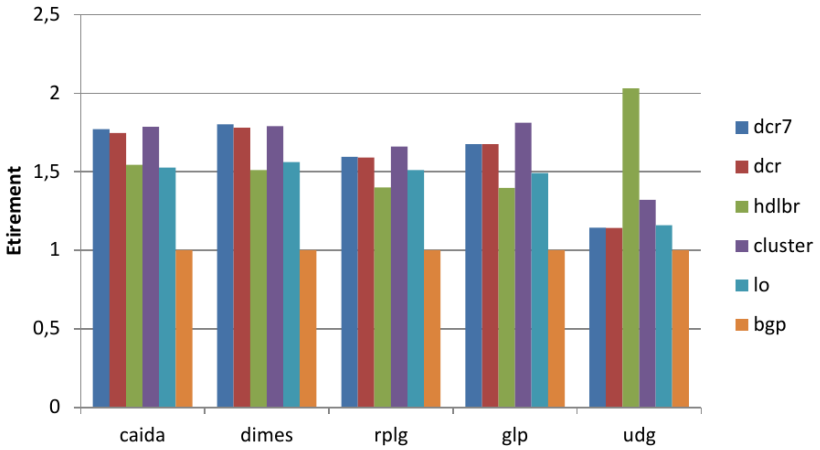

Figure 4.8: Étirements multiplicatif de tous les algorithmes sur tous les graphes.

Il est finalement assez surprenant que des algorithmes dédiés aux graphes sans-échelle, tels que HDLBR ou CLUSTER, aient de bons étirement sur les UDG. En effet, ces algorithmes sont basés sur le fait que les nœuds de fort degré sont centraux dans le graphe ce qui n'est pas nécessairement le cas dans un UDG (au delà des bords du graphe qui ont un degré plus faible, les autre nœuds ont un degré qui ne dépend pas de leurs positions dans le plan et donc pas de leur centralités). Cependant, les détails donnés en section 4.5.2 concernant ces deux algorithmes, montrent que l'étirement des routes est très variable et que certaines routes ont des étirement pouvant aller jusqu'à 100. Notons que l'étirement est dans ce dernier cas plus grand que le diamètre du graphe. Il est possible que l'étirement soit plus grand que le diamètre car plusieurs indirections sont effectuées durant le routage.

Mémoire. Nous pouvons constater en figure 4.9 que l'algorithme CLUSTER est significativement moins bon que les autres sur les UDGs, le nombre d'entrées peut

atteindre plus de 60% des nœuds du graphe. Tous les autres algorithmes obtiennent un nombre d'entré de 10%, autrement dit un facteur d'amélioration de 10 par rapport à tout algorithme de plus court chemin. En revanche, cet algorithme utilise en moyenne très peu d'entrées pour tous les graphes sans-échelle considérés : respectivement 8.22, 5 et 4.9 entrées par nœud dans les graphes CAIDA, RPLG et DIMES. La section 4.5.3 montre, plus en détail que l'algorithme CLUSTER est strictement meilleur que tous les autres sur les graphes sans-échelle et strictement moins bon sur les UDG.

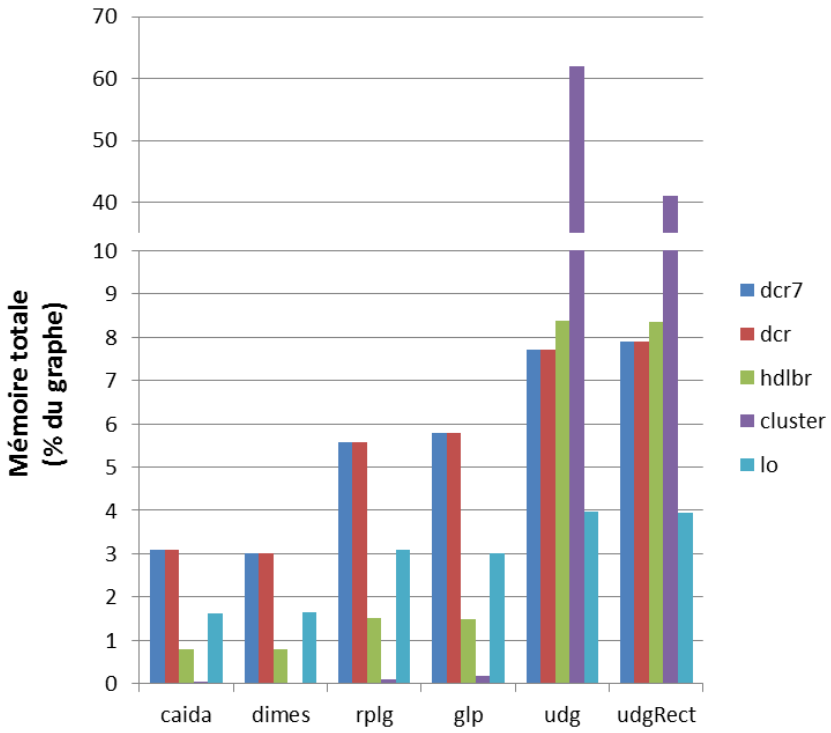

Figure 4.9: Taille des tables de routage en pourcentage des destinations stockées. Remarquons que DVECTOR stocke dans sa table de routage 100% des destinations quel que soit le graphe considéré.

Coût de communication. Comme le montre la figure 4.10 on pouvait s'y attendre, le coût de communication est très lié à la complexité mémoire des algorithmes. La tendance générale pour les coût de communication est la même que pour la mémoire. Voici tout de même le détail des pourcentage par rapport au meilleur algorithme de plus court chemin théorique. Nous considérons que le meilleur algorithme de plus court chemin a un coût de communication d'exactement n^2 :

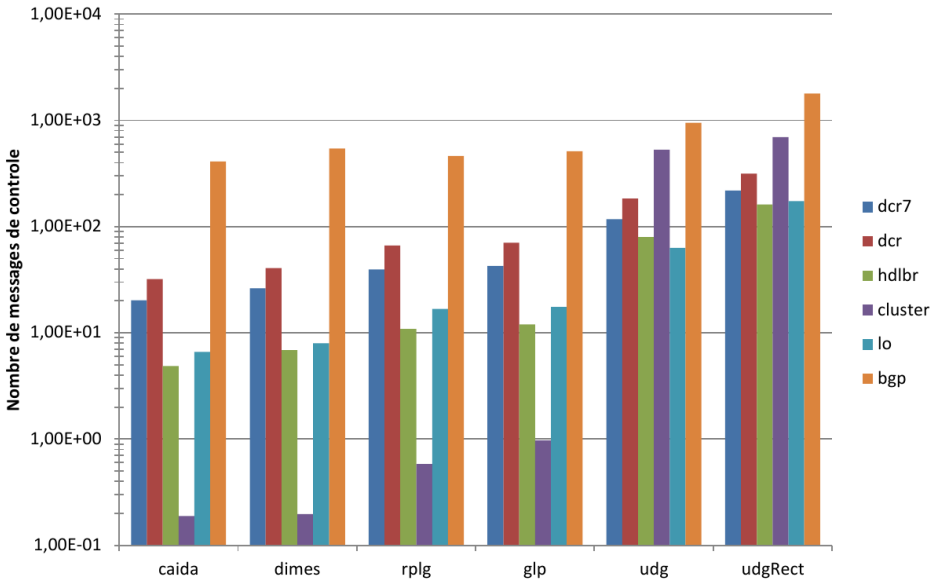

Figure 4.10: Résumé des coûts de communication.

Nous pouvons remarquer que pour tout graphe, les algorithmes compacts ont un coût de communication inférieur à celui de DVECTOR. Cependant, pour les graphes du modèle UDG, un algorithme de plus court chemin pourrait avoir un coût de communication plus faible. En effet, les coût de communication observées dépassent n^2 que nous avons montré dans le chapitre 3 être une borne inférieure pour cette mesure de complexité.

Plus de précisions. Les sections suivantes entrent plus en détail dans les différentes mesures de performance. En effet, nous nous intéressons non seulement aux performances d'un point de vue macroscopique mais également aux garanties individuelles des nœuds. Il n'est par exemple pas souhaitable qu'un algorithme de routage rende un nœud difficilement accessible (grand étirement des routes pour ce nœud) ou surcharge la mémoire d'un nœud (grande table de routage). Par la suite nous aborderons également l'influence de l'asynchronisme sur les coûts de communication ainsi que la charge des nœuds en messages de contrôle et en paquets de routage.

La conclusions sur les résultats détaillés est présentée en section 4.6.1.

4.5.2 Étirement

L'étirement moyen de tous les algorithmes comme nous l'avons vu est très homogène. Cependant, en observant les résultats de plus près, en figures 4.11 et 4.12, pour le graphe de type UDG carré, nous pouvons voir que certaines routes ont en fait un étirement très important (jusqu'à 100 pour HDLBR et 20 pour CLUSTER), cela impliquerait en pratique que certains nœuds aient une très forte latence, ce qui est à priori un défaut rédhibitoire. Remarquons tout de même qu'une très grande majorité des routages pour HDLBR a un étirement proche de la moyenne :

- les quartiles 1 et 3 sont respectivement de 1 et 2.3 ;
- nous pouvons de plus observer en figure 4.12 qu'au moins de 1% des routes ont un étirement supérieur à 8.

En ce qui concerne CLUSTER l'étirement de 20 n'apparaît que pour une seule route (rappelons que les routes ne sont pas symétrique ce qui explique qu'il puisse y avoir une unique route d'étirement maximum). De plus, moins de 1% des routes ont un étirement supérieur à 3. L'étirement de ce second algorithme n'est donc pas si catastrophique, bien qu'aucune garantie théorique ne puisse être donnée pour ce modèle de graphes.

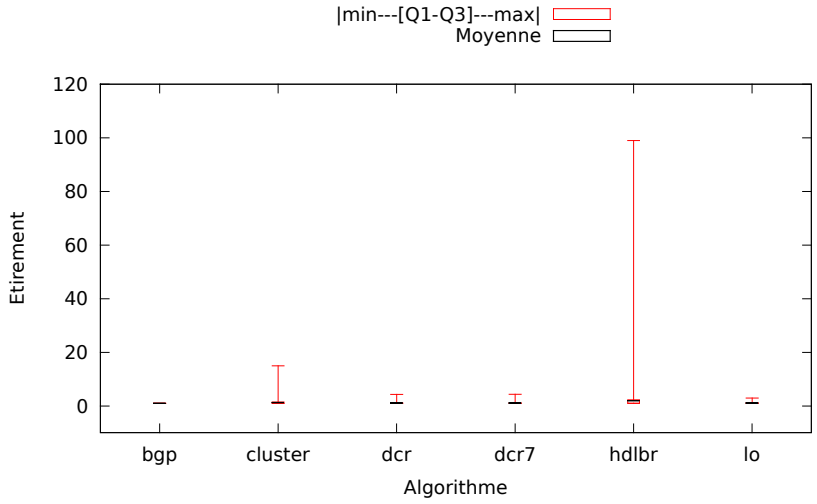

Figure 4.11: Étirements comparatif de tous les algorithmes sur l'UDG carré.

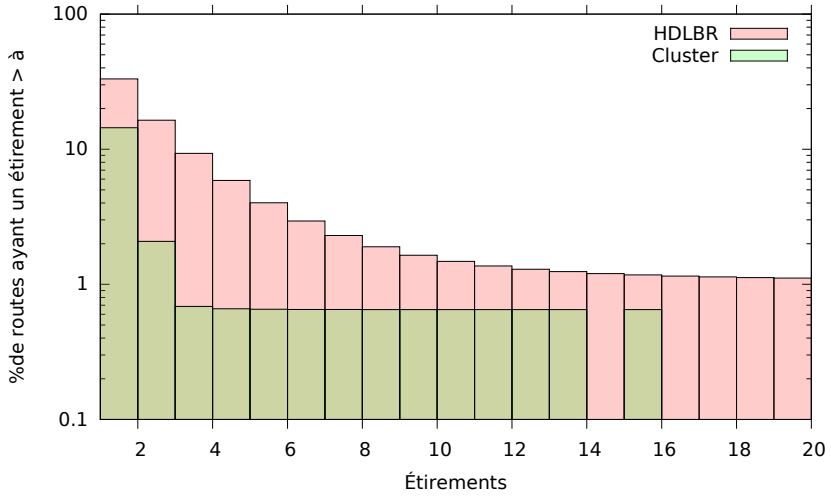

Figure 4.12: Distribution des étirements pour les algorithmes HDLBR et CLUSTER sur un UDG carré.

Il est également important de noter que l'algorithme DCR dans ses deux variantes obtient bien évidement un étirement inférieur à 5 (et 7) dans tous les cas. De plus l'étirement moyen est du même ordre de grandeur que celui d'HDLBR ou de CLUSTER, il n'y a donc aucune raison, considérant l'étirement, pour ne pas choisir l'algorithme DCR plutôt que HDLBR. La figure 4.13 montre la distribution de l'étirement sur CAIDA, sur laquelle on voit que DCR est moins bon que HDLBR uniquement sur

le nombre pourcentage de nœuds ayant un étirement strictement plus grand que 1, mais qu'à partir de 2 c'est DCR qui est le meilleur. Enfin, le meilleur algorithme en terme d'étirement est LO que ce soit en moyenne ou au regard de la distribution et ce pour tous les graphes considérés.

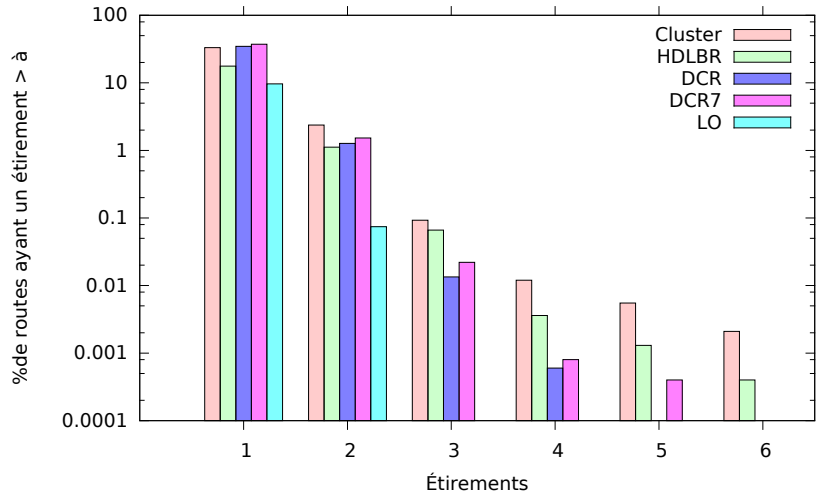

Figure 4.13: Distribution des étirements pour tous les algorithmes sur CAIDA.

4.5.3 Mémoire

Les algorithmes HDLBR et LO ont une très bonne mémoire en moyenne, cependant, les figures 4.14 et 4.15 montrent que ces deux algorithmes affectent à certains nœuds une table de routage bien plus grande que DCR et ce même sur des graphes sans-échelle. Nous pouvons également observer que l'algorithme CLUSTER a une meilleure distribution de mémoire que tous les autres sur un graphe sans-échelle (CAIDA). En contre partie, cet algorithme est le pire sur ce même critère pour un graphe de diamètre plus important (UDG).

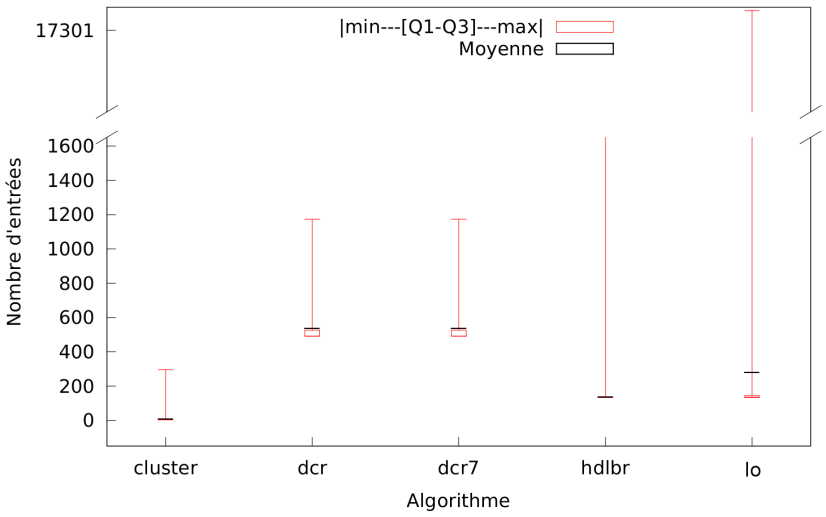

Figure 4.14: caida

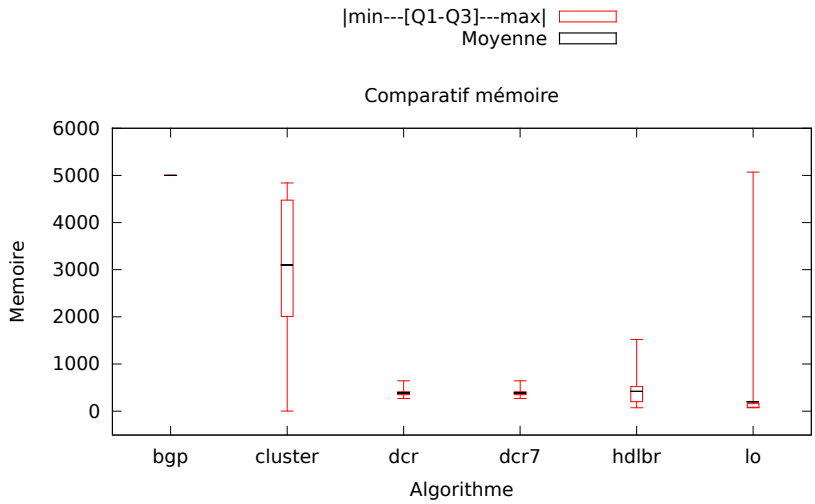

Figure 4.15: udg

4.5.4 Coût de communication

Comparatifs. Sur les figures 4.16 et 4.17 nous pouvons observer que les algorithmes DCR et DCR7 ont une mauvaise répartition de la charge. Cependant, comme nous l'avons déjà fait remarquer dans la conclusion du chapitre 3 il serait possible de réduire significativement cette surcharge en élisant plusieurs landmakrs (un par couleur) et en construisant ainsi chaque arbre logique dans des arbres différents.

La charge des nœuds en messages de contrôle est mal équilibrée pour tous les algorithmes considérés, cependant, cette mauvaise répartition est en partie due à la topologie des graphes. Nous pouvons du moins confirmer cela pour l'UDG, en effet nous pouvons observer que le facteur de variation sur la charge pour l'algorithme de plus court chemin DVECTOR (BGP) est d'environ 21 ($\max = 103699 / \min = 4999$). Ce qui signifie plus ou moins que certains nœuds sont parcourus par 21 fois plus de plus court chemins que d'autres, autrement dit ces nœuds sont centraux. Il parait donc difficile d'obtenir un facteur plus faible à moins d'éviter les plus courts chemins, ce qui n'est pas le cas des algorithmes considérés qui sont tous basés sur des constructions d'arbres de plus court chemin. Notons tout de même que les algorithmes DCR et DCR7, bien qu'ayant un coût de communication moyen bien meilleur que DVECTOR, imposent à certains nœuds une charge environ deux fois plus élevé que celui observé avec ce dernier.

Malheureusement, à cause de contraintes techniques[‡] nous n'avons pas pu obtenir la charge en messages de contrôle pour la carte CAIDA et l'algorithme de plus court chemin de type DVECTOR (BGP).

Enfin, la figure 4.16 permet de confirmer que l'algorithme CLUSTER est le meilleur sur le graphe CAIDA en terme de coût de communication, que ce soit pour les quartiles, la moyenne ou le maximum observé.

[‡]Mémoire RAM insuffisante pour mettre en file les messages.

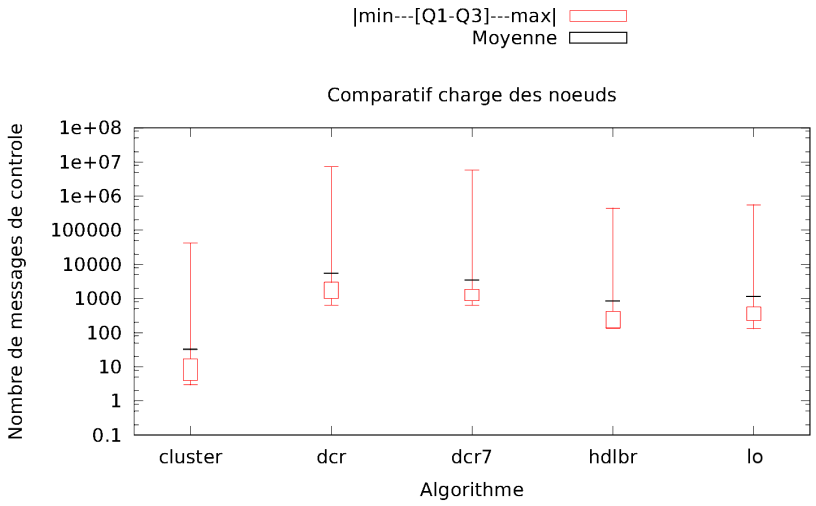

Figure 4.16: Charge des nœuds en messages de contrôle pour la carte CAIDA

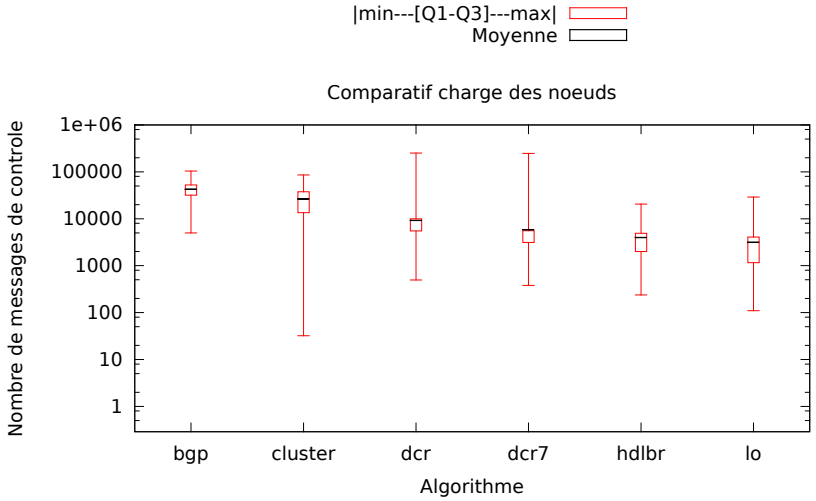

Figure 4.17: Charge des nœuds en messages de contrôle pour un UDG carré

Influence de la latence. Ce paragraphe présente l'influence du paramètre Δ pour le modèle de communication Δ-borné. La tailles des tables et l'étirement ne sont pas (en théorie) influencés par la latence, nous observerons donc uniquement le nombre de messages en fonction de Δ. Ces expériences ne sont données que pour la carte CAIDA. La figure 4.18 montre les résultats obtenus pour l'ensemble des algorithmes.

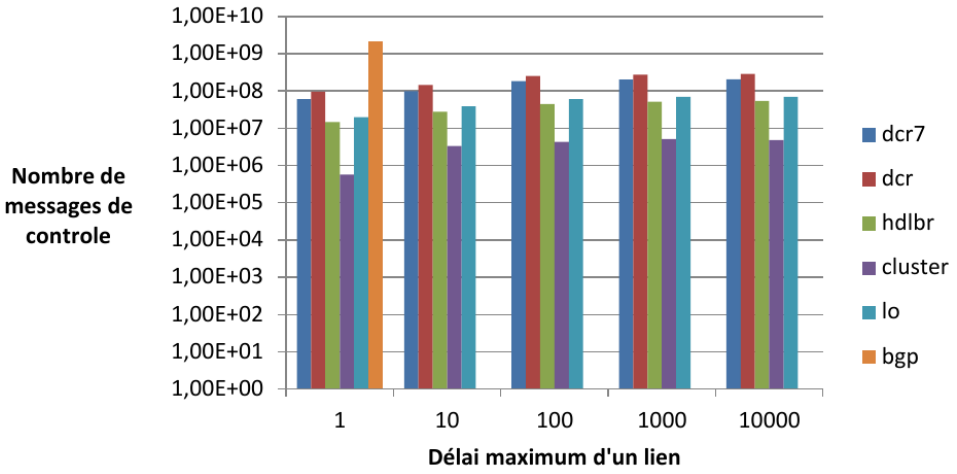

Figure 4.18: Coût de communication en fonction de la latence maximale des liens, i.e. dans le modèle Δ-borné.

Cette figure montre que la latence a, lorsqu'elle est distribuée de manière aléatoire uniforme, une influence relativement faible sur le coût de communication. Quelque soit l'algorithme un facteur d'environ 10 est observé entre le cas 1-borné et le cas 10000-borné. Comparons la valeur théorique dans le modèle $\mathcal{LOCAL}$, aux valeurs observées pour l'algorithme DCR :

- En théorie, le nombre de messages dans le modèle $\mathcal{LOCAL}$ est de :
$$\mathcal{O}\left(m(k\log k + n/k) + \frac{n^2}{k}\cdot\min\{D,k\}\right)$$
soit dans notre cas 37 352 239 messages échangés ;
- En pratique, pour $\Delta = 1$: 95 797 660 messages échangés ;
- En pratique, pour $\Delta = 10000$: 285 480 292 messages échangés.

Soit un facteur de 2.5 entre le cas théorique $\mathcal{LOCAL}$ et le cas expérimental 1-borné et un facteur 7.6 entre théorie et expérimentation pour le cas 10000-borné. Le modèle $\mathcal{LOCAL}$ semble donc être un modèle, qui dans notre cas approche assez fidèlement la réalité, même lorsque Δ est de l'ordre du nombre de nœuds. Qui plus est, un facteur 2 est caché dans la constante de la borne théorique, car la quasi totalité des algorithmes fonctionnent avec des accusés de réception[§]. Il serait évidement intéressant de voir si cette remarque tient toujours dans le cas pondéré.

Enfin pour information, utiliser un α-synchroniseur sur les constructions d'arbres de DCR pourrait être intéressant car, rappelons-le, son utilisation n'implique qu'un surcoût de communication de mD quel que soit la latence, ici 355 470 messages supplémentaires. Cela permettrait d'avoir approximativement la même complexité en 1-borné qu'en 10000-borné.

[§]En calculant la borne de manière plus précise, on peut obtenir une complexité théorique de 81 109 651. Cette complexité est obtenue en prenant en compte les accusés de réceptions et le fait que la valeur m apparaissant sous le $\mathcal{O}(.)$ est en fait la somme des degrés ($2m$) :
$$4m\,(k\log k + n/k) + \frac{n^2}{k}\cdot\min\{D,k\}\ .$$

4.6 Conclusions

4.6.1 Quel algorithme choisir ?

Cette section tente de répondre à la question : “Quel algorithme choisir ?”, en fonction de différents critères de choix qui nous paraissent pertinents.

Unique objectif : réduire les coûts de communication. Si l'unique objectif est de réduire les coûts de communication, alors l'algorithme LO est probablement la meilleure option, il est bon sur tous les graphes pour ce critère. De plus ses complexités en terme d'étirement sont excellentes et garanties théoriquement. Le problème majeur de cet algorithme vient du fait qu'avec certitude, $\sqrt{n}$ noeuds auront une mémoire linéaire. Si la mémoire est un critère important, alors LO n'est clairement pas une option viable. Remarquons notamment que les nœuds ayant une mémoire linéaire sont choisis de manière aléatoire. Ce qui impliquerait dans un contexte réel que les nœuds auraient intérêt à “tricher” pour choisir de ne pas avoir une mémoire linéaire, ce qui aurait pour effet de complètement détruire toute garantie de routage. Donc, si la mémoire est un critère important, ou si les noeuds peuvent tricher, le meilleur choix d'algorithme en tout généralité est probablement DCR.

Proposer une alternative robuste à un algorithme de plus court chemin. L'algorithme DCR serait en effet, une alternative envisageable à DVECTOR, cet algorithme détériorant uniquement l'étirement moyen (d'un facteur d'environ 1.7). Il est est strictement meilleur que DVECTOR sur tous les autres critères. Cependant, si le graphe possède des propriétés de graphe sans-échelle et qu'il est certain qu'il les vérifiera toujours, alors le meilleur algorithme est probablement CLUSTER.

Router dans un graphe sans-échelle. En effet, l'algorithme CLUSTER a un coût de communication et une complexité mémoire bien meilleurs que tous les autres algorithmes présentés pour les graphes sans-échelle. Il est important de remarquer que si le graphe était amené à changer de forme il est très probable que ces complexités se dégradent rapidement, comme le montrent les résultats sur les UDG. Dans le cas du réseau des AS, les cartes obtenues par envoi de sondes dans le réseau, ne sont pas fiables à 100%. Il serait donc risqué dans se cas de choisir ce type d'algorithme pour router dans ce réseau.

Router dans un graphe que l'on suppose être sans-échelle. Dans ce dernier cas, où le graphe a très probablement des propriété de graphe sans échelle, mais sans certitude absolue, l'algorithme HDLBR se positionnant entre DCR et CLUSTER, il serait probablement un bon choix. Ce dernier a de bonnes performances dans les graphes sans-échelle sans pour autant avoir des complexités rédhibitoires dans les autres graphes (considérés dans cette étude). Le point négatif de cet algorithme réside en très grande partie dans l'étirement maximal des routes. Choisir HDLBR implique que certains nœuds du graphes auront des latences bien supérieures à la moyenne. Cet algorithme ne dessert pas les nœuds de manière équitable.

4.6.2 Influence de l'assortativité sur les performances

L'assortativité est une propriété du graphe, qui tend à refléter à quel point les nœuds de forts degrés s'attirent mutuellement, autrement dit, à quel point existe-t-il un *cœur* dans le graphe. Li *et al.* définissent dans [LADW05] une mesure simple et assez proche de l'assortativité. Cette métrique est calculée dans un graphe $G = (V, E)$ ainsi :

$$\int(G) = \sum_{\{u,v\} \in E} \text{degre}(u) \cdot \text{degre}(v)$$

Li *et al.* font également remarquer que la valeur maximale de $\int(G)$ est :

$$\int_{\max} = \sum_{u \in V} (\text{degre}(u) \cdot 1/2) \cdot \text{degre}(u)^2$$

L'assortativité d'un graphe G définie dans [New02] est proportionnelle à $\int(G)$, cependant sa définition est plus complexe et permet d'obtenir une valeur normalisée $\in [-1, 1]$. Il serait intéressant d'étudier l'impact de cette propriété sur les algorithmes dédiés aux graphes sans-échelle. Il est très probable que cette propriété influe sur la mémoire pour CLUSTER et HDLBR car dans ces deux algorithmes la taille de boules dépend notamment de la distance au "cœur" du graphe. L'étirement serait également impacté car la distance entre les nœuds de haut degré est à priori réduite si les nœuds de fort degré sont très inter-connectés.

Il serait intéressant dans cette optique de considérer également le modèle de génération de graphe PARG (*Parallel Addition and Rewiring Growth*) proposé dans [PPZ09] par Piraveenan *et al.*. Ce dernier permettant d'avoir un contrôle plus fin sur l'assortativité du graphe généré.

4.6.3 Idées pour les algorithmes futur

Tous les algorithmes efficaces en pratique sur les graphes petit monde ont de mauvaise garanties d'étirement et/ou de mémoire en toute généralité. Est-il possible de décrire un algorithme ayant de bonne garanties théorique sur la mémoire et d'étirement pour tout graphe tout en restant particulièrement efficace que les algorithmes pour des graphes sans-échelle ? Un premier pas dans cette direction pourrait être obtenu en utilisant l'algorithme DCR et en plaçant simplement les landmarks de façon plus intelligente. Cette idée a déjà été prouvée efficace pour améliorer l'étirement de l'algorithme TZ [TZ01] pour des graphes sans-échelle dans [CSTW09]. La stratégie est assez simple et consiste à placer les landmarks dans le centre du graphe, i.e. sur les nœuds de fort degré. Un placement hybride des landmarks pourrait permettre d'atteindre cet objectif.

4.6.4 Étirement moyen

Nous avons constaté dans cette section que les étirements moyens de tous les algorithmes décrits sont assez bon dans les graphes aléatoires considérés. Une question intéressante à aborder serait : est-ce toujours le cas ? Ou existe-t-il des graphes ou familles de graphes dans lesquels l'étirement moyen est mauvais ? Cette question pourrait être abordée d'un point de vue expérimental, mais également d'un point de vue théorique.

Remarquons que, dans le cas général, il est assez simple de trouver des graphes pour lesquels l'étirement d'HDLBR et de CLUSTER sont proportionnels au diamètre en moyenne (voir figure 4.19).

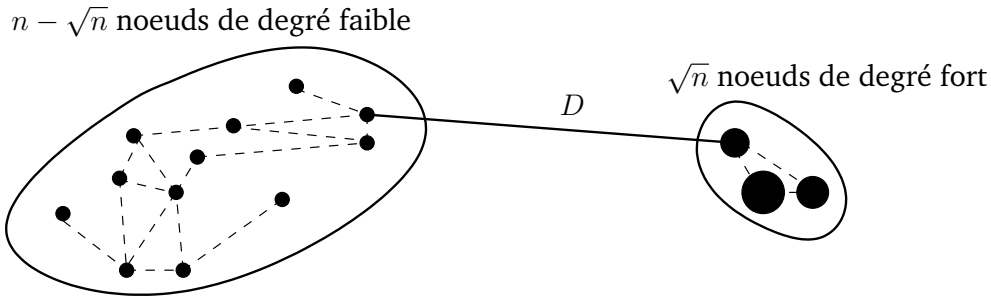

Figure 4.19: Le routage entre la majorité des paires (deux nœuds de l'ensemble de gauche) se fait en passant par les landmarks (ensemble de droite), l'étirement moyen est donc proportionnel au diamètre.

4.6.5 Analyser de manière plus fine les algorithmes HDLBR et CLUSTER

Pour finir, nous avons pu constater qu'il existait un grosse différence entre les valeurs théoriques de $\mathcal{B}$, $\mathcal{B}'$ et les valeurs observées en pratique. Il serait intéressant de donner une analyse plus précise de ces deux paramètres.

Maintenance d'arbres de plus courts chemins dans un réseau dynamique

5

Sommaire

Considérons un graphe $G = (V, E)$, un nœud donné $r \in V$ ainsi que la composante connexe contenant ce nœud r notée V_r. L'objectif des travaux présentés dans ce chapitre est de décrire un algorithme auto-stabilisant permettant de construire un arbre de plus court chemin T_r enraciné en r et couvrant tous les nœuds de V_r. Tout nœud appartenant à T_r devra connaître sa distance à r, ainsi que son parent dans T_r et tout nœud n'appartenant pas à V_r devra déterminer qu'il n'appartient pas à V_r.

Origine de la problématique. Les algorithmes de routage de type *vecteur de distance*, tels que RIP [Hed88] (*Routing Information Protocol*) ou BGP [RLH06] (*Border Gateway Protocol*) sont basés sur la construction d'arbres de plus court chemins. Pour toute destination r, un arbre de plus court chemin enraciné en r est construit par le schéma de routage. Le routage vers le nœud r est garanti uniquement depuis toute source $u \in V_r$. Cependant, aucune assurance n'est donnée quant à la convergence du schéma de routage en présence d'erreurs hors de le composante V_r. Cela a notamment pour effet de saturer les sous-réseaux composés des nœuds $V \setminus V_r$ de messages de contrôle. Cette situation qui empêche la convergence de l'algorithme est liée à la dynamique et est appelée *problème du comptage à l'infini* [LGW04], bien que plus connu sous sa dénomination anglophone : *count-to-infinity problem*. L'apparition de ce problème se résume par la présence de messages d'annonces concernant des routes vers r, entre nœuds n'appartenant pas à V_r, mais essayant coûte que coûte de trouver un chemin vers celui-ci. Un cas de figure dans lequel ce problème survient est le suivant.

Problème du comptage à l'infini. Cette situation peut notamment survenir suite à une suppression d'arête impliquant une division du réseau en deux composantes connexes. Illustrons le problème via l'exemple de la figure 5.1 dans lequel les nœuds calculent une distance et un nexthop vers le nœud r.

Supposons qu'un nœud u détecte la suppression d'une arête adjacente et appartenant au plus court chemin sélectionné par u vers la racine r. Il essaye alors de trouver, via son voisinage, des routes alternatives vers r. Toutefois, il est possible que $u \notin V_r$. Or, si $u \notin V_r$, le nœud u ne devrait, à priori, trouver aucune route vers le nœud r. Cependant, si un voisin de u pense connaître un chemin et le lui propose, u ne sera pas capable de détecter localement qu'il n'appartient plus à V_r et acceptera donc cette nouvelle distance qui représente la longueur d'un chemin inexistant. Les nœuds de la composante connexe du nœud u continueront d'échanger des distances concernant ce chemin inexistant dont la longueur supposée ne fera alors que croître par la création de cycles d'annonces comme le montre la figure 5.1.

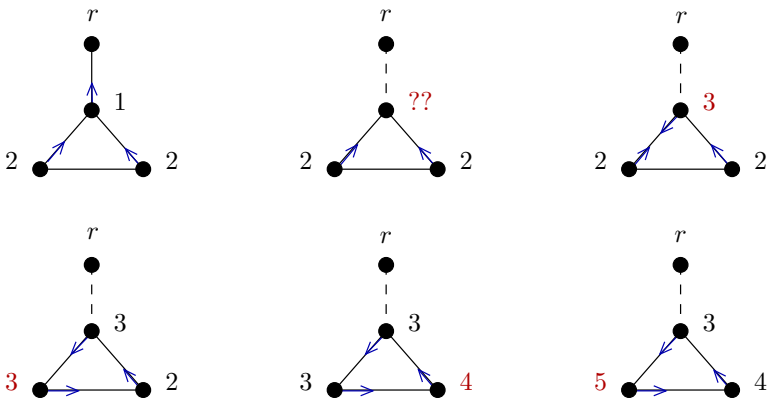

Figure 5.1: Exemple de création de boucle de messages de contrôle. Les flèches représentent les pointeurs indiquant leurs parents supposés dans l'arbre, l'arête en pointillés est supprimée, enfin les distances en rouge marquent les nœuds ayant changé d'état.

Il existe différentes techniques pour pallier ce problème, elle seront décrites brièvement dans la section 5.1. La suite du chapitre présentera le modèle utilisé, en section 5.2, ainsi qu'une nouvelle solution de maintient d'arbre de plus court chemin en cas de déconnexion (temporaire ou définitive) du graphe suite à des suppressions d'arêtes en sections 5.4 et 5.5. Enfin, la preuve de correction de cet algorithme sera présentée dans la section 5.6.

5.1 Travaux connexes

La technique la plus classique et simple pour maintenir à jour des arbre de plus court chemin est de répéter des annonces de vecteur de distance à une fréquence donnée. La convergence de cet algorithme n'est pas garantie, il existe cependant différentes versions de cet algorithme à vecteur de distance permettant d'éviter le problème du comptage à l'infini.

5.1.1 Solutions au problème du comptage à l'infini en corrigeant le protocole à vecteur de distance

Utiliser la connaissance du diamètre. La première solution à ce problème utilise une connaissance sur le réseau pour permettre la détection de la terminaison de construction de l'arbre de plus court chemin T_r. Une borne supérieure δ sur le diamètre du graphe G est considérée connue par tout nœud du réseau. À noter que, par définition, aucun plus court chemin ne peut avoir une longueur supérieure à δ. Tout nœud recevant une annonce de chemin avec une distance plus grande arrêtera la diffusion de cette annonce considérée comme non valide. Cela ne permet cependant pas d'éviter la création de boucle de longueur inférieure à δ lors de la propagation de messages de contrôle.

Cette solution est celle implémentée dans la deuxième version de l'algorithme RIP [Mal94]. La borne supérieure utilisée pour le diamètre est de $\delta = 16$. Le problème de cette solution est qu'elle implique de connaître une borne correcte δ. L'algorithme RIP utilise également une technique connue sous le nom de *split-horizon*. Elle garantit qu'aucune boucle de longueur deux n'est créée en diffusant des annonces à tous ses voisins sauf son parent dans T_r.

Utilisation de timers. La dernière solution en date a avoir été adoptée en pratique est décrite dans [MS97] et propose quant à elle de détecter les routes non valides pour les détruire avec une métrique de distance infinie. On parle alors d'*empoisonnement de routes*. Les nœuds voisins coopèrent en s'échangeant régulièrement des messages pour tester la présence des liens de communication (messages communément appelés *HELLO*). Un nœud détectant une anomalie dans son voisinage, empoisonne la route, annonçant une distance infinie pour le sous réseau devenu inatteignable. Si un nœud perd le lien vers son parent, il prévient ses voisins qu'ils sont peut-être déconnectés de la composante V_r. Lorsqu'un nœud apprend une distance infinie à la racine, il passe dans un état d'attente ($\mathrm{hold\text{-}down}$) et lance un compte à rebours (timer). Un nœud en état d'attente ne prend pas certaines annonces en compte jusqu'à ce que son compteur ait atteint zéro. Pendant cette période, seules les annonces concernant des routes plus courtes ou provenant de son parent sont prises en compte. Une fois le compteur à zéro, le nœud se considère comme déconnecté de la composante V_r et accepte donc toute route proposée, même si elle est plus longue que la route (invalidée) connue. Cette technique est appelée *hold-down timer*. Dans RIP le compteur timer est par défaut initialisé à 180 secondes. Le problème de cette solution est la calibration de ce dernier paramètre. De plus, l'utilisation d'une telle technique a pour effet d'augmenter le temps de convergence.

5.1.2 Autres solutions utilisées en pratique

La solution très simple utilisée dans l'algorithme LSR [MRR80] (*Link State Rounting*), permet d'éviter toute boucle d'annonces. Cet algorithme est basé sur la connaissance de la topologie complète (*état des liens*) par tout nœud, lui permettant ainsi de réaliser un calcul centralisé sur un état représentant la globalité du réseau à un instant donné. Il requiert donc une mémoire $\Theta(m)$ par nœud et entraîne une forte charge du réseau en messages de contrôle ($\Omega(mn)$ dans le cas statique). Le grand nombre de messages de contrôle peut être réduit en pratique dans les réseaux denses, grâce à des techniques comme celles proposées par l'algorithme OLSR [CeA+03]. OLSR est actuellement utilisé dans certains réseaux mobiles, comme *MANET* (*Mobile Ad-hoc*

NETworks). Le principe utilisé pour réduire le nombre de messages de contrôle est assez simple. OLSR utilise des nœuds relais qui effectueront les diffusions, pendant que les nœuds non relais seront uniquement des récepteurs. L'objectif est alors uniquement de définir une couverture (par les nœuds relais) du graphe G, tel que le sous-graphe induit par cette couverture est aussi petite que possible en nombre d'arêtes *.

Une autre alternative, connue sous le nom de DSDV (*Destination Sequenced distance-vector*), est également utilisée dans les réseaux mobiles. L'algorithme DSDV utilise une mémoire dépendant du nombre d'arbres construits dans le réseau, $\mathcal{O}(q \log n)$ bits d'information pour q arbres. Il est présenté dans [PB94]. Cet algorithme est basé sur un algorithme Bellman-Ford distribué. La différence est que la métrique utilisée n'est pas basée uniquement sur la distance à la racine de l'arbre, mais également sur un *numéro de séquence*, celui-ci représentant en quelque sorte l'âge de l'information. La racine de l'arbre initie, à une fréquence donnée, un calcul distribué d'arbre de plus court chemin. À chaque émission, elle incrémente son numéro de séquence, permettant ainsi de faire la différence entre une information passée ayant bouclé et une information récente. Cet algorithme garantit l'absence de boucle dans la diffusion de messages de contrôle. Une variante réactive de DSDV, nommée AODV est proposée dans [PR99]. Le terme réactif signifie que la route est à priori inconnue tant qu'elle n'est pas demandée par la source. Autrement dit, les nœuds ne connaissent que les destinations vers lesquelles ils ont eu besoin de router. Cette alternative permet d'économiser de la mémoire et d'alléger la charge du réseau en sacrifiant la latence lors du routage vers une nouvelle destination. La latence est liée à l'attente de la création de la route requise par un processus de demande/accusé de réception. La variante AODV est en pratique plus utilisée que la version proactive DSDV.

Un comparaison expérimentale entre les protocoles OLSR et DSDV est présentée dans [Huh04].

5.1.3 Algorithmes auto-stabilisants

Le point de vue *auto-stabilisant*, présenté originellement par Dijkstra [Dij74], est le suivant. L'ensemble des états des nœuds est appelé *configuration*. Le graphe sur lequel les calculs sont effectués est non dynamique. Cependant, la configuration initiale considérée par l'algorithme est arbitraire. Cette approche permet de s'abstraire du type de fautes pouvant apparaître en observant uniquement l'état du réseau après, par exemple, une successions de pannes ou d'attaques. Les états des nœuds du réseau ne sont pas nécessairement cohérents. Un algorithme auto-stabilisant permet de converger vers une configuration stable et légitime (ces terminologies seront définies en section 5.2) quelle que soit la configuration initiale considérée et ce en un temps fini. Le modèle de communication stipule que chaque nœud ou processus dispose d'un état que seul lui et ses voisins peuvent voir, on parle alors de *mémoire partagée* [Dol00].Le modèle d'exécution usuellement considéré est celui d'une entité, appelée *démon*, décidant d'un ordre d'exécution sur les différents processus.

L'ensemble des nœuds ou processus activables est le sous-ensemble de V tel que tout nœud appartenant à cet ensemble peut changer d'état (il vérifie un des prédicats ou *garde* de l'algorithme).

*Le gain sur le nombre de messages étant directement proportionnel au ratio nombre d'arêtes dans G sur nombre d'arête dans le sous graphe induit par la couverture.

De nombreux articles tels que [CS94, HL02], proposent des algorithmes auto-stabilisants permettant de calculer un ou plusieurs arbres de plus court chemins en considérant un démon *centralisé*, i.e. un seul processus activable s'exécute à un temps donné.

D'autres études considèrent que plusieurs nœuds activable peuvent s'exécuter simultanément. Le terme *démon distribué* [BMG88] est utilisé pour caractériser l'*adversaire* décidant de façon arbitraire quels seront les nœuds/processus activables exécutant une action durant le prochain pas de calcul. Les pas de calculs sont effectués simultanément et de façon atomique.

Les algorithmes considérant un démon centralisé présentés dans [HL02, CS94] sont prouvés valides dans le modèle de démon distribué par Huang dans [Hua05].

Dans [Dol93], Dolev présente un algorithme auto-stabilisant permettant de calculer n arbres dans un environnement dynamique en un nombre optimal de rondes (ce terme est défini dans la section 5.2). La mémoire est, dans ce dernier, partagé via des registres, le modèle de calcul considéré est l'atomicité de la lecture et de l'écriture dans les registres. Tout couple de nœuds voisins possède un registre dans lequel chacun d'entre eux peut écrire et lire. La mémoire requise par les nœuds est la même que pour l'algorithme DSDV, soit $\mathcal{O}(n \log n)$, de plus le nombre de rondes est de $\mathcal{O}(D)$ et le nombre de pas de calculs de $\mathcal{O}(n^2)$. Dans [Joh97], Johnen montre qu'il est également possible de construire un arbre, sans connaissance de la distance, et ainsi obtenir une mémoire constante par lien pour chaque arbre construit dans le modèle à état défini par Dijkstra dans [Dij74].

D'autres algorithmes tels que [AG94, AKY91, GP03] donnent des résultats similaires à ceux de [HL02, CS94, DIM93] et utilisent des idées algorithmique proches, quasiment toutes basées sur le principe de vecteur de distance. Les différences viennent principalement de la connaissance supposée des nœuds sur le diamètre ou la taille du réseau ou encore du modèle d'*anonymat* utilisé, certains considérant que les nœuds sont indistinguables à l'exception de la racine (*quasi-uniforme*). D'autres utilisent un ensemble de nœuds ayant tous un identifiant unique. Il est prouvé dans [Ang80], que dans le cas uniforme (tous les nœuds exécutent le même code), il est impossible de construire de manière déterministe un arbre couvrant, et donc a fortiori un arbre de plus court chemin.

5.1.4 En résumé

Les correctifs apportés à l'algorithme de vecteur de distances requièrent une connaissance sur le diamètre du graphe ou au moins une connaissance sur le nombre de nœuds, qui permettrait de borner le diamètre par n.

Une hypothèse est commune à toutes les solutions alternatives à l'algorithme vecteur de distances. Les algorithmes ne considèrent que des nœuds appartenant à la composante connexe de la racine. Dans notre cas, nous souhaitons un algorithme auto-stabilisant permettant de garantir que tous les nœuds d'un réseau donné sont considérés. Notamment, un nœud n'appartenant pas à la composante connexe de la racine doit être capable de le détecter et donc de décider que sa distance à la racine "n'existe pas" ou est infinie, et ce de manière auto-stabilisante. Il n'existe pas d'étude du problème de détection de connectivité de façon auto-stabilisante ou dynamique et distribué.

L'algorithme présenté dans ce chapitre ne requiert aucune connaissance sur le graphe d'entré est auto-stabilisante et converge vers un état stable quelle que soit la connectivité du graphe.

5.2 Modèle Auto-Stabilisant

On considère un graphe $G = (V, E)$ non pondéré et non nécessairement connexe.

Modèle à état. L'état d'un nœud est défini par l'ensemble de ses variables, la variable X d'un nœud u sera notée X_u. La mémoire est partagée entre voisins, comme décrit dans [Dij74], tout nœud connaît l'état de l'ensemble de ses voisins en plus son propre état.

Configurations. L'ensemble des états des nœuds du réseau est appelé *configuration*. Deux configurations spécifiques sont considérées : la *configuration initiale*, observée en début de calcul, et la *configuration terminale*, observée après le calcul, lorsque plus aucune règle n'est appliquée. Une configuration est *légitime* si l'état de tous les nœuds sont corrects vis-à-vis des objectifs du problème donné. Par exemple, les distances stockées par les nœuds correspondent aux distances réelles dans le graphe G.

Pas de calcul. Pour un nœud donné $u \in V$, chaque *pas de calcul* est constitué de la lecture des états de tous ses voisins et de son état, puis d'un calcul local et enfin d'un changement d'état.

Plus formellement, le calcul local correspond à la vérification de *gardes*, écrites sous forme de prédicats booléens. Par exemple, on peut énoncer le prédicat "être seul", qui indique si u est l'unique nœud de sa composante connexe, comme suit

$$P_{\text{être seul}}(u) \equiv \Gamma(u) = \varnothing$$

Si la garde P est vérifiée par un nœud u, le nœud est dit *activable* par la garde P. Un nœud activable par P *exécute* l'*action* associée à cette garde dès qu'il est choisit par l'ordonnanceur (*démon*). Les correspondances entre les gardes et les actions sont appelées *règles*.

Démon ou Ordonnanceur. Le modèle d'ordonnanceur considéré est celui d'un démon *distribué* et *faiblement équitable*. Faiblement équitable signifie que tout processus continuellement activable sera activé par le démon après un temps fini. Autrement dit un processus ne peut pas être ignoré infiniment longtemps par le démon. Cela équivaudrait en terme de modèle de passage de message à des latences bornées sur les liens de communication.

Nœud neutralisé. Un nœud est dit neutralisé s'il devient non activable suite au changement d'état d'un autre nœud.

Exécution. On appelle *exécution* d'un algorithme A une suite de configurations obtenues en appliquant les règles définies par A.

Ronde. Une ronde d'exécution d'un algorithme donné se termine lorsque tous les nœuds activables au début de la ronde (à la fin de la ronde précédente) ont été soit activés soit neutralisés.

Auto-stabilisation Un algorithme est dit *auto-stabilisant* s'il converge en un nombre fini de rondes vers une configuration terminale légitime.

5.3 Résumé des résultats

L'algorithme DECO présenté dans les sections suivantes permet de répondre au problème du comptage à l'infini en détectant la déconnexion d'un sous-ensemble de nœud de la composante V_r.

Théorème 5.1. *La configuration initiale est arbitraire. Dans le modèle de démon distribué faiblement équitable, après un nombre fini de pas de calculs, si aucune erreur ne survient pendant l'exécution de l'algorithme* DECO *sur* G *alors* DECO *converge vers un état terminal dans lequel pour un nœud* $u \in V$*,*

- *si* $u \in V_r$*, alors le nœud* u *stocke sa distance à* r *ainsi que l'identité d'un voisin appartenant à un plus court chemin vers* r *;*
- *sinon le nœud* u *n'a pas de parent et sait qu'il n'appartient pas à la composante connexe* V_r*.*

Ce théorème sera prouvé en deux parties, dans les théorèmes 5.3 et 5.2 de la section 5.6.

5.4 Notations et définitions

Variables des nœuds/états. Pour un graphe $G = (V, E)$, l'état de tout nœud $u \in V$ est défini par les valeurs des variables suivantes :

- d_u destiné à être la distance de u à la racine r dans le graphe G, la distance stockée est à tout moment $d_u \in \mathbb{N}$, $d_u < \infty$;
- parent_u un nœud destiné à être un voisin appartenant à un plus court chemin de u à r ;
- st_u le statut du nœud u, avec $\mathrm{st}_u \in \{I, C, E\}$, ces états potentiels ayant pour signification respectives `Isolé` , `Correct` et `Erreur`. Intuitivement, un nœud ayant le statut :

  I) `Isolé` est supposé ne pas appartenir à la même composante connexe que r dans G ;

  C) `Correct` est supposé être dans la même composante connexe que le nœud r et donc faire partie de l'arbre enraciné en r ;

  E) `Erreur` est supposé être dans un état incertain, le nœud ne peut pas déterminer s'il est dans l'état C ou I.

  Ainsi, les statuts I et C sont les seules valeurs terminales légitimes. Autrement dit, aucun nœud n'est censé se stabiliser avec un statut E.

Remarque. Par abus de langage le terme "état" suivi d'un de ces statuts sera utilisé. Le statut d'un nœud donnant dans la majorité des cas suffisamment d'informations pour énoncer des propriétés/théorèmes ou preuves sans ambiguïtés.

Nœud racine. Le nœud racine r est capable de déterminer seul qu'il est racine de T_r. Son état est donc à tout instant le suivant

$$d_r = 0, \mathrm{st}_r = C, \mathrm{parent}_r = \mathrm{NULL}$$

Le nœud racine n'est pas soumis à l'application des règles suivantes et ne change donc jamais d'état.

Définition 5.1 (Enfants)**.** *Tous les nœuds ont un accès en lecture aux états de leurs voisins. Il est donc possible pour un nœud de connaître l'ensemble des nœuds voisins l'ayant choisi comme parent, autrement dit il peut déterminer qui sont ses* enfants*. Pour faciliter la lecture, par la suite, l'ensemble des enfants d'un nœud u sera utilisé sans se référer explicitement aux états des voisins de u. Cet ensemble se définit plus formellement comme suit :*

$$\mathrm{enfants}_u = \{v \in \Gamma(u) \mid (\mathrm{parent}_v = u) \wedge (d_v > d_u)\}$$

5.5 Algorithme DECO

5.5.1 L'idée derrière l'algorithme DECO

Pour aider à la compréhension de cet algorithme commençons par une explication dans un cas simple qui permettra de donner quelques intuitions de l'algorithme décrit en détail dans les sections 5.5.2, 5.5.3 et 5.5.4. Supposons que l'état des nœuds est cohérent puis qu'une unique arête $\{u, v\}$ appartenant à l'arbre de plus court chemin calculé par l'algorithme est supprimée. Macroscopiquement, trois situations peuvent être observées :

1. Il existe une arête $\{u, v'\}$ telle que $d(v', r) \leqslant d(v, r)$: Dans ce cas seul le nœud u changera d'état et se rattachera à son nouveau parent v' (voir prédicat P_{corriger}).

2. Le graphe est toujours connexe mais il n'existe pas de parent alternatif à v (sous-entendu plus proche que de r que v) : Dans ce cas, le nœud u passera dans l'état d'erreur et cet état se propagera dans son sous-arbre (voir prédicat P_{erreur}). La propagation de l'état d'erreur s'achève dès qu'un nœud w possède un parent alternatif dans son voisinage, tout nœud appartenant au sous-arbre de w ne changera pas d'état et conservera un statut C. Enfin, si aucune alternative n'est trouvée dans un sous-arbre, les nœuds passeront progressivement dans l'état I (voir prédicat P_{isoler}), cette propagation de statut se fera depuis les feuilles en remontant vers u et ce pour tout nœud ayant l'intégralité de son sous-arbre avec un statut I. Le passage dans l'état I impliquant l'oubli de son père par le nœud concerné. Enfin, un nœud est autorisé à se connecter à un voisin quelconque (sans regard sur la distance de ce dernier, en considérant uniquement les nœuds dans l'état C) si et seulement si il n'a plus d'enfants (voir prédicat P_{creer}). Toutes les distances de l'ancien sous-arbre de u devront être recalculées. Ce dernier cas de figure est présenté en figure 5.2.

3. Le graphe est déconnecté, $u \notin V_r$: Ce dernier cas est très similaire au précédent, cependant aucun nœud dans le sous-arbre de u ne trouvera un voisin n'appartenant pas au sous-arbre de u. Tous les nœuds du sous-arbre enraciné en u se trouveront donc dans l'état I.

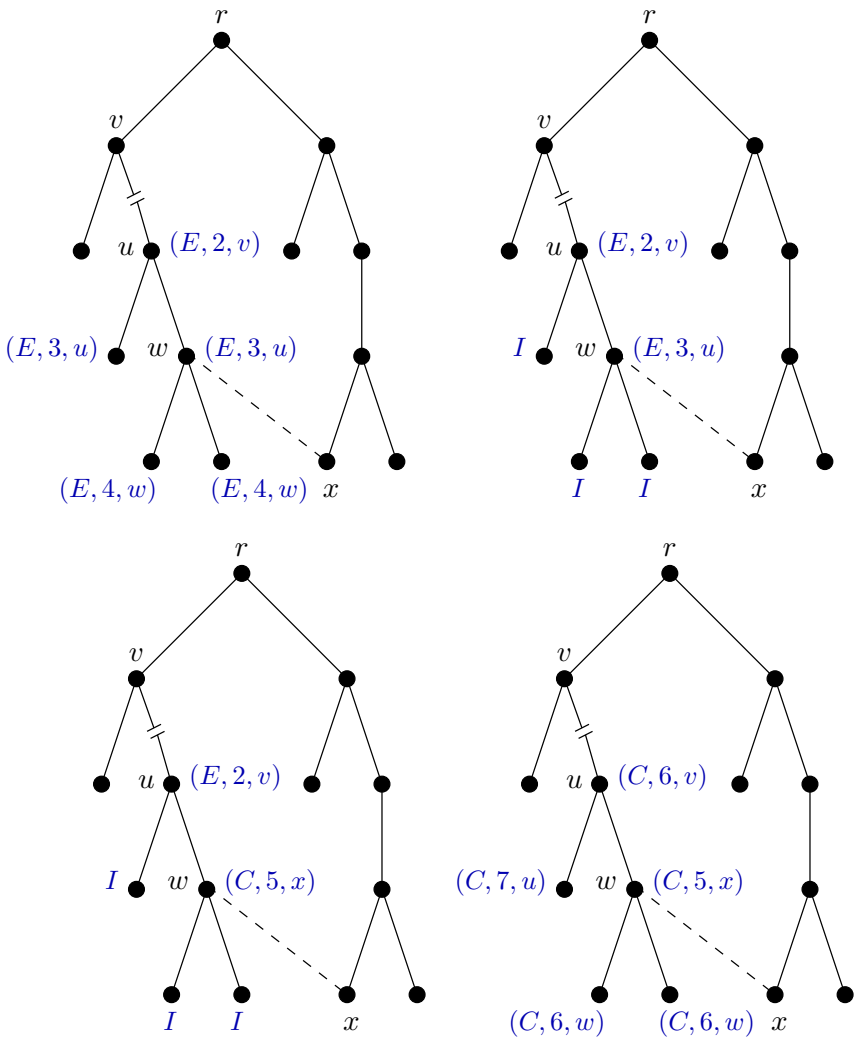

Figure 5.2: L'arête en pointillés représente l'unique arête du graphe G que l'algorithme DECO n'utilise pas pour l'arbre enraciné en r. L'état d'un nœud u est représenté par le triplets $(\mathrm{st}_u, d_u, \mathrm{parent}_u)$, la distance et le parent ne sont pas précisés pour les nœuds de statut I car ces nœuds se considèrent comme déconnectés, les valeurs de ces métriques ne sont donc pas pertinentes.

L'algorithme DECO est décrit formellement par la suite sous la forme de règles. Chaque règle étant décrite par une union de gardes, ou prédicats, entraînant un même changement d'état. Les règles sont regroupées en trois catégories. La première catégorie, présentée en section 5.5.2, contient la règle relatives à la construction de l'arbre de plus court chemin. La seconde, présentée en section 5.5.3, contient la règle liée à la détection d'incohérences locales. La dernière quant à elle contient la règle relative à la détection de déconnexion, i.e. non appartenance à la composante connexe V_r. Toutes les gardes et actions qui suivent sont données pour un nœud $u \in V$.

5.5.2 Calculs de distances R_C

Si un nœud u vérifie une des gardes de cette sous-section ($P_{\text{creer}}(u)$, $P_{\text{ameliorer}}(u)$ ou $P_{\text{corriger}}(u)$) est vérifiée, alors le nœud u passe dans l'état C par l'action A_C. Le passage dans cet état correspond intuitivement à l'attachement de u à une branche de l'arbre T_r.

Entrée dans T_r. Si le nœud u n'est pas dans l'état correct et qu'il possède un parent potentiel dans son voisinage, alors u exécute A_C. Plus formellement, si le prédicat suivant est vérifié alors l'action A_C est exécutée :

$$P_{\text{creer}}(u) \equiv (\text{st}_u \neq C) \wedge (\text{enfants}_u = \varnothing) \wedge (\exists v \in \Gamma(u) \mid \text{st}_v = C)$$

Amélioration de la distance connue à r. Si le nœud u possède un voisin lui permettant d'améliorer sa distance à r, alors u exécute A_C. Plus formellement, si le prédicat suivant est vérifié alors l'action A_C est exécutée :

$$P_{\text{ameliorer}}(u) \equiv (\exists v \in \Gamma(u) \mid (\text{st}_v = C) \wedge (d_v + 1 < d_u))$$

Changement de parent dans r. Si le nœud u détecte une anomalie concernant sa relation avec son parent actuel, mais qu'un de ses voisins est un "aussi bon parent" que ce dernier, alors u exécute A_C. Plus formellement, si le prédicat suivant est vérifié alors l'action A_C est exécutée :

$$\begin{aligned} P_{\text{corriger}}(u) \equiv &((\text{parent}_u \notin \Gamma(u)) \vee (d_u \neq d_{\text{parent}_u} + 1) \vee (\text{st}_{\text{parent}_u} \neq C) \vee (\text{st}_u \neq C)) \\ &\wedge (\exists v \in \Gamma(u) \mid (\text{st}_v = C) \wedge (d_v + 1 = d_u)) \end{aligned}$$

Définition 5.2 (A_C : Choix d'un nouveau parent)**.** *Pour un nœud donné $u \in V$, notons v le nœud voisin de u tel que : $\text{st}_v = C$ et $d_v = \min\{d_w | w \in \Gamma(u) \wedge (\text{st}_w = C)\}$. Lorsque le nœud $u \in V$ vérifie une des trois gardes de R_C, il met à jour son état, à savoir :*

$$P_{\text{creer}}(u) \vee P_{\text{ameliorer}}(u) \vee P_{\text{corriger}}(u) \rightarrow \begin{cases} \text{st}_u \leftarrow C \\ \text{parent}_u \leftarrow v \\ d_u = d_v + 1 \end{cases}$$

5.5.3 Détection d'incohérences locales R_E

Le passage dans cet état correspond intuitivement à la détection d'une anomalie locale. Le nœud se met alors dans un état de stase, cet état représente l'impossibilité de décider localement d'une transition vers I ou C. Plus formellement, si le prédicat suivant est vérifié alors l'action A_E est exécutée :

$$P_{\text{erreur}}(u) \equiv (\text{st}_u = C) \wedge (\forall v \in \Gamma(u) \mid (d_u \leqslant d_v) \vee (\text{st}_v \neq C))$$

Définition 5.3 (A_E : Mise en attente)**.** *Lorsqu'une anomalie est détectée localement, le nœud passe dans l'état erreur :*

$$P_{\text{erreur}}(u) \rightarrow (\text{st}_u \leftarrow E)$$

5.5.4 Détection de déconnexion R_I

Le passage dans cet état correspond intuitivement au retrait du nœud de l'arbre T_r, i.e. à la détection de non-appartenance à V_r.

$$\begin{aligned} P_{\text{isoler}}(u) \equiv (\text{enfants}_u = \varnothing) \wedge (\forall v \in \Gamma(u) \mid \text{st}_v \neq C) \\ \wedge (\text{st}_u \neq C) \wedge (\ (\text{st}_u \neq I) \vee (\text{parent}_u \neq \text{NULL})\ ) \end{aligned}$$

Définition 5.4 (A_I : Déconnexion/Isolement)**.** *Lorsqu'un nœud ne peut pas trouver de parent parmi ses voisins et qu'il n'est plus parent et ne peut plus le devenir, le nœud passe dans l'état isolé. Plus formellement, si le prédicat P_{isoler}est vérifié alors l'action A_I est exécutée :*

$$P_{\text{isoler}}(u) \rightarrow \begin{cases} \text{st}_u \leftarrow I \\ \text{parent}_u \leftarrow \text{NULL} \end{cases}$$

5.6 Preuve de correction

Lemme 5.1. *Les règles R_C, R_E et R_I sont disjointes deux à deux.*

Preuve. Remarquons que pour un nœud $u \in V$, si le prédicat $P_{\text{isoler}}(u)$ est vérifié alors aucun des prédicats de R_C ne peut être vérifié, en effet :

$$\begin{aligned} P_{\text{isoler}}(u) &\implies (\forall v \in \Gamma(u) \mid \text{st}_v \neq C) \\ &\implies (\neg P_{\text{creer}}(u) \wedge \neg P_{\text{ameliorer}}(u) \wedge \neg P_{\text{corriger}}(u)) \end{aligned}$$

En ce qui concerne les règles R_I et R_E remarquons qu'elles sont disjointes car elles nécessitent un statut différent pour être activées :

$$\begin{aligned} P_{\text{erreur}}(u) &\implies \text{st}_u = C \\ &\implies \neg P_{\text{isoler}}(u) \end{aligned}$$

Enfin, les règles R_E et R_C sont disjointes car R_C requiert qu'un voisin de u soit un parent potentiel quand R_E requiert qu'aucun voisin ne puisse être parent :

$$\begin{aligned} P_{\text{erreur}}(u) &\implies (\forall v \in \Gamma(u) \mid (d_u \leqslant d_v) \vee (\text{st}_v \neq C)) \\ &\implies (\neg P_{\text{creer}}(u) \wedge \neg P_{\text{ameliorer}}(u) \wedge \neg P_{\text{corriger}}(u)) \end{aligned}$$

∎

Cette remarque sera utilisée de manière implicite dans les preuves suivantes, lorsqu'une règle est applicable, par définition aucune autre ne l'est.

Définition 5.5 (Nœud racine, racine illégale)**.** *Le nœud u est racine ssi. :*

$$P_{\text{racine}}(u) \equiv ((\text{parent}_u \notin \Gamma(u)) \vee (d_{\text{parent}_u} \geqslant d_u)) \wedge ((\text{st}_u = C) \vee (\text{enfants}_u \neq \varnothing))$$

Un nœud u est qualifié de racine illégale *si $u \neq r$.*

Définition 5.6 (Branche)**.** *Une* branche *est une suite maximale de nœuds $(u_1, u_2, ..., u_p)$ telle que $p > 1$ et $\forall i, 1 \leqslant i \leqslant p-1 : u_i \in \text{enfants}_{u_{i+1}}$ Une branche se terminant par une racine illégale est appelée* branche illégale*. En opposition, toute autre branche sera qualifiée de* légale*.*

Remarque. Une branche se termine nécessairement par une racine et ne contient pas de cycle.

Lemme 5.2. *Aucune racine illégale ne peut être créée par les règles R_C, R_E ou R_I.*

Preuve. Considérons un pas de calcul faisant passer un nœud $u \in V$ d'un état e_1 à un état e_2. Ces états sont tels que u n'était pas racine et le devient. Dans l'état e_1 on a donc

$$\neg P_{\text{racine}}(u) \equiv ((\text{parent}_u \in \Gamma(u)) \wedge (d_{\text{parent}_u} < d_u)) \vee ((\text{st}_u \neq C) \wedge (\text{enfants}_u = \varnothing))$$

Étudions donc quelles règles pourraient amener u dans l'état racine illégale à partir de ces deux hypothèses. Si $\neg P_{\text{racine}}(u)$ est vérifié, alors l'un des deux prédicats suivant est vérifié :

Cas 1 $((\text{st}_u \neq C) \wedge (\text{enfants}_u = \varnothing))$: Le nœud u ne peut pas être choisi comme parent car $\text{st}_u \neq C$. Donc $\text{enfants}_u = \varnothing$ reste vérifié et $\neg P_{\text{racine}}(u)$ est vérifié. Cependant, u peut également vérifier $P_{\text{corriger}}(u)$, l'état de u peut donc devenir correct en appliquant A_C, mais dans ce cas u choisit un parent parmi ses voisins et $d_{\text{parent}_u} < d_u$ est alors vérifié. Or, $(\text{parent}_u \in \Gamma(u)) \wedge (d_{\text{parent}_u} < d_u) \implies \neg P_{\text{racine}}(u)$, donc u ne devient pas une racine illégale dans ce cas.

Cas 2 $((\text{parent}_u \in \Gamma(u)) \wedge (d_{\text{parent}_u} < d_u))$: Pour que $P_{\text{racine}}(u)$ soit vérifié il est requis que $(\text{parent}_u \notin \Gamma(u)) \vee (d_{\text{parent}_u} \geqslant d_u)$ soit vérifié.

- La seule règle permettant d'obtenir $\text{parent}_u \notin \Gamma(u)$ est R_I, mais en appliquant R_I le nœud u passe dans l'état I.
- Il est donc nécessaire que u change de parent. Or la sélection d'un nouveau parent implique, après exécution de A_C, que $d_u = d_{\text{parent}_u} + 1$ et donc que $(d_{\text{parent}_u} < d_u)$. Le nœud u ne peut donc pas devenir une racine illégale dans ce cas.

∎

Définition 5.7 (État à un instant donné)**.** *La valeur d'une variable X d'un nœud v à l'instant t sera notée $X_v(t)$, par exemple la distance du nœud v à l'instant t est $d_v(t)$.*

Théorème 5.2. *Toute exécution de l'algorithme* DECO *dans la composante connexe V_r est finie et, dans une configuration terminale, tout nœud u appartenant à l'ensemble $V_r \setminus \{r\}$ vérifie :*

$$\begin{aligned} P(u) \equiv &(\text{st}_u = C) \wedge (d_u = d(u,r)) \\ &\wedge (\text{parent}_u \in \Gamma(u)) \wedge (d_{\text{parent}_u} = d_u - 1) \end{aligned}$$

Preuve. Pour tout temps t donné, soit $S(t)$ l'ensemble des sommets u de V_r étant dans l'état C et dont la variable $d_u(t)$ est strictement plus petite que sa distance réelle à la racine :

$$S(t) = \{u \mid (st_u(t) = C) \wedge (d_u(t) < d(u,r)) \wedge (u \notin V_r)\}$$

Soit $d_{\min}(t) = \min_{u \in S(t)} d_u(t)$, si $S(t)$ est vide, nous posons $d_{\min}(t) = +\infty$.

- **Fini implique borné.** *Si $d_{\min}(t)$ est fini (autrement dit $S(t)$ est non vide), alors $d_{\min}(t)$ est borné supérieurement par l'excentricité de r dans la composante V_r :*
  En effet, pour tout nœud $u \in S(t)$, la distance $d_u(t)$ est inférieure à $d(u,r)$ donc

$$d_{\min}(t) = \min_{u \in S(t)} d_u(t) \leqslant \min_{u \in S(t)} d(u,r) \leqslant \max_{u \in V} d(u,r)$$

- **Croissante.** *La fonction $d_{\min}(t)$ est croissante :*
  Pour faire décroître la valeur de $d_{\min}$ il est nécessaire qu'un nœud change de distance, autrement dit il faut qu'un nœud v applique la règle R_C. Notons t l'instant avant application de R_C par v et t' l'instant après application de la règle. Pour que $d_{\min}(t') < d_{\min}(t)$, il faut que le nœud v choisisse un un nouveau parent u induisant $d_v(t') < d(v,r) < d_{\min}(t)$. Observons alors la distance stockée par le nœud u à l'instant t : $d_u(t) = d_v(t') - 1$, ce qui implique que $d_u(t) < d_v(t') < d_{\min}(t)$ ce qui est, par définition de $d_{\min}(t)$, impossible.

  Enfin, pour un instant t donné, si $S(t)$ est vide, alors d'après la première observation (fini implique borné), pour tout $t' < t$, $d_{\min}(t') \leqslant d_{\min}(t)$.

- **Type de croissance de la fonction.** *Si $d_{\min}(t)$ est fini alors il existe un instant t' tel que $t' > t$ et $d_{\min}(t') > d_{\min}(t)$:*
  En effet, pour tout nœud $u \in S(t)$ tel que $d_u(t) = d_{\min}(t)$, tout les voisin v de u est soit dans $S(t)$ soit a une distance $d_v(t) \geqslant d_u(t)$ le prédicat $P_{\text{erreur}}(u)$ est donc vérifié à l'instant t et jusqu'à ce que le nœud u applique R_E. Le démon d'exécution étant faiblement équitable il existe donc un instant auquel tous ces nœuds exécuterons R_E. Pour tout instant t il existe un instant $t' > t$ tel que $d_{\min}(t') > d_{\min}(t)$.

Ainsi il existe un instant t tel que $d_{\min}(t) = +\infty$ autrement dit il existe un instant t à partir duquel $S(t)$ est vide.

Prouvons maintenant par récurrence sur la distance à r que la distance stockée pour tout nœud $u \in V_r$ sera à jour après un temps fini, i.e. aura $d_u = d(u,r)$ ainsi qu'un parent valide $d_u = d_{\text{parent}_u} + 1$:

- **Hypothèse de récurrence :** Il existe un instant t à partir duquel tout nœud $u \in V_r$ à distance i de r est à jour et le reste.

- **À distance 0 :** La racine sait qu'elle est racine et donc aura à tout moment ses variables à jour.

- **À distance $i+1$:** Considérons un nœud u à distance $d(u,r) = i+1$, par hypothèse de récurrence, il existe un instant t à partir duquel tout nœud v voisin de u tel que $d(v,r) = i$ a sa distance à jour : $d_v = i$. De plus il existe un instant t' à partir duquel $S(t')$ est vide. Donc, à l'instant $t_i = \max\{t, t'\}$ aucun voisin de u n'est à distance $< i$ et il existe au moins un nœud v voisin de u tel que $d(v,r) = i$. Observons maintenant le nœud u à l'instant t_i :

  - si le nœud u est à jour, son parent est un nœud à distance i de r et il stocke $d_u = i+1$;
  - sinon le nœud u vérifie en fonction de sa distance actuelle et celle de son parent $P_{\text{ameliorer}}$ ou P_{corriger} (respectivement sa distance était supérieure à $d(u,r)$ ou sa distance était égale mais son parent n'était pas correct). Après application de la règle R_C le nœud u sera à jour.

∎

Théorème 5.3. *Toute exécution de l'algorithme* DECO *hors de la composante connexe V_r est finie et tout nœud u n'appartenant pas à la composante V_r vérifie :*

$$(\mathrm{st}_u = I) \wedge (\mathrm{parent}_u = \mathrm{NULL})$$

Dans la preuve de ce théorème, la notion de longueur de branche est utilisée :

Définition 5.8 (Longueur de branche)**.** *La longueur de la branche d'un nœud u à un instant t est le nombre de nœuds séparant u de la racine de sa branche. La longueur de la branche de u à un instant t sera notée* $\mathrm{len}(u,t)$.

Preuve. Pour tout temps t donné, soit $S'(t)$ l'ensemble des nœuds tels que u hors de V_r étant dans l'état C :

$$S'(t) = \{u \mid (st_u(t) = C) \wedge (u \notin V_r)\}$$

Soit $d'_{\min}(t)$ la distance minimale parmi les nœuds de $S'(t)$ en nombre de sauts par rapport aux racines illégale de leurs branches respectives. Autrement dit, si pour tout nœud u la racine illégale de sa branche est notée r_u alors :

$$d'_{\min}(t) = \min_{u \in S'(t)} \mathrm{len}(u,t)$$

Si $S'(t)$ est vide, posons $d_{\min}(t) = +\infty$.

- **Fini implique borné.** *Si $d'_{\min}(t)$ est fini, alors $d'_{\min}(t)$ est borné supérieurement par le nombre de nœuds n :*
  En effet, à tout instant t la longueur maximale d'une branche est de n donc :

  $$\forall u \in V : \mathrm{len}(u,t) \leqslant n \implies d'_{\min}(t) \leqslant n$$

- **Fonction croissante.** *La fonction $d_{\min}(t)$ est croissante :*
  En effet, faire décroître la valeur de $d'_{\min}(t)$ implique qu'il existe un nœud v qui se rattache à un instant $t' > t$ à une branche enracinée en un nœud r_u via un nouveau parent u se trouvant sur une branche de longueur $\mathrm{len}(u,t) = d'_{\min}(t) - 1$. Or par définition de $d'_{\min}(t)$, tout nœud dont la longueur de branche inférieure à $d'_{\min}(t)$ d'une racine n'est pas dans l'état C et ne pourra donc pas être choisit par v comme parent ($P_{\mathrm{creer}}(v)$, $P_{\mathrm{ameliorer}}(v)$ et $P_{\mathrm{corriger}}(v)$ requièrent tous les trois qu'il existe un voisin u de v tel que $\mathrm{len}(u,t) = d'_{\min}(t) - 1$ et se trouvant dans l'état C).

- **Type de croissance de la fonction.** *Si $d'_{\min}(t)$ est fini alors il existe un instant t' tel que $t' > t$ et $d'_{\min}(t') > d'_{\min}(t)$:*
  Considérons le sous-ensemble $S'_{\min}(t) \subseteq S'(t)$ ayant une longueur de branche de $d'_{\min}(t)$:

  $$S'_{\min}(t) = \left\{u | u \in S'(t) \wedge \mathrm{len}(u,t) = d'_{\min}(t)\right\}$$

  Pour tout nœud $u \in S'_{\min}(t)$, à l'instant t et jusqu'à ce que u applique une règle :

  – soit u vérifie $P_{\mathrm{erreur}}(u)$ et passe dans l'état E ;

– soit u ne vérifie pas $P_{\text{erreur}}(u)$, donc :

$$\exists v \in \Gamma(u) \mid (d_v(t) < d_u(t)) \wedge (\text{st}_v(t) = C)$$

le nœud u vérifie alors $P_{\text{ameliorer}}(u)$ ou $P_{\text{corriger}}(u)$ et passe dans l'état C. Or $u \in S'_{\min}(t)$, il n'existe donc aucun nœud v tel que $\text{len}(v,t) < \text{len}(u't)$ le nœud u aura donc a l'instant t', après choix de son nouveau parent, $\text{len}(u,t') \leqslant \text{len}(u,t) + 1$. Pour résumer, le nœud u se rattache à une branche de longueur strictement supérieure à $d'_{\min}(t)$ ou passe dans l'état E.

Dans ces deux cas, la taille de l'ensemble $S'_{\min}(t)$ diminue, de plus le démon d'exécution est faiblement équitable donc tout nœud de $S'_{\min}(t)$ effectuera son pas de calcul après un temps fini. Il existe donc pour tout instant t tel que $S'_{\min}(t)$ non vide un instant t' auquel $d'_{\min}(t') > d'\min(t)$.

Ainsi il existe un instant t tel que $d'_{\min}(t) = +\infty$, à cet instant tous les nœuds de la composante connexe considérée ne seront plus dans l'état C. Donc, pour toute feuille dans l'état E de cette composante P_{isoler} est vérifié jusqu'à ce qu'il applique R_I. De plus, tout nœud dans l'état I ne vérifie aucun prédicat car tous ses voisins sont soit dans l'état I soit dans l'état E. Le nombre de nœuds dans un état différent de I ou ayant un parent non nul est donc strictement décroissant. Il existe donc un instant t' auquel tout nœud u de la composante connexe sera dans l'état $(\text{st}_u = I) \wedge (\text{parent}_u = \text{NULL})$. ∎

5.7 Conclusion

Cet algorithme pourrait être utilisé dans un contexte où aucune information sur le graphe n'est calculable ou estimable suffisamment souvent pour permettre à des algorithmes tels que ceux présentés dans l'introduction de fonctionner. Par exemple, si la dynamique du réseau est imprévisible et que le diamètre ou le nombre de nœuds du graphe change fréquemment, tout algorithme nécessitant ce type de connaissances échouera alors que l'algorithme présenté dans ce chapitre s'adaptera à ce type de changements.

La principale perspective de ces travaux serait de considérer le modèle de passage de messages ce qui est à priori simple. De plus pour être utilisé dans le cadre du routage compact il serait nécessaire de réaliser cette transition en utilisant une mémoire sous-linéaire et donc sans connaître l'ensemble de ses enfants dans l'arbre construit. Ce dernier point rend la traduction dans le modèle de passage de message difficile et cette contribution serait intéressante en soit. Il serait également intéressant d'analyser la complexité de cet algorithme aussi bien de manière théorique qu'expérimentale et ainsi le comparer à des algorithmes utilisant une connaissance sur le graphe pour converger. La question principale étant, la robustesse de ce nouvel algorithme implique t-elle un gros sacrifice en terme de temps de convergence ou en coût de communication dans une utilisation en pratique. Pour le moment, l'algorithme est présenté pour un graphe non pondéré, mais pourrait être adapté simplement pour fonctionner sur des graphes pondérés.

Caractérisation du nombre d'erreurs dans un réseau dynamique

6

Sommaire

Ce chapitre, tout comme le précédent, s'intéresse aux réseaux, ou systèmes distribués, dynamiques. Dans le cadre du routage vers une destination fixée, chaque nœud stocke un chemin dans le réseau vers cette destination. Le stockage d'un chemin dans sa totalité n'est cependant pas nécessaire si les nœuds se coordonnent pour effectuer une même tâche. Dans notre cas, un simple *conseil* vers un nœud voisin se trouvant sur un plus court chemin vers la destination commune suffit.

La dynamique du réseau ainsi que sa grande taille ont un impact néfaste sur la fiabilité de ces conseils. Après une panne, avant que les informations de routage ne soient mises à jour par le schéma de routage, une période d'instabilité s'installe. Habituellement, les nœuds dans un état instable, délivrant des informations de routage erronées, sont qualifiés de *menteurs*. Ce terme est utilisé bien que les nœuds menteurs n'aient pas nécessairement d'intentions malicieuses. En utilisant des techniques de routage classiques telles que celles présentées dans le chapitre 3, la présence de menteurs défait toute garantie de routage. Par exemple, BGP*, l'algorithme de routage actuellement utilisé sur le réseau des systèmes-autonomes, ne donne aucune garantie de routage lorsque les tables de routage contiennent des erreurs.

Il existe cependant des algorithmes permettant de router en présence de menteurs. C'est ce que montre une série d'articles, [HKK04, HKKK08, HIKN10], concernant la localisation de cibles/objectifs dans un réseau contenant des menteurs. Cette tâche

*Border Gateway Protocol

est équivalente à un routage vers un nœud cible donné. Un premier modèle, introduit par Kranakis et Krizanc [KK99], décrit un algorithme de localisation dans un réseau distribué de topologie anneau ou tore, dans lequel chaque nœud a une probabilité constante d'être un menteur. Un second modèle, plus réaliste, proposé par Hanusse *et al.* [HKK04] définit les menteurs comme un sous-ensemble de $V_\mu \subset V$ de taille μ. Il est de plus considéré dans cet article que l'ensemble V_μ reste inchangé durant la phase de recherche, autrement dit durant le routage d'un paquet. Sur ce modèle, différents algorithmes, génériques ou dédiés à des topologies spécifiques, ont été présentés dans [HKK04, HKKK08, HIKN10]. Les bornes données dans ce domaine d'étude sont typiquement de la forme $\mathcal{O}\big(d + 2^{\mathcal{O}(\mu)}\big)$ pour les graphes de degré borné, avec d la distance entre la source et le nœud destination en nombre de sauts. En outre, pour les chaînes/anneaux et expandeurs l'algorithme générique [HIKN10] garantit un routage en au plus $d + k^{\mathcal{O}(1)}$ sauts. Beaucoup de graphes de terrains, tels que CAIDA possèdent des propriétés "proches" de celles des expandeurs. Dans ces article, une supposition implicite est faite : *le nombre de menteurs est faible*.

L'objectif de ce chapitre est de caractériser, partant d'une configuration sans menteurs, le nombre de menteurs créés par une ou plusieurs modifications atomiques. La modification atomique considérée est la suppression d'une arête dans le graphe représentant le réseau dynamique. On notera μ le nombre de menteurs.

La section 6.2 donne le détail des modèles utilisés pour cette étude. La section 6.4 montre que le nombre de menteurs créé par $\mathcal{M}$ suppressions d'arêtes dans un graphe G de diamètre D, contenant n nœuds et m arêtes est en moyenne inférieur à $\mathcal{M} \cdot \frac{Dn}{m}$. La section 6.5 montre que la borne inférieure sur le nombre de menteurs pour une unique suppression est de $(D-8) \cdot \frac{n}{32m}$ dans un graphe G tel que $m \geqslant n \geqslant 2D$. Pour caractériser la précision de la borne inférieure dans des cas particuliers, des analyses plus précises pour des topologies régulières telles que la grille, les graphes d'Erdős-Rényi ou encore les hyper-cubes sont données en section 6.6. La fin du chapitre aborde quant à elle des expérimentations sur des modèles de graphes en *loi de puissance* ainsi que sur des carte d'Internet de type *CAIDA*. Les résultats des expérimentations sont présentés en section 6.7.

6.1 Travaux connexes

L'influence des changements de topologie dans les graphes est étudiée dans plusieurs travaux. Dans [CG84, SBvL87], il est prouvé que la suppression de $\mathcal{M}$ arêtes dans un graphe de diamètre D induit une multiplication du diamètre par un facteur $\Omega(\mathcal{M}+1)$. La borne inférieure de $(\mathcal{M}+1)D-2\mathcal{M}+2$ est obtenue à l'aide d'une chaîne à laquelle des raccourcis sont ajoutés comme en figure 6.1

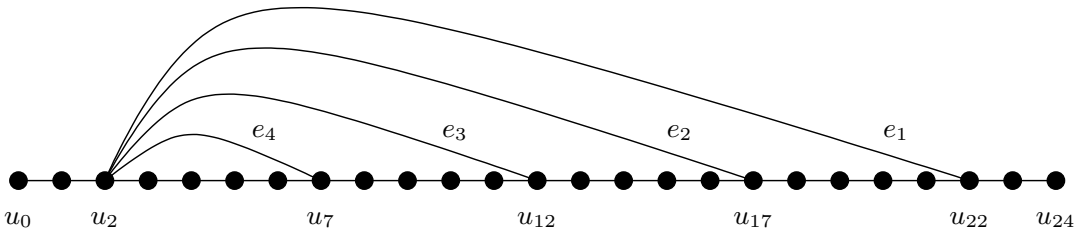

Figure 6.1: Exemple de construction pour la borne supérieure sur le facteur d'augmentation du diamètre après $\mathcal{M} \leqslant 4$ suppressions d'arêtes. Cet exemple est celui donné dans [CG84]. Dans ce graphe G de taille $n = 25$ et de diamètre $D(G) = 6$, la suppression des arêtes e_1, e_2, e_3 et e_4 crée un graphe G' de diamètre $D(G') = n - 1 = 24 = (\mathcal{M}+1) \cdot D(G) - 2\mathcal{M} + 2$. Autrement dit, chaque suppression ajoute environ 6 unités au diamètre courant du graphe.

Les travaux présentés dans ce chapitre sont également liés au calcul du *nœud le plus important* d'un chemin. Nardelli *et al.* mettent dans [NPW03] en exergue le scénario de suppression de sommet créant la plus grande augmentation de distance possible pour une paire de nœuds donnée. Plus précisément, cet article présente un algorithme centralisé qui permet de trouver le nœud le plus important du graphe en temps $\mathcal{O}(m + n \log n)$ avec une mémoire de travail de $\mathcal{O}(m)$.

Les travaux sur les structures *dynamiques* pour le calcul de plus courts chemins ou de distances se basent sur des modèles de graphes dynamiques :

- Le modèle le plus général, les *graphes évolutifs temporels* non contraints, introduits par Ferreira dans [FJ04] : Un graphe évolutif $\mathcal{G}$ est basé sur un graphe statique $G = (V, E)$ appelé *graphe sous-jacent*. L'état du graphe $\mathcal{G}$ à un instant t, noté G_t, est donné par une fonction de présence qui détermine pour chaque arête de G si elle est présente à l'instant t, ainsi qu'une fonction de poids qui détermine son poids à ce même moment. Ainsi, G_t est un sous-graphe de G. De plus, pour deux instants t et t', l'état du réseau peut fortement varier, i.e. le graphe G_t est très différent du graphe $G_{t'}$.

  Ce type de graphes dynamiques non contraints rend cependant la plupart des problèmes algorithmique insolubles. C'est pourquoi différentes sous-classes de ceux-ci existent et rendent calculables certains problèmes classiques. Par exemple, dans un graphe évolutif non contraint, il est impossible d'effectuer une diffusion. En restreignant l'étude aux graphes évolutifs constamment connexes ($\forall G_i \in \mathcal{G}, G_i$ est connexe) la diffusion devient possible en temps au plus n. Un tour d'horizon sur ces différents modèles à été fait par Santoro *et al.* dans [CFQS12].

- Cependant, dans nombre d'études le modèle de graphes considéré a une dynamique bien plus contrainte encore. Le cas standard dans les travaux liés au routage est le cas des *graphes avec pannes*. Intuitivement, il est considéré dans ce modèle que, partant d'un graphe initial, après avoir opéré $\mathcal{M}$ ajouts/suppressions de nœuds/arêtes le graphe reste stable pendant un temps suffisamment long. Par suffisamment long on entend supérieur au temps nécessaire pour effectuer un calcul sur le graphe. Ce temps de stabilité dépend donc de l'algorithme considéré et non du modèle uniquement.

Dans le cas des graphes avec pannes, la solution la plus naïve pour permettre le routage, consiste à recalculer l'ensemble des tables de routages après toute détection de modification topologique. Deux algorithmes centralisés dynamiques effectuant un recalcul global de tous les plus courts chemins à chaque modification sont présentés dans les travaux de Demetrescu et Italiano [DI04] ou Thorup [Tho04]. Ces deux algorithmes sont les plus rapides connus et permettent une mise à jour en un temps amorti de $\mathcal{O}\big(n^2 \operatorname{polylog}(n)\big)$ par modification atomique (en comparaison l'algorithme Bellman-Ford distribué permet de recalculer tous les plus courts chemin en temps $\mathcal{O}(mn)$). Ils sont tous les deux basés sur l'observation faite dans [KKP93], qui statue très sommairement, qu'un plus court chemin est une composition de plusieurs plus courts chemins et donc le fait de connaître un ensemble de plus courts chemin peut aider à en trouver d'autres de façon plus économique. Cependant, dans le modèle de pannes, il s'avère qu'il n'est pas toujours indispensable de recalculer tous les plus courts chemins pour garantir le routage. D'autres études proposent par exemple des structures de données efficaces dédiées aux problèmes de calculs de plus courts chemins ou de distances dans un réseau contenant un nombre d'erreurs donné $\mu = 1$. Ces algorithmes se distinguent en deux grandes catégories, les algorithmes dit *exacts* tels que [DTCR08, BK09] en opposition aux algorithmes d'*approximation à facteur constant* tels que [KB10, CLPR10]. Le facteur d'approximation faisant référence à l'étirement multiplicatif (ou ratio distance proposée, distance réelle dans le cas d'un oracle de distance, i.e. [CLPR10]). Le véritable challenge est de proposer un algorithme tolérant à plus d'une panne ($\mu > 1$). Le résultat le plus général connu est l'*oracle de distance $\mathcal{M}$-sensible* présenté par Chechik *et al.* dans [CLPR10]. Cet algorithme calcule une structure de données de taille $\mathcal{O}\big(\mathcal{M} \cdot s \cdot n^{1+1/s} \log n\big)$ en temps $\mathcal{O}(s \cdot \mathcal{M})$ et permet d'approximer la distance entre toute paire de nœuds à un facteur multiplicatif $\mathcal{O}\Big(\mathcal{M} \cdot \log^{\mathcal{O}(1)} n\Big)$ près.

Dans l'ensemble de ces travaux, les analyses sont faites dans le pire cas, celui d'un *adversaire fort*. La pire séquence de modifications est considérée. Cette vision est très pessimiste et arbitraire, elle ne donne pas une idée macroscopique du comportement de l'algorithme en contexte réel, sans attaque. C'est pourquoi nous considérons dans notre étude un deuxième modèle de fautes/pannes, le modèle de *fautes aléatoires* : toute séquence de $\mathcal{M}$ modifications a la même probabilité d'apparaître. Une estimation du nombre de changement de distances sous ce modèle dans un réseau dynamique, pourrait être utilisée pour analyser le temps de mise à jour d'algorithmes tels que celui de King [Kin99] (section 2.1). Le temps d'actualisation permettant de maintenir un arbre de plus court chemin à jour de ce dernier est de $\mathcal{O}(D \cdot \kappa)$ lorsque $\mathcal{M} = 1$, avec κ nombre de changement de distances à la racine. L'analyse présentée ici permet notamment de donner le temps de mise à jour moyen de cet algorithme centralisé.

6.2 Modèles et observations

Le graphe considéré est non pondéré, non nécessairement connexe. Un nœud spécifique appelé destination sera noté t. Tout nœud $u \in V \setminus \{t\}$ possède un *conseil* vers un nœud voisin :

$$\mathrm{Adv}(u) \in \Gamma(u)$$

Avec $\Gamma(u)$ l'ensemble ouvert des voisins de u, i.e. $u \in \Gamma(u)$. Si le nœud $\mathrm{Adv}(u)$ est sur un plus court chemin de u vers t alors le nœud u est dit *vérace*, sinon il est qualifié de *menteur*. L'ensemble des conseils (arcs) $\mathbb{A}$ de cardinalité $n - 1$, appelé *configuration*,

induit un sous-graphe orienté de G, noté $G_{\mathbb{A}} = (V, \mathbb{A})$. Si la configuration $\mathbb{A}$ ne contient pas de menteur elle est également qualifiée de *configuration vérace*. Pour deux nœuds $(u, v) \in V^2$, il existe un arc (u, v) dans $G_{\mathbb{A}}$, i.e. $(u, v) \in \mathbb{A}$, si et seulement si $\mathrm{Adv}(u) = v$. Lorsque $\mathbb{A}$ est vérace, le graphe $G_{\mathbb{A}}$ est un arbre de plus court chemin enraciné en t couvrant G.

Pour un graphe $G = (V, E)$, l'ensemble des graphes $\mathcal{G}_{G,\mathcal{M}}$ se définit comme l'ensemble des graphes pouvant être obtenus après $\mathcal{M}$ suppressions d'arêtes dans G :

$$\forall \tilde{G} = (V, \tilde{E}) : \left[\tilde{G} \in \mathcal{G}_{G,\mathcal{M}}\right] \iff \left[\left(\tilde{E} \subset E\right) \wedge \left(|\tilde{E}| = |E| - \mathcal{M}\right)\right]$$

Étant donné deux graphes G et $\tilde{G} \in \mathcal{G}_{G,\mathcal{M}}$ et une configuration vérace $\mathbb{A}$, les propriétés observées dans ce chapitre sont :

- $|\mathcal{S}|$ la taille de l'ensemble des nœuds dont la distance à t est différente dans G et $\tilde{G}$;
- et le nombre de menteurs $\mu = \mu_{\tilde{G}}(\mathbb{A})$.

Remarque. Il est possible que $\mathcal{M}$ conseils de la configuration $\mathbb{A}$ doivent être redirigés dans $\tilde{G}$. En effet, du point de vue d'un nœud u, si une arête $\{u, v\}$ telle que $\{u, v\} \in E$ et $(u, v) \in \mathbb{A}$ (i.e. $\mathrm{Adv}(u) = v$) est supprimée alors dans $\tilde{G}$ le nœud u a un conseil qui pointe vers un nœud dont il n'est pas voisin. Le nœud u devra donc choisir un autre conseil dans son nouvel ensemble de voisins. Si l'ensemble des voisins d'un nœud est vide, son conseil pointera vers lui-même ($\mathrm{Adv}(u) = u$). Ce nouveaux choix dépendra du modèle d'adversaire considéré et sera précisé par la suite.

La Figure 6.2 montre un exemple de graphes et de conseils. Après une suppression d'arête, il y a $n - (D + 1)$ nœuds menteurs pointant vers une "impasse" dans la partie extrémale droite ainsi que $D - 1$ nœuds dont la distance à t a changé. Il est à noté que l'un des nœuds a également changé son conseil.

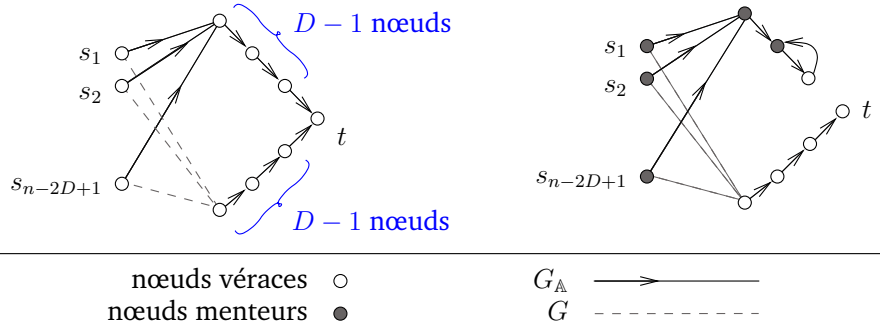

Figure 6.2: Un exemple de suppression d'arête conservant la connexité du graphe et créant $n - (D + 1)$ menteurs dans un graphe de diamètre D.

Il est également possible que le graphe $\tilde{G}$ soit déconnecté. Les nœuds n'appartenant plus à la composante connexe de t deviennent alors menteurs.

Voici pour finir, les deux modèles d'adversaire considérés pour l'étude du nombre de menteurs :

- **Modèle à adversaire (fort)** L'intérêt de ce modèle est d'analyser le pire cas en ce qui concerne la suppression d'arêtes. Pour un graphe $G = (V, E)$ donné, le choix des conseils $\mathbb{A}$ est fait par un adversaire parmi l'ensemble des configurations véraces. Ce même adversaire fait également le choix des $\mathcal{M}$ arêtes à supprimer dans G, autrement dit il choisit l'ensemble d'arêtes $\tilde{E}$ tel que :
$$\left(\tilde{E} \subset E\right) \wedge \left(\left|\tilde{E}\right| = \left|E\right| - \mathcal{M}\right)$$
L'objectif de l'adversaire étant de maximiser le nombre de menteurs $\mu_{\tilde{G}}(\mathbb{A})$. De plus, dans le cas où un nœud u a pour conseil $\mathrm{Adv}(u) = v$ et que l'arête (u, v) n'appartient pas à E' l'adversaire peut choisir le nouveau conseil de u dans $\Gamma(u) \setminus \{u\}$. Rappelons que dans le cas exceptionnel où $\Gamma(u) = \{u\}$ le conseil est $\mathrm{Adv}(u) = u$.

- **Modèle à fautes aléatoires** L'intérêt de ce modèle est d'analyser l'impact en moyenne des suppressions d'arêtes. Pour un graphe $G = (V, E)$ donné, le choix de la configuration de conseils $\mathbb{A}$ est fait de manière aléatoire uniforme dans l'ensemble des configurations véraces. Le graphe $\tilde{G}$ est lui aussi choisi de manière aléatoire uniforme dans $\mathcal{G}_{G,\mathcal{M}}$. De plus, dans le cas où un nœud u a pour conseil $\mathrm{Adv}(u) = v$ et que l'arête (u, v) n'appartient pas à $\tilde{E}$, le nouveau conseil de u est choisit de manière aléatoire uniforme dans $\Gamma(u) \setminus \{u\}$. Rappelons à nouveau que $\Gamma(u) = \{u\} \implies \mathrm{Adv}(u) = u$.

6.3 Résumé des contributions et définitions supplémentaires

6.3.1 Résultats et quelques remarques simples

Les résultats présentés se focalisent principalement sur le modèle à fautes aléatoires, la majorité des analyses sur le modèle à adversaire étant plus simples. Il est néanmoins intéressant de considérer ces deux modèles notamment pour distinguer des cas pathologiques, présentant de forts écarts en fonction du modèle supposé. Plus précisément, le résultat principal est le suivant :

Théorème 6.1. *Considérons un graphe $G = (V, E)$ avec n nœuds, m arêtes et un diamètre D. Pour toute cible, après $\mathcal{M}$ suppression d'arêtes choisies de manière aléatoire uniforme dans E, l'espérance du nombre de menteurs est $\mathbb{E}(\mu) \leqslant \mathcal{M} \cdot \frac{Dn}{m}$. De plus, l'espérance du nombre de changement de distances observés dans G est également inférieur à $\mathbb{E}(|\mathcal{S}|) \leqslant \mathcal{M} \cdot \frac{Dn}{m}$.*

La table Table 6.1 montre les résultats obtenus pour $\mathcal{M} = 1$ dans les deux modèles. Remarquons également que la suppression d'une arête n'implique pas nécessairement la création de menteurs et ce même dans le cas où certains nœuds auraient changé de distance. Par exemple, dans un graphe complet, aucun menteur ne peut être créé en ne supprimant qu'une seule arête (u, v) :

- si $(u, v) \in \mathbb{A}$ alors la distance de u à t de devient 2 et tout nœud voisin de u est plus proche de t ;
- si $(u, v) \notin \mathbb{A}$ alors u n'est pas menteur.

De manière équivalente, des menteurs peuvent apparaître sans qu'il n'y ait de changement de distance induit par la suppression d'arêtes.

Topologie	Adversaire	Faute aléatoire
Graphe de diamètre D	$\Theta(n)$	$\Theta\left(\frac{D \cdot n}{m}\right)$
Grille carrée	$\Theta(\sqrt{n})$	$\Theta(1)$
Graphe Erdős-Rényi	$\frac{n-1}{4} + 1$	$\Theta\left(\frac{1}{n}\right)$
Hypercube	$\log n - 1$	$\Theta\left(\frac{1}{\log n}\right)$

Table 6.1: Nombre de menteurs induits par une unique suppression d'arête.

Pour la famille des graphes de diamètre D, la borne inférieure pour le modèle à adversaire est aisée à atteindre. Il suffit de considérer une chaîne de D nœuds sur lequel une étoile de $n - D$ nœuds est attachée à une des extrémités. Si t est situé à l'autre extrémité de la chaîne, supprimer l'arête adjacente à t crée n menteurs. En se restreignant aux cas où $\tilde{G}$ doit être connexe, il est également possible d'exhiber un graphe simple permettant d'obtenir $\mu = \Omega(n - D - 1)$ comme le montre la figure 6.2.

6.3.2 Définitions supplémentaires

Voici quelques précisions sur les notations utilisées :

- $d_{\tilde{G}}(u, v)$: La distance de u à v dans $\tilde{G}$. Cette notation sera également raccourcie à $\tilde{d}(u, v)$ s'il n'y a pas d'ambiguïté.
- $\mathcal{S}$ (rappel) : L'ensemble des nœuds ayant changé de distance après suppressions d'arêtes :
$$\mathcal{S} = \{u \mid d_{\tilde{G}}(u, t) \neq d_G(u, t)\}$$
- $\Gamma(X)$: L'ensemble des voisins des nœuds de l'ensemble de nœuds X dans G
$$\Gamma(X) = \bigcup_{x \in X} \Gamma(x)$$
- $\mathrm{Adv}^{-1}(X)$: L'ensemble des nœuds conseillant un nœud appartenant à X :
$$\mathrm{Adv}^{-1}(X) = \{u \in V \mid \mathrm{Adv}(u) \in X\}$$

Plusieurs preuves sont basées sur la notion d'arêtes $\{s, t\}$-*déconnectantes* suivante :

Définition 6.1 (Arêtes déconnectantes)**.** *Une arête (u, v) est dite $\{s, t\}$-déconnectante si elle appartient à* tous *les plus courts chemins de s à t.*

La suppression d'une arête $\{s, t\}$-déconnectante implique l'événement :

$$\mathcal{E}_{s,t} \equiv \tilde{d}(s, t) > d(s, t)$$

Dans le cas contraire, il existe un plus court chemin de s à t ne contenant pas l'arête supprimée et donc par définition $\tilde{d}(s, t) = d(s, t)$. L'ensemble des arêtes $\{s, t\}$-déconnectante est noté $\mathcal{C}_{s,t}$. Il s'ensuit donc le lemme suivant :

Lemme 6.1. *La distance de s à t est modifiée par une suppression d'arête si et seulement si cette arête appartient à $\mathcal{C}_{s,t}$.*

Nous allons maintenant nous intéresser à l'effet des suppressions d'arêtes en commençant par donner la relation entre les changements de distances dans le graphe et le nombre de menteurs. Ce résultat sera utilisé pour caractériser le nombre de menteurs créé suite à une ou plusieurs suppressions d'arêtes dans les deux modèles considérés.

Remarquons que le nombre de menteurs et le nombre de changements de distance sont liés :

Lemme 6.2. *Pour un graphe $G = (V, E)$ et une configuration de conseils $\mathbb{A}$ donnés, après $\mathcal{M}$ suppressions d'arêtes, le nombre total de menteurs dans le graphe $\tilde{G} = (V, \tilde{E}) \in \mathcal{G}_{G,\mathcal{M}}$ satisfait :*

$$\begin{aligned} |\mathrm{Adv}^{-1}(\mathcal{S}) \setminus \mathcal{S}| \leqslant \mu &\leqslant |\mathrm{Adv}^{-1}(\mathcal{S})| + |\mathbb{A} \setminus E'| \\ &\leqslant |\Gamma(\mathcal{S})| + \mathcal{M} \end{aligned}$$

Preuve. Pour un nœud $u \in V$ tel que $\mathrm{Adv}(u) = v$. Si $\{u, v\}$ n'a pas été supprimée et $v \notin \mathcal{S}$, alors $u \notin \mathcal{S}$ et $d_{\tilde{G}}(v, t) = d_G(v, t)$ donc $d_{\tilde{G}}(v, t) < d_{\tilde{G}}(u, t)$ et u ne ment pas. Le nœud u ne peut mentir que si $v \in \mathcal{S}$ ou $\{u, v\} \notin \tilde{E}$. Le nombre de menteurs est donc d'au plus :

$$|\mathrm{Adv}^{-1}(\mathcal{S})| + |\mathbb{A} \setminus \tilde{E}|$$

Supposons maintenant que le nœud v ait changé de distance à t. Si u n'a pas changé de distance à t alors $d_{\tilde{G}}(v, t) = d_{\tilde{G}}(v, t) + d'$ avec $d' \geqslant 1$ donc $d_{\tilde{G}}(v, t) \geqslant d_{\tilde{G}}(u, t)$ et u est menteur. Le nombre total de menteurs est donc d'au moins :

$$|\mathrm{Adv}^{-1}(\mathcal{S}) \setminus \mathcal{S}|$$

∎

Ce lemme sera utilisé dans la partie d'étude du nombre de menteurs pour des topologies données, i.e. section 6.6.

6.4 Nombre d'erreurs après $\mathcal{M}$ suppressions d'arêtes

6.4.1 Dans le modèle aléatoire

Cette section endosse la preuve du théorème 6.1.

6.4.1.1 Effet de $\mathcal{M}$ suppressions d'arêtes aléatoires sur les distances

Dans le modèle aléatoire uniforme, toute arête d'un graphe à m arêtes a une probabilité $\frac{\mathcal{M}}{m}$ d'être supprimée. La probabilité de changer de distance est donc bornée pour tout nœud :

Lemme 6.3. *Pour des graphes $G = (V, E)$ et $\tilde{G} = (V, \tilde{E})$ choisit de manière aléatoire uniforme dans $\mathcal{G}_{G,\mathcal{M}}$, la probabilité qu'un nœud $u \in V$ ait une distance différente dans G que dans $\tilde{G}$ est :*

$$\Pr(u \in \mathcal{S}) \leqslant d_G(u, t) \cdot \frac{\mathcal{M}}{m}$$

Preuve. Les distances $d_G(u,t)$ et $d_{\tilde{G}}(u,t)$ sont différentes si et seulement si pour tout plus court chemin $P = \{u_0 = u, u_1, ..., u_{d_G(u,t)} = t\}$ au moins une arête $\{u_i, u_{i+1}\}$ n'appartient pas à $\tilde{E}$. La probabilité d'apparition d'un tel événement pour $\mathcal{M}$ suppressions d'arêtes distinctes choisies de manière aléatoire uniforme dans E est de :

$$\begin{aligned}\Pr\left[d_G(u,t) \neq d_{\tilde{G}}(u,t)\right] &\leqslant d_G(u,t) \cdot \Pr\left[\left\{u_i, u_{i+1} \notin \tilde{E}\right\}\right] \\ &\leqslant d_G(u,t) \cdot \frac{\mathcal{M}}{m}\end{aligned}$$

Donc $\Pr(u \in \mathcal{S}) \leqslant d_G(u,t) \cdot \frac{\mathcal{M}}{m}$. ∎

En bornant $d_G(u,t)$ par D la borne donnée dans le théorème 6.1 sur le nombre de changement de distances après $\mathcal{M}$ suppressions aléatoires peut être obtenue. Notons tout d'abord X_u la variable aléatoire ayant pour valeur 1 si $u \in \mathcal{S}$ et 0 sinon. L'espérance étant linéaire, la taille de l'ensemble $\mathcal{S}$ est bornée comme suit :

$$\begin{aligned}\mathbb{E}\left(|\mathcal{S}|\right) &= \sum_{u \in V} \mathbb{E}\left(X_u\right) \\ &= \sum_{u \in V} \Pr(u \in \mathcal{S})\end{aligned}$$

Et donc l'espérance du nombre de nœuds ayant changé de distance après $\mathcal{M}$ suppressions d'arêtes est :

$$\begin{aligned}\mathbb{E}\left(|\mathcal{S}|\right) &\leqslant \sum_{u \in V} d_G(u,t) \cdot \frac{\mathcal{M}}{m} \\ &\leqslant \frac{\mathcal{M}}{m} \sum_{u \in V} d_G(u,t) \\ &\leqslant \mathcal{M} \cdot \frac{nD}{m} \qquad (6.1)\end{aligned}$$

6.4.1.2 Borne supérieure après $\mathcal{M}$ suppressions d'arêtes aléatoires uniformes

Considérons les graphes $G = (V,E)$ et $\tilde{G} = (V,\tilde{E}) \in \mathcal{G}_{G,\mathcal{M}}$ ainsi que l'ensemble de conseils $\mathbb{A}$.

Lemme 6.4. *Si un nœud $u \in V$ est menteur dans $\tilde{G}$ alors son conseil pointe vers un nœud dont la distance est différente dans G et dans $\tilde{G}$ ou son conseil a changé suite à une suppression. Autrement dit, u ment uniquement si :*

$$u \text{ ment dans } \tilde{G} \implies \left[((u,v) \in \mathbb{A}) \wedge (v \in \mathcal{S})\right] \vee \left[\{u,v\} \notin \tilde{E})\right]$$

Preuve. Dans la configuration de départ, le conseil $(u,v) \in \mathbb{A}$ est vérace, car $\mathbb{A}$ est vérace par hypothèse. Donc $d_G(u,t) = d_G(v,t) + 1$, or v ne change pas de distance après les $\mathcal{M}$ suppressions, donc $d_G(v,t) = d_{\tilde{G}}(v,t)$, de plus le nœud u ne peut pas diminuer sa distance à t. Finalement, $d_{\tilde{G}}(u,t) \leqslant d_{\tilde{G}}(v,t) + 1$, si l'arête $\{u,v\}$ appartient à $\tilde{E}$ alors u et v sont voisins, donc $d_{\tilde{G}}(u,t) = d_{\tilde{G}}(v,t) + 1$ et u n'est pas menteur.

Si l'arête $\{u,v\}$ n'appartient pas à $\tilde{E}$ il est difficile de dire si u sera menteur ou non, considérons donc le pire cas possible et supposons que lorsque cette arête est supprimée le nœud u devient menteur. ∎

Le lemme suivant peut donc être déduit :

Lemme 6.5. *Un nœud $u \in V$ est menteur dans le graphe $\tilde{G}$ avec une probabilité inférieure à :*

$$(d_G(u,t)) \cdot \frac{\mathcal{M}}{m}$$

Preuve. D'après les lemmes 6.3 et 6.4, on peut déduire que pour un nœud donné $u \in V$ ayant pour conseil $\mathrm{Adv}(u) = v$, la probabilité que u soit un menteur dans $\tilde{G}$ se réduit à :

$$\begin{aligned}\Pr(u \text{ ment dans } \tilde{G}) &\leqslant \Pr(v \in \mathcal{S}) + \Pr(\{u,v\} \notin \tilde{E}) \\ &\leqslant d_G(v,t) \cdot \frac{\mathcal{M}}{m} + \frac{\mathcal{M}}{m} \\ &\leqslant (d_G(u,t) - 1) \cdot \frac{\mathcal{M}}{m} + \frac{\mathcal{M}}{m}\end{aligned}$$

Ou encore :

$$\Pr(u \text{ ment dans } \tilde{G}) \leqslant d_G(u,t) \cdot \frac{\mathcal{M}}{m}$$

∎

Notons $X_{s,t}$ la variable aléatoire ayant pour valeur 1 si s ment dans $\tilde{G}$ et 0 sinon. D'après le lemme 6.5, on peut déduire que l'espérance du nombre de menteurs dans $\tilde{G}$ est :

$$\begin{aligned}\mathbb{E}\left(\mu(\tilde{G}, \mathbb{A})\right) &= \mathbb{E}\left(\sum_{s \in V \setminus \{t\}} X_{s,t}\right) \\ &= \sum_{s \in V \setminus \{t\}} \mathbb{E}\left(X_{s,t}\right) \\ &= \sum_{s \in V \setminus \{t\}} \Pr(u \text{ ment dans } \tilde{G}) \\ &\leqslant \sum_{s \in V \setminus \{t\}} d_G(s,t) \cdot \frac{\mathcal{M}}{m} \\ &\leqslant \sum_{s \in V \setminus \{t\}} D \cdot \frac{\mathcal{M}}{m}\end{aligned}$$

Et donc :

$$\mathbb{E}\left(\mu(\tilde{G}, \mathbb{A})\right) \leqslant \mathcal{M} \cdot \frac{Dn}{m}$$

ce qui termine la preuve du théorème 6.1.

6.4.2 Remarque sur la suppression de nœuds

L'espérance du nombre de menteurs créé par la suppression d'un nœud peut être calculée en utilisant les mêmes techniques que pour la suppression d'arêtes. Assez brièvement, la probabilité de changer de distance pour tout nœud à distance d de t après $\mathcal{M}'$ suppression de nœuds peut être borné par $d \cdot \frac{\mathcal{M}'}{n}$. De cette borne peut être calculé la probabilité de mentir qui peut être bornée pour tout nœud par :

$$D \cdot \frac{\mathcal{M}'}{n}$$

L'espérance du nombre de menteurs crée par $\mathcal{M}'$ suppressions de nœuds serait donc :

$$\mathbb{E}(\mu') \leqslant \mathcal{M}' \cdot D$$

6.5 Borne inférieure sur le nombre d'erreurs

Posons maintenant la question sur la précision de cette borne supérieure et calculons pour cela la borne inférieure pour les graphes de diamètre D.

6.5.1 Borne inférieure pour $\mathcal{M} = 1$ dans le modèle à fautes aléatoire.

Théorème 6.2. *Pour tout triplet d'entiers (n, m, D) tels que $m \geqslant n \geqslant 2D \geqslant 20$,*

(1) il existe un graphe $G = (V, E)$ de $n + \mathcal{O}(1)$ nœuds, $\Theta(m)$ arêtes et de diamètre D, pour lequel le nombre de menteurs, après suppression d'une arête choisie de manière aléatoire uniforme dans E, est d'au moins $\frac{(D-8)n}{32m}$.

(2) il existe un graphe $G = (V, E)$ de $\Theta(n)$ nœuds, $\Theta(m)$ arêtes et de diamètre D, pour lequel l'espérance du nombre de changement de distances, après une suppression aléatoire uniforme d'arête, est de $\Omega(\frac{Dn}{m})$.

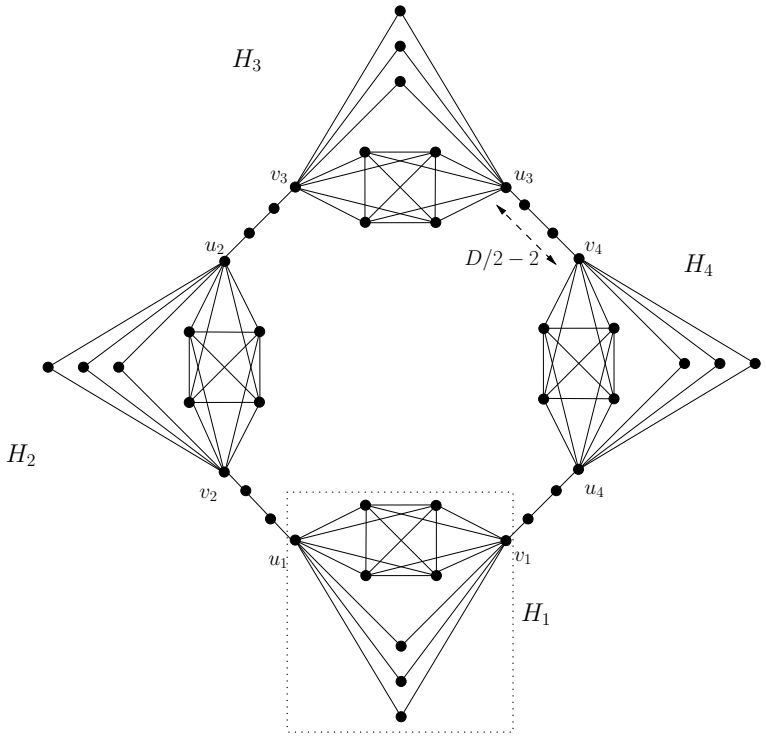

Figure 6.3: Un graphe de diamètre D pour lequel la borne supérieure pour une suppression d'arête est atteinte à un facteur multiplicatif constant près.

Preuve. Considérons le graphe G présenté en figure 6.3 et construit comme suit. Le graphe G est composé de quatre sous-graphes H_1, H_2, H_3 et H_4. Chacun de ces graphes H_i est constitué d'une clique de taille r et d'un stable de taille r' dont les nœuds sont tous liés à deux nœuds u_i et v_i. Les graphes H_i ont un diamètre de 2,

$r + r' + 2$ nœuds et $\frac{r(r-1)}{2} + 2(r + r')$ arêtes. Les graphes H_i sont connectés entre eux pour former le graphe G. Tout nœud u_i est attaché au nœud v_j avec $j = (i + 1) \mod 4$, par un chemin de longueur $D/2 - 2$. Ainsi, le graphe G est en quelque sorte un cycle de diamètre D dans lequel quatre des nœuds sont remplacés par les graphes H_i.

Configurons $r = \left\lceil \sqrt{\frac{m-D}{2}} \right\rceil$ et $r' = \left\lceil \frac{n-D}{4} - r \right\rceil$. Dans cette configuration, G a $n + \mathcal{O}(1)$ nœuds ainsi que $\Theta(m)$ arêtes. Sans perte de généralités supposons que la cible t se trouve entre u_1 et v_2 ou appartient à H_1. Dans le premier cas, tous les nœuds de H_3 (excluant v_3 et potentiellement u_3) ont leur conseil qui pointe vers v_3. La probabilité de supprimer de façon aléatoire uniforme une arête appartenant au chemin allant de u_2 à v_3 est de $p = \frac{D-8}{2m}$. L'espérance du nombre de menteurs/changements de distances créés est d'au moins $p(r + r') \geqslant \frac{(D-8)n}{16m}$.

Pour le second cas, tout nœud de H_3 à l'exclusion de u_3 ou de v_3 peut pointer de façon arbitraire vers u_3 ou v_3. Considérons celui de ces deux nœuds pointé par la majorité des nœuds de H_3. Si v_3 (respectivement u_3) est choisit, alors la probabilité p correspond à la probabilité qu'une arête du chemin allant de u_2 à v_3 (respectivement de v_3 à u_4) soit supprimée. L'espérance du nombre de menteurs est alors d'au moins $p(\frac{r+r'}{2}) \geqslant \frac{(D-8)n}{32m}$.

Dans ce deuxième cas, dans l'objectif d'avoir une borne similaire pour l'espérance du nombre de changements de distance, il est nécessaire de légèrement modifier les graphes H_i en substituant chaque nœud des stables par une arête liant deux nœuds. La taille des stables en nombre de nœuds devient alors $2r'$, chaque graphe H_i a donc $r + 2r'$ nœuds et $\frac{r(r-1)}{2} + 2r + 3r'$ arêtes. En considérant uniquement les changements de distance entre les r' nœuds de ce nouvel ensemble la borne donnée pour l'espérance du nombre de menteurs peut être atteinte. Cependant il est nécessaire pour cela que $r' = \Theta(n)$ et donc que le graphe G ait $\Theta(n)$ nœuds ($|V| = 2n$ est suffisant). ∎

6.6 Analyse pour quelques topologies régulières

Cette section aborde la précision de ces bornes génériques sur le nombre de menteurs créés pour une unique suppression d'arête pour quelques topologies particulières.

6.6.1 Graphes du modèle Erdős-Rényi

Proposition 6.1. *Dans un graphe Erdős-Rényi avec $p = 1/2$ et dans le modèle à fautes aléatoires, le nombre de menteurs est avec grande probabilité :*

$$\mathbb{E}(\mu) = \frac{2}{n-1} + \frac{1}{(n-1)2^{(n/2)-1}} = \frac{2}{n-1} + \mathcal{O}\left(\frac{1}{n}\right)$$

Preuve. Quel que soit la position du nœud cible t, dans un graphe Erdős-Rényi avec $p = 1/2$:

- Tout nœud s à distance 1 de t a $\mathcal{C}_{s,t} = \{\{s, t\}\}$;
- Pour un nœud s à distance 2 de t, avec probabilité $(1/2)^{(n/2)-1}$, un seul de ses voisins appartient à l'ensemble des nœuds à distance 1 de t et donc $|\mathcal{C}_{s,t}| = 2$. Autrement dit, avec probabilité $1 - (1/2)^{(n/2)-1}$ il existe au moins deux chemins allant de s à t.

- De plus, tout nœud à distance 2 peut mentir si son conseil est supprimé (il l'est avec probabilité $1/m$).

donc l'espérance du nombre de menteurs est avec forte probabilité de :

$$\begin{aligned}\mathbb{E}(\mu) &= \frac{n}{2m} + \frac{n}{2m} \cdot (1/2)^{(n/2)-1} + \frac{n}{2m} \\ &= \frac{2}{n-1} + \frac{1}{(n-1)2^{(n/2)-1}}\end{aligned}$$

∎

Remarque. En utilisant le théorème 6.1 on obtient une espérance du nombre de menteurs, pour $p = 1/2$, de :

$$\mathcal{M} \cdot \frac{Dn}{m} = \frac{2n}{n(n-1)/2} = \frac{4}{n-1}$$

Proposition 6.2. *Dans un graphe Erdős-Rényi avec $p = 1/2$ et dans le modèle à adversaire, le nombre de menteurs est avec grande probabilité de*

$$\mu = \frac{n-1}{4} + 1$$

Preuve. Tout nœud s à distance 1 de t ment uniquement si l'arête $\{s,t\}$ est supprimée. Il n'est donc pas intéressant pour l'adversaire de supprimer une de celles-ci dans le but de faire mentir le nœud s. Cependant il est possible de faire mentir tous les nœuds à distance 2 en une seule suppression d'arête, pour cela il suffit de faire pointer tous les conseils des nœuds à distance 2 vers le même nœud. Ainsi, le nombre de menteurs créé est égal au nombre de voisins du nœud de plus haut degré parmi les voisins de t. Si $p = 1/2$ et avec le nœud x celui ayant le plus de voisins à distance 2 de t, le nombre de menteurs est alors, avec grande probabilité de

$$\Big|\Gamma(x) \cap \big\{y \mid d(y,t) = 2\big\}\Big| + 1 = \frac{|\Gamma(x)|}{2} + 1 = \frac{n-1}{4} + 1$$

∎

6.6.2 Grille carrée $\sqrt{n} \cdot \sqrt{n}$

Proposition 6.3. *Dans une grille carrée, avec le modèle à fautes aléatoires l'espérance du nombre de menteurs est*

$$\mathbb{E}(\mu) \leqslant \frac{5n-4}{m}$$

Preuve. Dans une grille, tout nœud s appartenant à la même ligne que t (respectivement colonne) a $|\mathcal{C}_{s,t}| = d(s,t)$ et tout autre nœud a $\mathcal{C}_{s,t} = \varnothing$. La distance moyenne entre deux nœuds appartenant à un même chemin de longueur $\sqrt{n}$ est de $\sqrt{n}/4$, donc :

$$\sum_{s \in V} |\mathcal{C}_{s,t}| = \sum_{s \in \text{ligne de } t} d(s,t) + \sum_{s \in \text{colonne de } t} d(s,t) = \frac{2n}{8}$$

la probabilité de supprimer une telle arête est donc de $\frac{2n}{8m}$. Or chaque nœud a en moyenne un conseil pointant vers lui, donc, d'après le lemme 6.2 :

$$\mathbb{E}(\mu) \leqslant \frac{2n}{8m} + \frac{n-1}{m} \leqslant \frac{5n-4}{m} \quad \left(\leqslant \frac{5}{2}\right)$$

∎

Proposition 6.4. *Dans une grille, en considérant le modèle à adversaire, le nombre de menteurs est :*

$$\mu \leqslant 2(\sqrt{n}-1)+1$$

Preuve. Dans une grille, toutes les arêtes appartenant à la même ligne ou colonne que t sont disconnectantes et toutes les autres ne le sont pas. Pour maximiser le nombre de menteurs, t doit être placé par l'adversaire sur un des bords de la grille et l'arête supprimée doit être une de celles qui lie t au voisin n'appartenant pas à ce bord. D'après le lemme 6.2 le nombre de menteurs est alors :

$$\mu \leqslant |\mathrm{Adv}^{-1}(\mathcal{S})| + \mathcal{M} \leqslant |\Gamma(\mathcal{S})| + 1 \leqslant 2(\sqrt{n}-1)+1$$

∎

6.6.3 Hypercube de dimension Δ

Dans un hypercube, seuls les voisins de t risquent de changer de distance :

$$\bigcup_{s\in V} \mathcal{C}_{s,t} = \{\{x,t\} \in E \mid x \in \Gamma(t)\}$$

donc

$$\sum_{s\in V\setminus\{t\}} |\mathcal{C}_{s,t}| = |\Gamma(t)| = \log_\Delta(n)$$

Lemme 6.6. *Dans un hypercube de dimension Δ, dans le modèle à adversaire fort, le nombre de menteurs est :*

$$\mu = \Delta - 1$$

Preuve. Tout voisin x de t peut être choisi par l'adversaire, tous les voisins de x sont alors des menteurs et donc :

$$\mu = |\Gamma(x)\setminus\{t\}| = \log_\Delta(n) - 1 = \Delta - 1$$

∎

Lemme 6.7. *Dans un hypercube de dimension Δ, dans le modèle à fautes aléatoires, l'espérance du nombre de menteurs est :*

$$\mathbb{E}(\mu) \simeq \frac{\Delta-1}{2^\Delta} + \frac{1}{2\Delta}$$

Preuve. L'ensemble des nœuds pouvant potentiellement devenir menteurs suite à un changement de distance est équivalent à l'ensemble des nœuds à distance 2 de t. La taille de cet ensemble est donc de :

$$\binom{\Delta}{2}$$

Chacun des nœuds de cet ensemble pointe vers un des voisins de t et $m = 2^{\Delta-1}$ le nombre de menteurs induit par un changement de distance est donc :

$$\begin{aligned}
\Pr(\text{supprimer une arête } \{x,t\}) \times \frac{|\mathrm{Adv}^{-1}(\Gamma(x))|}{\Delta} &= \frac{\Delta}{m} \times \frac{\binom{\Delta}{2}}{\Delta} \\
&= \frac{\Delta(\Delta-1)}{2m} \\
&= \frac{\Delta-1}{2^\Delta}
\end{aligned}$$

De plus le nombre d'arêtes est de $m = \Delta \cdot n$, la probabilité de supprimer un conseil est donc de $\frac{n-1}{2(\Delta \cdot n)}$. L'espérance du nombre total de menteurs est donc de :

$$\mathbb{E}(\mu) = \frac{\Delta - 1}{2^{\Delta}} + \frac{n-1}{2(\Delta \cdot n)} \simeq \frac{\Delta - 1}{2^{\Delta}} + \frac{1}{2\Delta}$$

∎

6.7 Expérimentations

Cette section caractérise le nombre de menteurs obtenus après $\mathcal{M}$ suppressions d'arêtes dans le modèle aléatoire uniforme. Dans toutes les expérimentations, le processus suivant est itéré plusieurs fois et $\sqrt{n}$ arêtes sont supprimées successivement :

1. Calcul des distances avant suppression ;
2. Calcul d'une configuration vérace ;
3. Après chaque suppression d'arête :
   a) Calcul des distances et, si l'arête était utilisée par un conseil, redirection du conseil vers un voisin choisi de manière aléatoire uniforme.
   b) Comptabilisation du nombre de menteurs.

Les courbes présentées par la suite, montrent la relation entre le nombre de suppressions d'arêtes et :

1. le nombre de menteurs créés au total ;
2. le nombre de menteurs créés suite à une redirection de conseil ;
3. et le nombre de menteurs créé par déconnexion du graphe, autrement dit le nombre de nœud n'appartenant à la composante du nœud cible t.

Le nombre total de menteurs étant la somme des deux autres items 2) et 3) ajoutée au nombre de menteurs créés par un changement de distance, sans redirection de conseil. Remarquons que ce sont les seules façons qu'a un nœud de mentir.

Remarques sur les donnés et expérimentations. Les limitations de mémoire et de temps imposent de travailler sur des graphes ne possédant pas trop d'arêtes (de l'ordre de $100\ 000$) et pour un nombre d'itérations raisonnable (de l'ordre du millier).

6.7.1 Graphes réguliers

Hypercube. Pour un graphe hypercube de dimension 12, sur 2^6 itération d'expérience, la grande majorité des menteurs proviennent de redirection de conseils et le graphe n'est jamais déconnecté durant les expérimentations. En moyenne, sur les 2^6 itérations et pour 2^6 arrêtes supprimées par itération, le nombre de menteurs créés par un changement de distance est de 0.14 (3.03 menteurs au total dont 2.89 par un changement de conseil, soit $3.03 - 2.89$ menteurs causés par des changements de distance).

Erdős-Rényi. La forte densité des graphes aléatoire du modèle Erdős-Rényi, rend les expériences difficiles, pour ce modèle de graphe le nombre de menteurs créés par des suppressions d'arêtes est trop faible pour pouvoir être observé sur des graphes de taille significative.

Grille carrée $2^6, 2^6$ et UDG Dans une grille, une suppression a une faible probabilité de créer des menteurs, cependant si une suppression crée des menteurs elle en crée de l'ordre de $\sqrt{n}$. C'est pourquoi, si le nombre d'itérations de l'expérience est trop faible la courbe du nombre de menteurs en fonction du nombre de suppressions n'est pas représentative. Le nombre d'itérations est arbitrairement fixé à n. Les résultats sont présentés en figure 6.4. Ces expérimentations montrent que pour un nombre de suppressions $\mathcal{M} = \mathcal{O}(n)$, le nombre de menteurs créés est constant et est d'environ $1/2$ par suppression. Ce qui confirme pour la grille l'observation faite par le théorème 6.1 en ce qui concerne la forme de cette courbe. Cependant comme le montrait déjà la borne précise calculée pour le nombre menteurs dans une grille, sur cette topologie, la pente calculée théoriquement est très loin de la réalité. En effet, en appliquant directement le théorème 6.1, le nombre de menteurs par suppression d'arête pour une grille est de

$$\mathbb{E}(\mu) \leqslant \frac{Dn}{m} \leqslant 2^7$$

Cette différence vient de la supposition initiale utilisée et nécessaire à la preuve du théorème 6.1 : "Pour tout couple de nœuds, il n'existe qu'un plus court chemin.". Or cette supposition est très loin d'être vraie pour une grille ce qui explique la différence observée.

Le graphe UDG avec un rayon fixé à 0.028, il contient 4096 nœuds et 8045 arêtes, l'excentricité maximale de la destination sur l'ensemble des itérations est de 65. Les résultats obtenus sont très proches de ceux de la grille et sont présentés en figure 6.5.

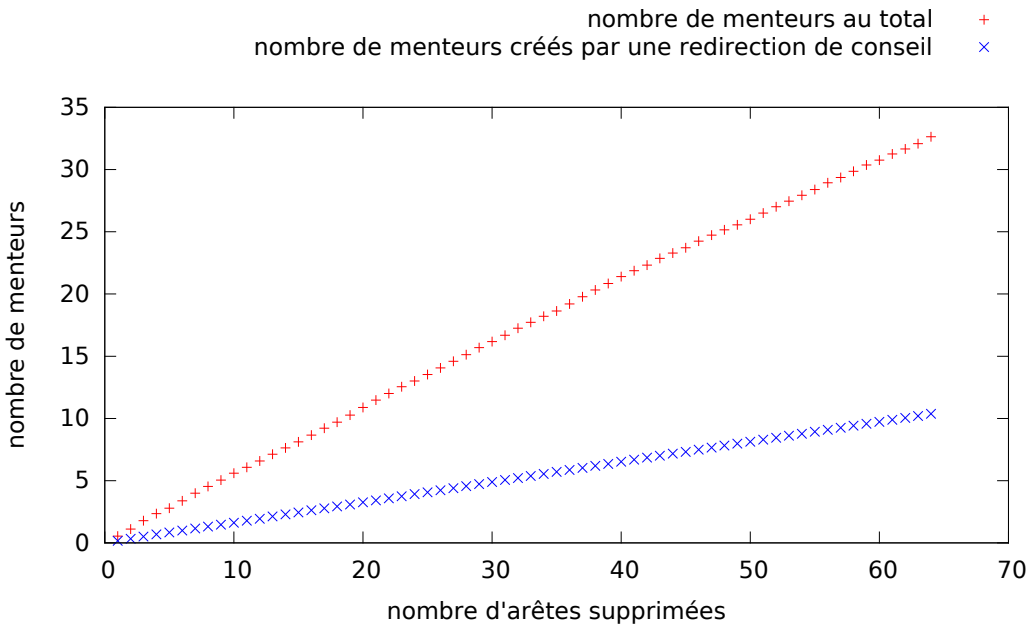

Figure 6.4: Nombre de menteurs obtenus sur une grille carrée de $2^6 \times 2^6$ nœuds pour 1 à 2^6 suppressions d'arêtes. Les points extrémaux pour le nombre de menteurs total sont $(1, 0.55)$ et $(64, 32.63)$, le nombre de menteurs constaté suit donc $\mathbb{E}(\mu) \simeq \frac{\mathcal{M}}{2}$ lorsque la borne précise pour une suppression donnait $\mathbb{E}(\mu) \leqslant \frac{5\mathcal{M}}{2}$ pour n grand.

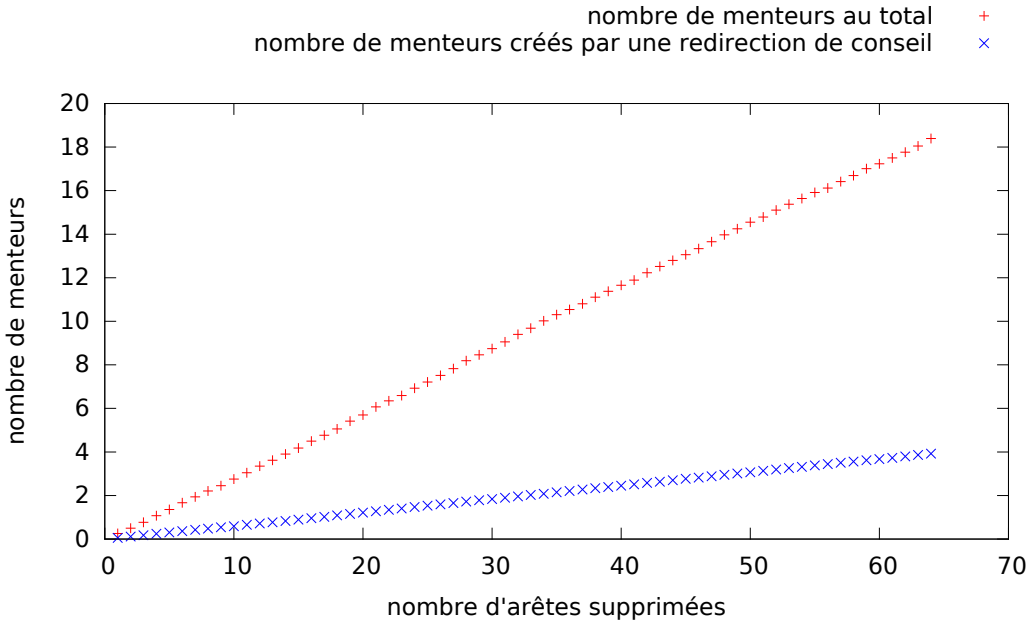

Figure 6.5: Nombre de menteurs obtenu sur un UDG de 2^{12} nœuds pour 1 à 2^6 suppressions d'arêtes. Les points extrémaux pour le nombre de menteurs total sont $(1, 43.3)$ et $(64, 72.49)$.

6.7.2 Graphes de terrain

GLP. Pour les paramétrage de GLP donnés dans le chapitre modèle, le graphe généré a 4096 nœuds et 10310 arêtes, l'excentricité maximale du nœud destination sur l'ensemble des itérations est de 7. En utilisant le théorème 6.1, l'espérance du nombre de menteurs peut être bornée comme suit :

$$\mathbb{E}(\mu) \leqslant \mathcal{M} \cdot \frac{Dn}{m} \leqslant \mathcal{M} \cdot \frac{28672}{10310} \leqslant \mathcal{M} \cdot 2.781$$

En pratique la pente observée est d'environ $1/4$, soit une erreur d'un facteur proche de 11. Les paliers observés sur la figure 6.6 proviennent de déconnexions d'une partie des nœuds du graphe. La figure 6.7 permet d'apprécier la corrélation entre le nombre de menteurs et le nombre de nœuds ne faisant pas partie de la composante de la cible. Pour exacerber ce phénomène, il est nécessaire d'augmenter le nombre d'itérations, il sera fixé à 4096. En effectuant plus d'itérations, il serait possible de réduire ces effets de paliers et d'approcher plus fidèlement l'espérance réelle.

CAIDA. Les résultats sont présentés en figure 6.8. En utilisant le théorème 6.1, le nombre de menteurs peut être estimé à :

$$\mathbb{E}(\mu) \leqslant \mathcal{M} \cdot \frac{Dn}{m} \leqslant \mathcal{M} \cdot \frac{173060}{35547} \leqslant \mathcal{M} \cdot 4.9$$

autrement dit la pente théorique de la droite est 4.9 contre 0.4 en pratique, soit une erreur d'un facteur de 12.25. De même que pour le graphe GLP, les paliers dans la courbe indiquent la séparation d'une partie des nœuds de la composante connexe de la cible. Dans une des 500 itérations il arrive même que la cible se retrouve seule dans sa composante connexe, le nombre de menteurs pour cette itération est alors de $n-1$, ce qui implique un très gros palier (un saut de $n - 1/500 = 34$ menteurs au moins). Le palier observé en pratique entre la 127ème suppression et la 128ème est de 34.67 ($= 51.776 - 17.106$). Ce phénomène de déconnexion est dans certains graphes encore plus visible par exemple dans un anneau le nombre de nœuds n'appartenant pas à la composante connexe du nœud cible est une fonction exponentiel du nombre de suppressions. Un exemple pour l'anneau est donné en figure 6.9.

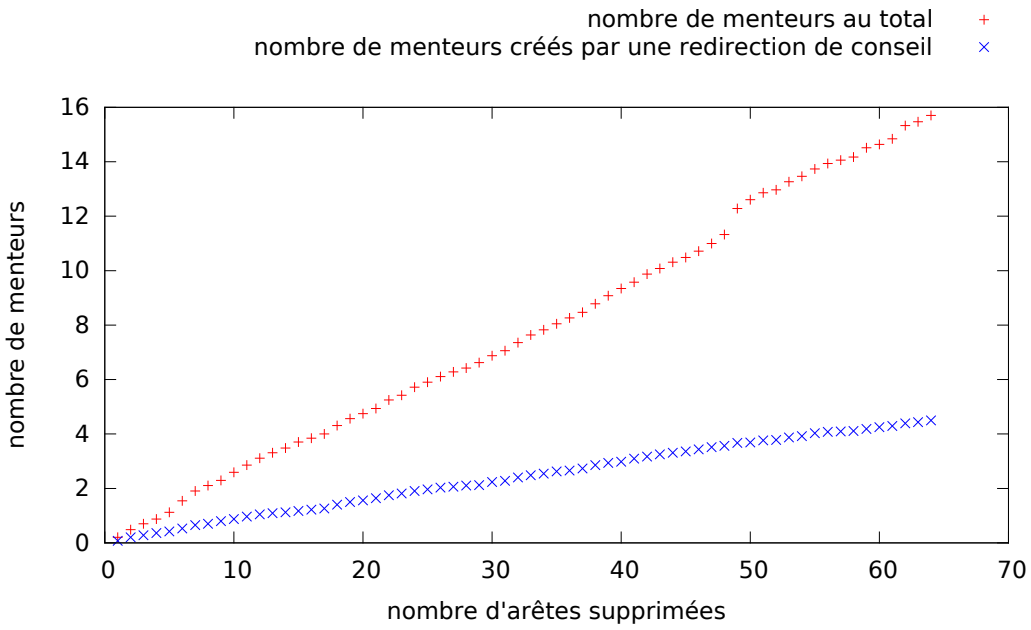

Figure 6.6: Nombre de menteurs obtenu pour un graphe GLP de 4096 nœuds et 10310 arêtes pour 1 à $\sqrt{n} = 64$ suppressions d'arêtes et 64 itérations. Les points extrémaux pour le nombre total de menteurs sont $(1, 0.2)$ et $(64, 15.7)$

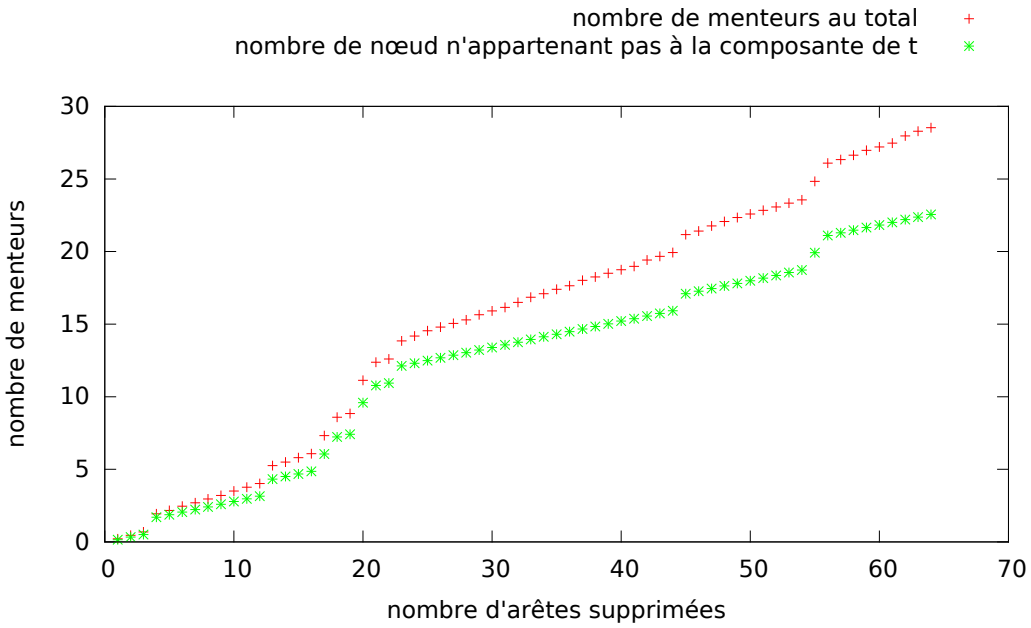

Figure 6.7: Nombre de menteurs obtenu pour un graphe GLP de 4096 nœuds et 10310 arêtes pour 1 à $\sqrt{n} = 64$ suppressions d'arêtes et 4096 itérations. Les points extrémaux pour le nombre total de menteurs sont $(1, 0.21)$ et $(64, 28.53)$

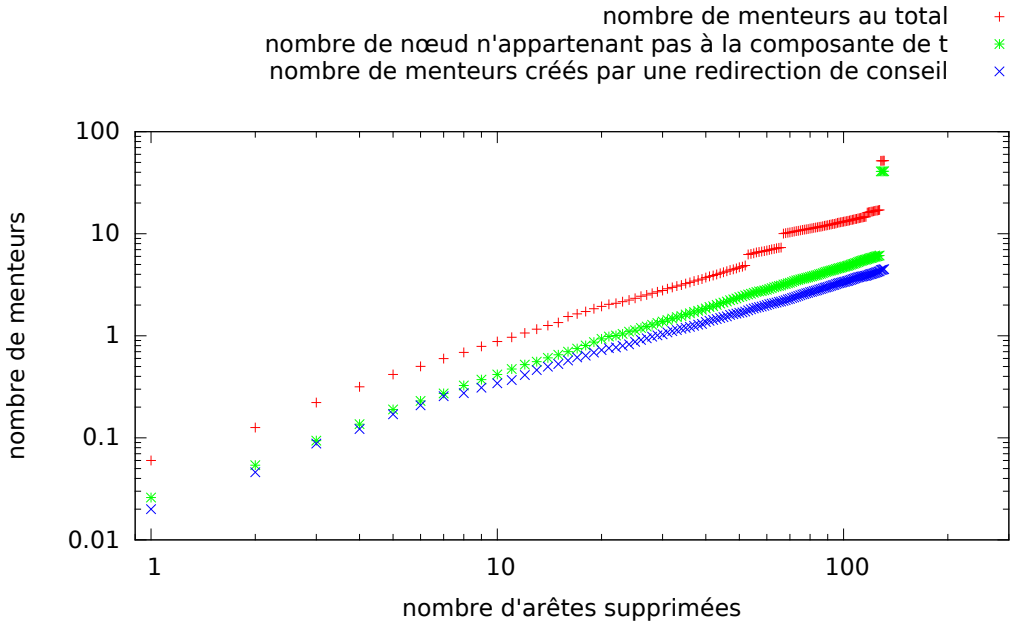

Figure 6.8: Nombre de menteurs obtenu sur une carte CAIDA de 17306 nœuds et 35547 arêtes pour 1 à $\sqrt{n} \simeq 131$ suppressions d'arêtes et 500 itérations. Les points extrémaux pour le nombre total de menteurs sont $(1, 0.05)$ et $(131, 52.066)$. Sur les 52 menteurs, 40 sont dus à une déconnexion du graphe.

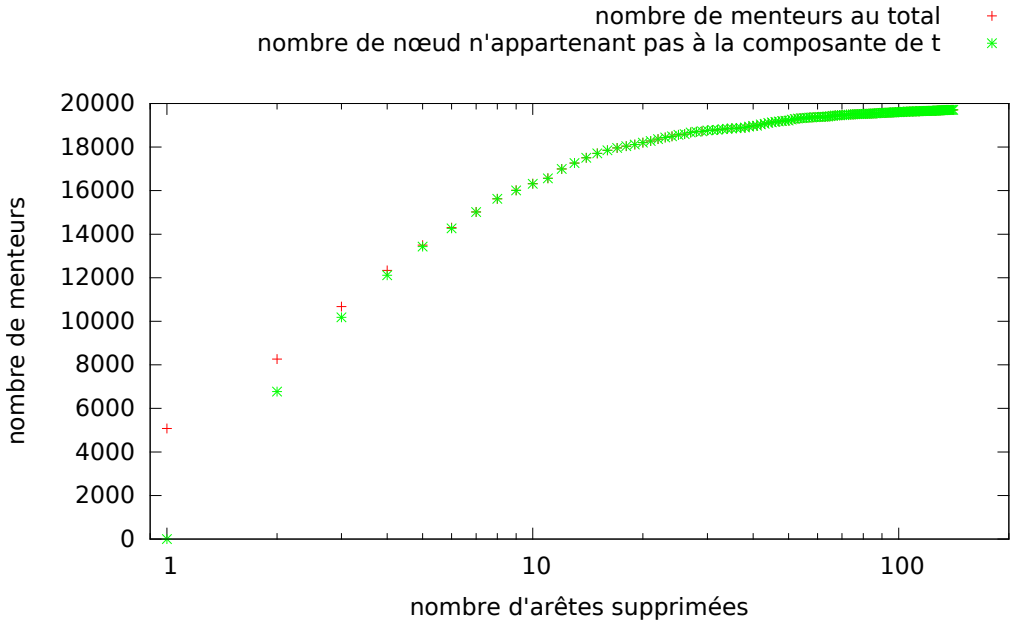

Figure 6.9: Les déconnexions, nombreuses, l'exemple de l'anneau. Pour un graphe de 20000 nœuds et $\mathcal{M} \in [1, \sqrt{n}]$.

6.8 Conclusion

Observations sur les résultats. Les résultats obtenus dans les expériences confirment que le nombre de menteurs est linéaire en fonction du nombres d'arêtes supprimées dans un graphe comme le suggérait la borne supérieure du théorème. Cependant les expérimentations montrent également que la pente de la borne supérieure n'est pas très fine pour les graphes considérés. Ce manque de finesse provenant du nombre de la multiplicité des plus courts chemins.

Il serait intéressant de trouver une borne inférieure sur le nombre de menteurs pour pouvoir caractériser plus finement la vulnérabilité des graphes aux pannes en toute généralité. Une première remarque sur cette voie est de constater que dans un graphe ayant $m > n$, en toute généralité il est possible de supprimer $(m - n)$ arêtes sans créer un seul menteur.

Nombre de menteurs après ajout d'une arête/d'un nœud. Cette relation semble nettement plus difficile à caractériser. Le nombre de changement de distance pourrait être estimé par la borne donnée pour l'ajout en considérant l'ajout d'une arête dans un graphe comme la suppression d'une arête dans son graphe complémentaire. Puis en utilisant le nombre de changements de distances ainsi estimé en addition aux bornes données par le lemme 6.2 il serait possible d'estimer le nombre de menteurs. Cependant, caractériser le nombre de menteurs créés par des ajouts de façon plus précise semble nettement plus complexe. Il serait intéressant de se pencher sur cette question, l'ajout d'arête ou de nœud étant un phénomène fréquent, même dans les réseaux considérés comme "statiques" (nœuds non mobiles) comme le graphe des systèmes-autonomes qui grandit de jour en jour.

Routage avec étirement borné et suppressions d'arêtes. Pour un algorithme de routage basé sur du routage dans des arbres de plus court chemin et ayant un étirement multiplicatif de s, est-il possible d'estimer/borner la proportion de routages atteignant la cible en fonction de s et de $\mathcal{M}$?

Conclusions et perspectives

Concevoir un algorithme distribué efficace. Comme nous l'avons vu dans les expérimentations, utiliser un algorithme centralisé ayant une faible complexité mémoire comme base pour un algorithme distribué semble être une bonne stratégie pour réduire les coûts de communication. Cela a permis, pour tous les algorithmes étudiés dans le chapitre 4, d'obtenir des complexités de communication sous-quadratiques. Néanmoins cela n'est pas toujours aussi simple. Par exemple, pour le cas de l'algorithme [AGM+08], la partie de l'algorithme permettant d'obtenir un étirement de 3 requiert également des tables de routage de taille sous-linéaire. Il parait cependant très difficile d'adapter cette technique dans le modèle distribué en conservant un coût de communication faible. Obtenir un étirement de 3 pourrait être une perspective intéressante en soit. Plus précisément la question que nous nous posons est la suivante :

Q : Est-il possible de concevoir un algorithme distribué **avec indépendance des noms**, et ayant un compromis étirement/mémoire optimal ($3/\mathcal{O}(\sqrt{n})$) ainsi qu'un coût de communication $\mathcal{O}(n^2)$ pour la famille des graphes sans-échelle dans le modèle $\mathcal{LOCAL}$?

La difficulté d'implémentation des techniques centralisées proposées dans [AGM+08] vient du fait que la structure calculée pour obtenir un étirement de 3 nécessite d'obtenir beaucoup d'informations qui ne seront pas stockées.

Obtenir un étirement de 3 avec les même contraintes dans le modèle étiqueté parait cependant simple. Il suffirait pour cela de modifier l'algorithme LO proposé dans le chapitre 4 pour le rendre étiqueté et ainsi avoir des entrées de taille $\mathcal{O}(n)$ et un coût de communication sous-quadratique pour les graphes sans-échelle.

Enfin et plus généralement, obtenir une complexité de communication sous-quadratique avec un étirement borné pour des graphes de diamètre élevé serait intéressant.

Un algorithme compact dans un environnement dynamique. Comme nous l'avons vu dans le chapitre 6 le nombre d'entrées erronées apparaissant suite à une suppression est proportionnel au diamètre. L'algorithme DCR, proposé dans le chapitre 3, utilise des arbres couvrant moins de nœuds qu'un algorithme de plus court chemin. Suite à des suppressions de nœuds ou d'arêtes, le nombre d'entrées non à jour dans les tables de DCR serait donc a priori plus faible que pour un algorithme de plus court chemin tel que DVECTOR. Il y a donc un bon espoir quant au fait de proposer un jour un algorithme dynamique générant moins de messages de contrôle que DVECTOR, voir peut être même moins que tout algorithme de plus court chemin.

Dans cet optique, l'algorithme de calcul d'arbres auto-stabilisant proposé dans le chapitre 5 pourrait être utilisé, cela permettrait par la même occasion d'éliminer proprement le problème du comptage à l'infini. Pour cela il serait néanmoins nécessaire de proposer une version dans un modèle à passage de messages de cet algorithme et surtout ayant un temps de convergence de l'ordre du diamètre, ce qui n'est pas le cas dans la version décrite dans le chapitre 5. Il existe cependant un algorithme auto-stabilisant, décrit dans [KK13] qui effectue le même calcul avec un temps de convergence de D, mais dans un environnement synchrone. Un dernier aspect important serait de proposer un algorithme, utilisant une mémoire sous-linéaire pour conserver la nature compacte du schéma de routage.

Économiser la mémoire et les coût de communication pour économiser de l'énergie. Dans les réseaux de capteurs, les entités ont une durée de vie limitée. Cette durée de vie est largement dépendante de la quantité totale d'énergie utilisée par le capteur. Dans le but de prolonger la longévité des capteurs de nombreuses études s'intéressent au routage *énergétiquement-efficace*. La consommation d'énergie est liée à deux processus, l'échange de messages et les calculs locaux. Comme le montre le tour d'horizon des algorithmes de routage énergétiquement-efficace [JSAC01], il existe un compromis entre ces deux forme de consommation énergétique. Les algorithmes de routage compact distribués pourraient permettre d'économiser sur les deux plans, la réduction de la taille des tables de routage pouvant permettre de réduire les coûts de calculs locaux. Et comme nous avons pu le constater, les coûts de communication de ces algorithmes peuvent être significativement meilleurs que des algorithmes de routage de plus court chemin classique.

Router en présence d'erreurs. Dans le but de réduire encore plus le nombre de messages de contrôle émis dans le cas dynamique, il serait intéressant de considérer des algorithmes de routages tolérants aux pannes/erreurs. En effet, même si un algorithme tolère la dynamique du réseau, la garantie que celui-ci donne est que si il existe une période de stabilité suffisamment longue, alors les tables de routages arriveront dans un état légitime. Cependant, dans un cas d'utilisation réel, les requêtes de routages sont émises en continu. Or il n'y a une garantie de succès des routages que si toutes les tables de routage sont dans un état légitime. Utiliser un algorithme tolérant aux erreurs pourrait permettre d'avoir une garantie de routage à tout instant. Malheureusement il n'existe pas aujourd'hui d'algorithme de routage tolérant un nombre arbitraire d'erreurs quel que soit le graphe considéré. Des travaux préliminaires non présentés dans cette thèse laissent cependant entrevoir la possibilité de router avec un étirement additif borné supérieurement par le temps d'exploration d'une boule de rayon k plus k^2, avec k le nombre d'erreurs présentes dans le graphe au moment du routage. Il serait également intéressant d'analyser ce type d'algorithmes sur des topologies de graphes sans-échelle.

Les algorithmes tolérants au erreurs proposent des garanties d'étirement en fonction du nombre d'erreurs, il serait donc possible de ne pas mettre à jour les tables de routage tant que les routes ne sont pas trop étirées. Autrement dit un routeur pourrait chercher à mettre à jour sa table lorsqu'il constate que les routes sont trop dégradées. Ainsi le nombre de messages émis pour garder les tables "à jour" serait plus faible.

Router lorsque les nœuds agissent de manière égoïste. Cet un aspect important des algorithmes de routage qui n'a pas du tout été abordé durant cette thèse. Cette perspective est plus éloignée du thème de recherche général de cette thèse, mais n'en reste pas moins intéressante. En effet, dans un cadre réel, les nœuds ne coopèrent pas nécessairement pour construire les tables de routage. Par exemple, dans le réseau des AS, chaque entité agit de manière à minimiser ses coûts (financiers), dans cette optique, les AS annoncent des routes avec des métriques fausses pour orienter le trafic comme ils le désirent. Soit pour attirer du trafic soit pour en rejeter en fonction des rapports commerciaux qu'il entretient avec les AS voisins.

Dans ce contexte, il est important qu'il existe un équilibre de Nash pour l'algorithme proposé et les politiques d'administration des nœuds. Par équilibre nous entendons que même si les nœuds agissent de manière égoïste le routage est garanti. Il serait intéressant d'étudier l'effet des politiques sur les différents algorithmes proposés dans cette thèse.

Bibliographie

[ABND95] Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. Sharing memory robustly in message-passing systems. *Journal of the ACM (JACM)*, 42(1):124–142, 1995.

[ABNG94] Baruch Awerbuch, Amotz Bar-Noy, and Madan Gopal. Approximate distributed Bellman-Ford algorithms. *IEEE Transactions on Communications*, 42(8):2515–2519, 1994.

[ABNLP90] Baruch Awerbuch, Amotz Bar-Noy, Nathan Linial, and David Peleg. Improved routing strategies with succinct tables. *Journal of Algorithms*, 11(3):307–341, 1990.

[ABP90] Baruch Awerbuch, Alan Baratz, and David Peleg. Cost-sensitive analysis of communication protocols. In *Proceedings of the ninth annual ACM symposium on Principles of distributed computing*, pages 177–187. ACM, 1990.

[ACL00] William Aiello, Fan Chung, and Linyuan Lu. A random graph model for massive graphs. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 171–180. Acm, 2000.

[ACL+06] Marta Arias, Lenore J Cowen, Kofi A Laing, Rajmohan Rajaraman, and Orjeta Taka. Compact routing with name independence. *SIAM Journal on Discrete Mathematics*, 20(3):705–726, 2006.

[AG94] Anish Arora and Mohamed Gouda. Distributed reset. *Computers, IEEE Transactions on*, 43(9):1026–1038, 1994.

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

[AGM06] Ittai Abraham, Cyril Gavoille, and Dahlia Malkhi. On space-stretch trade-offs: Lower bounds. In 18^{th} *Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 217–224. ACM Press, July 2006.

[AGM+08] Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, Noam Nisan, and Mikkel Thorup. Compact name-independent routing with minimum stretch. *ACM Transactions on Algorithms*, 3(4):Article 37, June 2008.

[AGMW10] Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, and Udi Wieder. Strong-diameter decompositions of minor free graphs. *Theory of Computing Systems*, 47(4):837–855, November 2010.

[AJB99] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Internet: Diameter of the world-wide web. *Nature*, 401(6749):130–131, 1999.

[AKY91] Yehuda Afek, Shay Kutten, and Moti Yung. Memory-efficient self stabilizing protocols for general networks. In *Distributed Algorithms*, pages 15–28. Springer, 1991.

[AM05] Ittai Abraham and Dahlia Malkhi. Name independent routing for growth bounded networks. In 17^{th} *Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 49–55. ACM Press, July 2005.

[Ang80] Dana Angluin. Local and global properties in networks of processors. In *Proceedings of the twelfth annual ACM symposium on Theory of computing*, pages 82–93. ACM, 1980.

[AP92] Baruch Awerbuch and David Peleg. Routing with polynomial communication-space srade-ff. *SIAM Journal on Discrete Mathematics*, 5(2):151–162, 1992.

[AR93] Yehuda Afek and Moty Ricklin. Sparser: A paradigm for running distributed algorithms. *Journal of Algorithms*, 14(2):316–328, 1993.

[Aub03] David Auber. *Tulip : A huge graph visualisation framework*. P. Mutzel and M. Junger, 2003.

[Awe84] Baruch Awerbuch. An efficient network synchronization protocol. In *STOC*, pages 522–525, 1984.

[Awe85] Baruch Awerbuch. A new distributed depth-first-search algorithm. *Information Processing Letters*, 20(3):147 – 150, 1985.

[BBR+11] Lars Backstrom, Paolo Boldi, Marco Rosa, Johan Ugander, and Sebastiano Vigna. Four degrees of separation. *CoRR*, abs/1111.4570, 2011.

[BDBK+94] Shai Ben-David, Allan Borodin, Richard Karp, Gabor Tardos, and Avi Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994.

[BEY98] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*, volume 53. Cambridge University Press Cambridge, 1998.

[BFdlV82] Béla Bollobás and W Fernandez de la Vega. The diameter of random regular graphs. *Combinatorica*, 2(2):125–134, 1982.

[BG92] Dimitri P. Bertsekas and Robert G. Gallager. *Data Networks (2nd edition) – Chp.5: Routing in Data Networks*. Prentice Hall, 1992.

[BK09] Aaron Bernstein and David R. Karger. A nearly optimal oracle for avoiding failed vertices and edges. In Michael Mitzenmacher, editor, *STOC*, pages 101–110. ACM, 2009.

[BMG88] James E Burns, Raymond Edward Miller, and Mohamed G Gouda. *On relaxing interleaving assumptions*. School of Information and Computer Science, Georgia Institute of Technology, 1988.

[BMSU01] Prosenjit Bose, Pat Morin, Ivan Stojmenović, and Jorge Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless networks*, 7(6):609–616, 2001.

[Bol01] Béla Bollobás. *Random graphs*, volume 73. Cambridge university press, 2001.

[BT02] Tian Bu and Don Towsley. On distinguishing between internet power law topology generators. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 638–647. IEEE, 2002.

[BYCR93] Ricardo A. Baeza-Yates, Joseph C. Culberson, and Gregory J. E. Rawlins. Searching in the plane. *Information and Computation*, 106(2):234–252, 1993.

[Cai] CAIDA maps - as relationships. http://www.caida.org/data/active/as-relationships/. Dernière visite : 2013-07-18.

[CCJ90] Brent N Clark, Charles J Colbourn, and David S Johnson. Unit disk graphs. *Discrete mathematics*, 86(1):165–177, 1990.

[CeA+03] Thomas Clausen, Philippe Jacquet (editors), Cédric Adjih, Anis Laouiti, Pascale Minet, Paul Muhlethaler, Amir Qayyum, and Laurent Viennot. Optimized link state routing protocol (OLSR). pages 1–75, October 2003. Network Working Group.

[CFQS12] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

[CG84] F. R. K. Chung and M. R. Garey. Diameter bounds for altered graphs. *Journal of Graph Theory*, 8(4):511–534, 1984.

[CL02] Fan Chung and Linyuan Lu. The average distances in random graphs with given expected degrees. *Proceedings of the National Academy of Sciences*, 99(25):15879–15882, 2002.

[CLPR10] Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. f-sensitivity distance oracles and routing schemes. In *Proceedings of the 18th annual European conference on Algorithms: Part I*, ESA'10, pages 84–96, Berlin, Heidelberg, 2010. Springer-Verlag.

[Cow99] Lenore J Cowen. Compact routing with minimum stretch. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 255–260. Society for Industrial and Applied Mathematics, 1999.

[CS94] Srinivasan Chandrasekar and Pradip K Srimani. A self-stabilizing distributed algorithm for all-pairs shortest path problem. *Parallel Algorithms and Applications*, 4(1-2):125–137, 1994.

[CSTW09] W. Chen, C. Sommer, S.H. Teng, and Y. Wang. Compact routing in power-law graphs. In *Proceedings of the 23rd international conference on Distributed computing*, pages 379–391. Springer-Verlag, 2009.

[DI04] Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *J. ACM*, 51(6):968–992, 2004.

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

[Dim] DIMES maps - as relationships. http://www.netdimes.org/new/. Dernière visite : 2013-07-18.

[DIM93] Shlomi Dolev, Amos Israeli, and Shlomo Moran. Self-stabilization of dynamic systems assuming only read/write atomicity. *Distributed Computing*, 7(1):3–16, 1993.

[Dol93] Shlomi Dolev. Optimal time self stabilization in dynamic systems. In *Distributed Algorithms*, pages 160–173. Springer, 1993.

[Dol00] Shlomi Dolev. *Self-stabilization*. The MIT press, 2000.

[DRM13] DRMSIM. *version 1.3.1*. INRIA COATI, Nice - France, 2013.

[DTCR08] Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37:1299–1318, January 2008.

[Elk05] Michael Elkin. Computing almost shortest paths. *ACM Transactions on Algorithms*, 1(2):283–323, October 2005.

[ER59] P. Erdos and A. Renyi. On random graphs. *Publ. Math. Debrecen*, 6(290-297):156, 1959.

[EZ06] Michael Elkin and Jian Zhang. Efficient algorithms for constructing $(1+\epsilon, \beta)$-spanners in the distributed and streaming models. *Distributed Computing*, 18(5):375–385, April 2006.

[Fel74] William Feller. Introduction to probability theory and its applications, vol. ii pod. 1974.

[FG01a] P. Fraigniaud and C. Gavoille. Routing in trees. *Automata, Languages and Programming*, pages 757–772, 2001.

[FG01b] Pierre Fraigniaud and Cyril Gavoille. Routing in trees. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, 28^{th} *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2076 of Lecture Notes in Computer Science, pages 757–772. Springer, July 2001.

[FJ04] Afonso Ferreira and Aubin Jarry. Complexity of minimum spanning tree in evolving graphs and the minimum-energy broadcast routing problem. In *Proceedings of WiOpt*, volume 4, 2004.

[Gal82] Robert G Gallager. Distributed minimum hop algorithms. Technical report, DTIC Document, 1982.

[Gil59] Edgar N Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, 1959.

[Gil61] Edward N Gilbert. Random plane networks. *Journal of the Society for Industrial & Applied Mathematics*, 9(4):533–543, 1961.

[GP96a] Cyril Gavoille and Stéphane Pérennès. Memory requirement for routing in distributed networks. In *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, pages 125–133. ACM, 1996.

[GP96b] Cyril Gavoille and Stéphane Pérennès. Memory requirement for routing in distributed networks. In 15^{th} *Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 125–133. ACM Press, May 1996.

[GP03] Felix C Gärtner and Henning Pagnia. Time-efficient self-stabilizing algorithms through hierarchical structures. In *Self-Stabilizing Systems*, pages 154–168. Springer, 2003.

[HBS13] Geoff Huston, Tony Bates, and Philip Smith. Cidr report. http://www.cidr-report.org/as2.0/, 2013.

[Hed88] Charles L Hedrick. Routing information protocol. 1988.

[HIKN10] Nicolas Hanusse, David Ilcinkas, Adrian Kosowski, and Nicolas Nisse. Locating a Target with an Agent Guided by Unreliable Local Advice. In *Proceedings of the 29th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing PODC 2010*, pages 355–364, Zurich Suisse, 2010. ACM New York, NY, USA.

[HKK02] Nicolas Hanusse, Evangelos Kranakis, and Danny Krizanc. Searching with mobile agents in networks with liars. *Discrete applied mathematics - ISSN 0166-218X*, 2002.

[HKK04] N. Hanusse, E. Kranakis, and D. Krizanc. Searching with mobile agents in networks with liars. *Discrete Applied Mathematics*, 137:69–85, 2004.

[HKKK08] N. Hanusse, D. J. Kavvadias, E. Kranakis, and D. Krizanc. Memoryless search algorithms in a network with faulty advice. *Theor. Comput. Sci.*, 402(2-3):190–198, 2008.

[HL02] Tetz C Huang and Ji-Cherng Lin. A self-stabilizing algorithm for the shortest path problem in a distributed system. *Computers & Mathematics with Applications*, 43(1):103–109, 2002.

[Hua05] Tetz C Huang. A self-stabilizing algorithm for the shortest path problem assuming the distributed demon. *Computers & Mathematics with Applications*, 50(5):671–681, 2005.

[Huh04] Aleksandr Huhtonen. Comparing aodv and olsr routing protocols. In *Seminar on Internetworking, Sjkulla*, pages 26–27, 2004.

[iCS01] Ramon Ferrer i Cancho and Richard V Solé. The small world of human language. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 268(1482):2261–2265, 2001.

[Joh97] Colette Johnen. Memory efficient, self-stabilizing algorithm to construct bfs spanning trees. In *PODC*, page 288, 1997.

[JSAC01] Christine E Jones, Krishna M Sivalingam, Prathima Agrawal, and Jyh Cheng Chen. A survey of energy efficient network protocols for wireless networks. *wireless networks*, 7(4):343–358, 2001.

[KB10] Neelesh Khanna and Surender Baswana. Approximate shortest paths avoiding a failed vertex: Optimal size data structures for unweighted graphs. In Jean-Yves Marion and Thomas Schwentick, editors, *STACS*, volume 5 of *LIPIcs*, pages 513–524. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[KFY04] D. Krioukov, K. Fall, and X. Yang. Compact routing on internet-like graphs. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 1. IEEE, 2004.

[KGKS05] Y.J. Kim, R. Govindan, B. Karp, and S. Shenker. Geographic routing made practical. pages 217–230, 2005. Routage par face, glouton sur un graphe quasi-planaire — quelques arêtes peuvent persister dans certains cas — l'article présente différentes techniques de planarization de graphes. Leur technique de planarization, permet de garantir qu'un routage par face n'échouera jamais. Cette technique est basé sur un parcours de face permettant la détection de "cross links" (CLDP). L'avantage de cette technique est qu'elle peut être distribuée, chaque noeud du réseau emet un paquet, si le paquet revient avec une notification de croisement, alors l'arête incriminée doit être supprimée si cela ne déconnecte pas le graphe. Description précise.

[Kin99] Valerie King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *FOCS*, pages 81–91, 1999.

[Kis02] Laszlo B Kish. End of moore's law: thermal (noise) death of integration in micro and nano electronics. *Physics Letters A*, 305(3):144–149, 2002.

[KK77] Leonard Kleinrock and Farouk Kamoun. Hierarchical routing for large networks; performance evaluation and optimization. *Computer Networks*, 1(3):155–174, January 1977.

[KK99] E. Kranakis and D. Krizanc. Searching with uncertainty. pages 194–203, 1999.

[KK13] Alex Kravchik and Shay Kutten. Time optimal synchronous self stabilizing spanning tree. In 27^{th} *International Symposium on Distributed Computing (DISC)*, volume 8205 of Lecture Notes in Computer Science, pages 418–432. Springer, October 2013.

[KKP93] David R Karger, Daphne Koller, and Steven J Phillips. Finding the hidden path : Time bounds for all-pairs shortest paths. *SIAM Journal on Computing*, 22(6):1199–1217, 1993.

[Kle04] Jon Kleinberg. The small-world phenomenon and decentralized search. *SiAM News*, 37(3):1–2, 2004.

[Kle07] Robert Kleinberg. Geographic routing using hyperbolic space. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1902–1909. IEEE, 2007.

[KRT96] Ming-Yang Kao, John H. Reif, and Stephen R. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. *Information and Computation*, 131(0092):63–79, 1996.

[KRX06] Goran Konjevod, Andréa Werneck Richa, and Donglin Xia. Optimal-stretch name-independent compact routing in doubling metrics. In 25^{th} *Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 198–207. ACM Press, July 2006.

[KSU99] Evangelos Kranakis, Harvinder Singh, and Jorge Urrutia. Compass routing on geometric networks. In *in Proc. 11 th Canadian Conference on Computational Geometry*. Citeseer, 1999.

[LADW05] Lun Li, David Alderson, John C Doyle, and Walter Willinger. Towards a theory of scale-free graphs: Definition, properties, and implications. *Internet Mathematics*, 2(4):431–523, 2005.

[Lai07] Kofi Ambrose Laing. Name-independent compact routing in trees. *Information Processing Letters*, 103(2):57–60, July 2007.

[LBA10] Antoine Lambert, Romain Bourqui, and David Auber. 3d edge bundling for geographical data visualization. In *Information Visualisation (IV), 2010 14th International Conference*, pages 329–335. IEEE, 2010.

[LGW04] Alberto Leon-Garcia and Indra Widjaja. *Communication Networks*. McGraw-Hill, Inc., New York, NY, USA, 2 edition, 2004.

[Lu01] Linyuan Lu. The diameter of random massive graphs. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 912–921. Society for Industrial and Applied Mathematics, 2001.

[Lu02] Linyuan Lincoln Lu. *Probabilistic methods in massive graphs and internet computing*. PhD thesis, 2002. AAI3061653.

[M+65] Gordon E Moore et al. Cramming more components onto integrated circuits, 1965.

[Mal94] Gary Malkin. Rip version 2-carrying additional information. 1994.

[May06] Petar Maymounkov. Greedy embeddings, trees, and euclidean vs. lobachevsky geometry. Technical report, Technical Report, available at http://pdos. csail. mit. edu/petar/pubs. html, 2006.

[MF03] Pablo Molinero-Fernández. *Circuit Switching in the Internet*. PhD thesis, Stanford University, 2003.

[MRR80] John McQuillan, Ira Richer, and Eric Rosen. The new routing algorithm for the arpanet. *Communications, IEEE Transactions on*, 28(5):711–719, 1980.

[MS97] G Meyer and S Sherry. Triggered extensions to rip to support demand circuits. Technical report, RFC 2091, January, 1997.

[MZF+07] David Meyer, Lixia Zhang, Kevin Fall, et al. Report from the iab workshop on routing and addressing. *IETF Internet Standard, RFC*, 4984, 2007.

[New02] Mark EJ Newman. Assortative mixing in networks. *Physical review letters*, 89(20):208701, 2002.

[NPW03] Enrico Nardelli, Guido Proietti, and Peter Widmayer. Finding the most vital node of a shortest path. *Theor. Comput. Sci.*, 296:167–177, March 2003.

[PB94] Charles E Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *ACM SIGCOMM Computer Communication Review*, volume 24, pages 234–244. ACM, 1994.

[Pel00] David Peleg. *Distributed computing: a locality-sensitive approach*, volume 5. SIAM, 2000.

[PPZ09] Mahendra Piraveenan, Mikhail Prokopenko, and Albert Y Zomaya. Local assortativity and growth of internet. *The European Physical Journal B*, 70(2):275–285, 2009.

[PR99] Charles E Perkins and Elizabeth M Royer. Ad-hoc on-demand distance vector routing. In *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA'99. Second IEEE Workshop on*, pages 90–100. IEEE, 1999.

[PU89] David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *Journal of the ACM (JACM)*, 36(3):510–530, 1989.

[RLH06] Y Rekhter, T Li, and S Hares. Rfc 4271: Border gateway protocol 4, 2006.

[SBvL87] A. Schoone, H. Bodlaender, and J. van Leeuwen. Improved diameter bounds for altered graphs. In Gottfried Tinhofer and Gunther Schmidt, editors, *Graph-Theoretic Concepts in Computer Science*, volume 246 of *Lecture Notes in Computer Science*, pages 227–236. Springer Berlin / Heidelberg, 1987.

[Seg83] Adrian Segall. Distributed network protocols. *Information Theory, IEEE Transactions on*, 29(1):23–35, 1983.

[SGF+10] Ankit Singla, P. Brighten Godfrey, Kevin Fall, Gianluca Iannaccone, and Sylvia Ratnasamy. Scalable routing on flat names. In 6^{th} *International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, page Article No. 20. ACM Press, November 2010.

[Tho04] Mikkel Thorup. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In Torben Hagerup and Jyrki Katajainen, editors, *SWAT*, volume 3111 of *Lecture Notes in Computer Science*, pages 384–396. Springer, 2004.

[TK84] Hideaki Takagi and Leonard Kleinrock. Optimal transmission ranges for randomly distributed packet radio terminals. *Communications, IEEE Transactions on*, 32(3):246–257, 1984.

[TM69] Jeffrey Travers and Stanley Milgram. An experimental study of the small world problem. *Sociometry*, pages 425–443, 1969.

[TS95] John N. Tsitsiklis and George D. Stamoulis. On the average communication complexity of asynchronous distributed algorithms. *Journal of the ACM*, 42(2):382–400, March 1995.

[Tul13] Tulip. *version 3*. LaBRI, CNRS, INRIA, Université de Bordeaux, France, 2013. Dernière visite : 2013-08-09.

[TZ01] M. Thorup and U. Zwick. Compact routing schemes. In *13 th Annual Symposium on Parallel Algorithms and Architectures(SPAA 2001)*, pages 1–10, 2001.

[TZLL12] M. Tang, G. Zhang, T. Lin, and J. Liu. Hdlbr: a name-independent compact routing scheme for power-law networks. *Computer Communications*, 2012.

[WC03] Xiao Fan Wang and Guanrong Chen. Complex networks: small-world, scale-free and beyond. *Circuits and Systems Magazine, IEEE*, 3(1):6–20, 2003.

[WP09] Cedric Westphal and Guanhong Pei. Scalable routing via greedy embedding. pages 2826–2830, 2009.

[WS98] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440–442, 1998.

Annexes A

A.1 Algorithmes distribués détaillés du calculs d'étiquettes de routage pour l'algorithme DCR

Cette section donne les détails omis de la section 3.6.3.

Algorithm 15: Réception par un nœud u d'un message $\mathrm{mStartLabel}()$ provenant d'un nœud w

```
1  if w = D_u[lmin_u].nexthop then
2  |   globalValid_u ← 0
3  |   SEND_{Γ(u)}(mStartLabel())
4  |   D_u[u].size ← 1
5  |   D_u[u].largestChild ← null
6  |   D_u[u].largestSize ← 0
7  |   D_u[u].valid ← 0
8  |   SEND_{Γ(u)}(mSizes(u))
9  else
10 |   SEND_w(mCollectTermination())
11 end
```

Algorithm 16: Réception par un nœud u d'un message $\mathrm{mSizes}(v)$ provenant d'un nœud w

```
1  if w = D_u[v].nexthop then
2  |   D_u[v].valid ← 0
3  |   SEND_{Γ(u)}(mSizes(v))
4  else
5  |   SEND_w(mAckSizes(v, 0))
6  end
```

Algorithm 17: Réception par un nœud u d'un message d'acquittement $\mathrm{mAckSizes}(v, \texttt{size})$ provenant d'un nœud w

$\mathcal{D}_u[v].\mathrm{size} \leftarrow \mathcal{D}_u[v].\mathrm{size} + \texttt{size}$
if $\texttt{size} > \mathcal{D}_u[v].\mathrm{largestSize}$ **then**
  $\mathcal{D}_u[v].\mathrm{largestSize} \leftarrow \texttt{size}$
  $\mathcal{D}_u[v].\mathrm{largestChild} \leftarrow w$
end
$\mathcal{D}_u[v].\mathrm{valid} \leftarrow \mathcal{D}_u[v].\mathrm{valid} + 1$
if $\mathcal{D}_u[v].\mathrm{valid} = \mathrm{degree}(u)$ **then**
  if $v \neq u$ **then**
    $v' \leftarrow \mathcal{D}_u[v].\mathrm{nexthop}$
    $\mathrm{SEND}_{v'}(\mathrm{mAckSizes}(v, \mathcal{D}_u[v].\mathrm{size}))$
  else
    $\mathcal{D}_u[u].\mathrm{label} \leftarrow \mathrm{rootLabel}(u, \mathcal{D}_u[u].\mathrm{largestChild})$
    $\mathrm{SEND}_{\Gamma(u)}(\mathrm{mLabels}(u, \mathcal{D}_u[u].\mathrm{label}))$
  end
end

Algorithm 18: Réception par un nœud u d'un message $\mathrm{mLabels}(v, \texttt{label})$ provenant d'un nœud w

if $w = \mathcal{D}_u[v].\mathrm{nexthop}$ **then**
  $\mathcal{D}_u[v].\mathrm{label} \leftarrow \mathrm{computeLabel}(u, \texttt{label}, \mathcal{D}_u[v].\mathrm{largestChild})$
  $\mathrm{SEND}_{\Gamma(u)}(\mathrm{mLabels}(v, \mathcal{D}_u[v].\mathrm{label}))$
  if (*all labels at u have been computed*) $\wedge$ $(\mathrm{globalValid}_u = \mathrm{degree}(u))$ **then**
    if $u \neq \mathrm{lmin}_u$ **then**
      $v' \leftarrow \mathcal{D}_u[\mathrm{lmin}_u].\mathrm{nexthop}$
      $\mathrm{SEND}_{v'}(\mathrm{mCollectTermination}())$
    else
      $\mathrm{closestLandmark}_u \leftarrow u$
      $\mathcal{I}_u \leftarrow \varnothing$
      $\mathrm{LogNeighbor}_u \leftarrow \varnothing$
      $\mathrm{LogRoot}_u \leftarrow \{1, \cdots, k\}$
      $\mathrm{LogRoot}_u[1].\mathrm{node} \leftarrow u$
      $\mathrm{LogRoot}_u[1].\mathrm{label} \leftarrow \mathcal{D}_u[u].\mathrm{label}$
      $\mathrm{SEND}_{\Gamma(u)}(\mathrm{mLabelsDone}())$
    end
  end
end

Algorithm 19: Réception par un nœud u d'un message $\mathrm{mCollectTermination}()$ provenant d'un nœud w

```
globalValid_u ← globalValid_u + 1
if (globalValid_u = degree(u)) ∧ (all labels at u have been computed) then
    if u ≠ lmin_u then
        v' ← D_u[lmin_u].nexthop
        SEND_{v'}(mCollectTermination())
    else
        closestLandmark_u ← u
        I_u ← ∅
        LogNeighbor_u ← ∅
        LogRoot_u ← {1, ··· , k}
        LogRoot_u[1].node ← u
        LogRoot_u[1].label ← D_u[u].label
        SEND_{Γ(u)}(mLabelsDone())
    end
end
```

Algorithm 20: Réception par un nœud u d'un message $\mathrm{mLabelsDone}()$ provenant d'un nœud w

```
if w = D_u[lmin_u].nexthop then
    closestLandmark_u ← M_u[1].node
    I_u ← ∅
    LogNeighbor_u ← ∅
    SEND_{Γ(u)}(mLabelsDone())
    v' ← D_u[lmin_u].nexthop
    SEND_{v'}(mLogicalParent(u, D_u[lmin_u].label, c(u)))
    if u is not a landmark then
        v ← M_u[h(u)].node
        v' ← D_u[v].nexthop
        SEND_{v'}(mLogicalContact(v, u, closestLandmark_u, D_u[closestLandmark_u].label))
        foreach v ∈ B(u) do
            v' ← D_u[v].nexthop
            SEND_{v'}(mLocalBroadcast(v, u, D_u[v].label))
        end
    end
end
```