Algorithmes des graphes et des réseaux

Laurent Viennot

Algorithmes des graphes et des réseaux

Laurent Viennot

Mots-clés : algorithmes, graphes, réseaux, parcours de graphe, inondation, plus courts chemins, routage, arbre couvrant, diffusion, flot, coloration.

Résumé : les graphes constituent une structure mathématique privilégiée pour modéliser les réseaux. Une grande partie de l'algorithmique des réseaux entretient donc des liens étroits avec l'algorithmique des graphes. Ce chapitre propose de mettre en lumière ces relations en explorant notamment les techniques utilisées pour coordonner entre eux les routeurs d'un réseau. La diffusion d'un message s'apparente ainsi au parcours d'un graphe, le problème du routage revient à calculer de plus courts chemins. Les trafics d'un réseaux se modélisent par des flots dans un graphe et les problèmes d'allocation de fréquences dans un réseau radio s'apparentent à la coloration d'un graphe. Chaque algorithme de graphe repose souvent sur l'utilisation d'une structure mathématique sous-jacente comme un chemin, un arbre ou un flot. La compréhension de l'algorithme se fonde alors sur les propriétés mathématiques de cette structure.

1 Graphes et réseaux

De même que les villes sont reliées par des routes, les routeurs d'Internet sont reliés par des liens de communication. Un réseau peut donc se modéliser par un graphe dont les sommets sont les routeurs et les arcs sont les liens de communication. La figure 1.1 illustre ainsi le réseau national de la recherche (Renater 3) qui constitue la portion académique de l'Internet français en novembre 2004. Pour les besoins de la discussion, nous identifierons les routeurs par la ville dans laquelle ils se trouvent.

Il faut noter que les liens d'un réseau sont généralement bidirectionnels : l'existence d'un lien entre Pau et Bordeaux, par exemple, indique qu'il est possible d'envoyer des données de Pau vers Bordeaux, mais aussi de Bordeaux vers Pau, ce qui se représentera par les deux arcs $(\text{Pau}, \text{Bordeaux})$ et $(\text{Bordeaux}, \text{Pau})$ dans le graphe modélisant le réseau. On dit alors que le graphe est *symétrique* (et représente donc un graphe *non orienté*). Les arcs peuvent de plus être *valués* pour refléter une métrique sur les liens. On associe alors un *poids* $l(u, v)$ à tout arc (u, v) du graphe. Selon les cas, ce poids pourra être considéré comme un coût, une longueur ou un délai. Par exemple, en valuant par le temps de transmission d'un giga-octet (soit 8 Gbits ou 8000 Mbits), on obtiendra ainsi $l(\text{Pau}, \text{Bordeaux}) = 8000/2400$ et $l(\text{Dijon}, \text{Paris}) = 8000/622$. Dans le cas le plus simple, tous les arcs ont poids 1. Les algorithmes décrits dans ce chapitre s'effectuent pour la plupart sur un graphe donné dont on désignera par n le nombre de sommets et par m le nombre d'arcs. Ces nombres déterminent la taille de l'entrée pour un calcul sur un graphe. Ainsi un algorithme linéaire devra avoir un nombre d'opérations proportionnel à $n + m$, ce qui implique généralement de ne considérer qu'un nombre constant de fois chaque sommet et chaque arc.

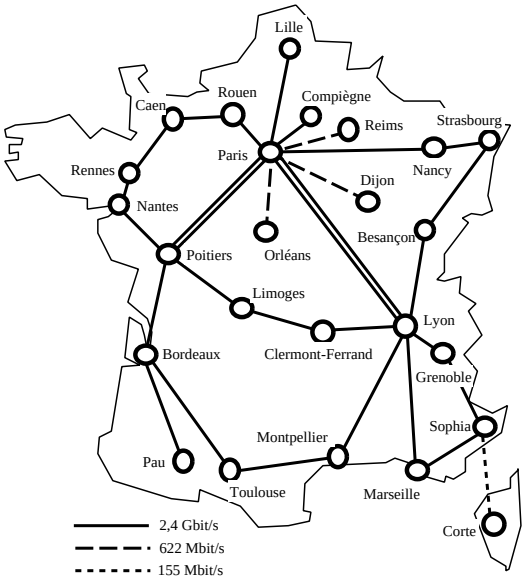

FIG. 1.1 – Le réseau Renater 3 en 2004

Le routage dans un réseau consiste à acheminer des données de proche en proche en suivant

plusieurs arcs pour atteindre la destination voulue. Plus précisément, il s'agit de trouver une suite d'arcs $(u_1, u_2), (u_2, u_3), \ldots, (u_{k-1}, u_k)$ joignant la source u_1 à la destination u_k. Une telle suite d'arcs s'appelle un *chemin* de u_1 à u_k et constitue l'une des structures les plus élémentaires de l'algorithmique des graphes. Cependant, la découverte efficace de chemins se heurte à l'explosion combinatoire de leur nombre : comment trouver des chemins optimaux parmi la multitude de ceux qui existent dans un graphe ? Les algorithmes fondamentaux de graphes obtiennent une complexité linéaire en ne considérant qu'une seule fois chaque arc du graphe tout en garantissant l'exploration d'un ensemble satisfaisant de chemins (voir le paragraphe 3).

Les réseaux de données offrent un domaine très particulier d'application des graphes. En effet, un réseau peut tenir lieu à la fois de graphe d'entrée sur lequel portent les calculs et de vecteur de transmission de messages pour effectuer ce calcul de manière distribuée. Cette situation se rencontre en particulier dans les protocoles de routage qui spécifient les échanges d'informations entre routeurs leur permettant de construire les tables de routage. Ces échanges ne pouvant alors reposer sur le routage lui-même, ils doivent utiliser des mécanisme plus basiques d'échanges d'informations comme la diffusion par exemple.

2 Parcours de graphe et diffusion dans un réseau

Une des méthodes algorithmiques les plus fondamentales est le *parcours* d'un graphe. De nombreux algorithmes sont issus de cette méthode. Il s'agit de visiter un à un les sommets d'un graphe en suivant les arcs. Pour cela, la structure de donnée codant le graphe doit permettre d'obtenir la liste des *voisins* d'un sommet u, c'est-à-dire les sommets v atteignables par un arc (u, v) depuis u (voir la structure de listes d'adjacences au chapitre ££Structures de données££). La fonction *parcours* ci-après permet de parcourir un graphe à partir d'un sommet u.

Pour les besoins de la discussion, nous dirons qu'un sommet est *découvert* dès qu'il a été marqué comme « découvert », et qu'il a été visité une fois qu'il est marqué comme « visité ». Un sommet peut être découvert à plusieurs reprises mais n'est visité qu'une seule fois. La figure 1.2 illustre ainsi le début d'un parcours sur le graphe du réseau Renater 3 de la figure 1.1. Les sommets noirs sont marqués visités et les sommets hachurés sont marqués découverts. Seuls figurent les arcs utilisés jusque-là, c'est-à-dire ceux entre un sommet visité et ses voisins. Poitiers a été découvert par Bordeaux et Limoges mais Limoges n'a été découvert que par Clermont-Ferrand.

```
parcours(u) {          /* Parcours à partir du sommet u. */
  marquer le sommet u comme « découvert ».
  tant que il existe un sommet marqué « découvert » faire
    prendre un sommet v marqué « découvert ».
    si v n'est pas marqué « visité » alors
      marquer v comme « visité ».
      marquer les voisins de v comme « découverts ».
    fin si
  fin tant que
}
```

La méthode de parcours est intimement liée à la notion de *connexité*. En effet, l'ensemble des sommets v qui seront visités par un appel à *parcours*(u) sont ceux atteignables par un chemin $(u, v_1), (v_1, v_2), \ldots, (v_{k-1}, v)$ reliant u à v. Un graphe non orienté est dit *connexe* s'il existe un chemin reliant toute paire de sommets du graphe. C'est par exemple le cas pour le graphe de Renater 3 de la figure 1.1. Dans le cas d'un graphe non orienté, on peut montrer qu'un appel à *parcours*(u) parcourra toute la *composante connexe* du sommet u, c'est-à-dire le plus grand sous-ensemble de sommets contenant u et formant un graphe connexe. Le coût de l'appel est linéaire en le nombre d'arcs figurant dans cette composante connexe.

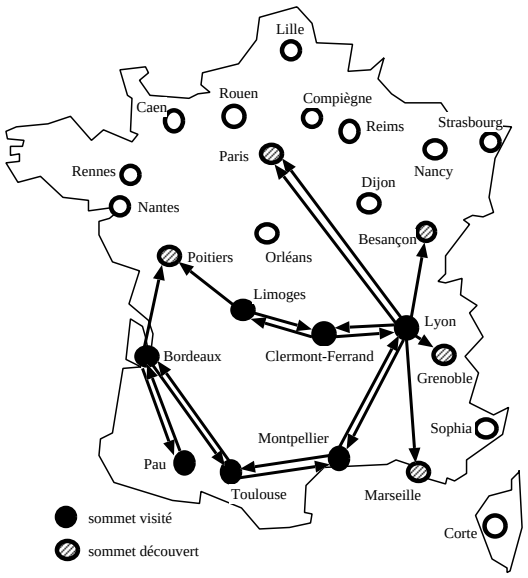

FIG. 1.2 – Le graphe du réseau Renater 3 en cours de parcours

Une technique similaire au parcours de graphe permet de diffuser un message dans un réseau, il s'agit de l'*inondation*. L'intérêt de cette technique provient du fait que le graphe connectant les routeurs d'un réseau n'est pas connu à priori. En effet, le réseau est dynamique : des liens peuvent « casser », des routeurs peuvent tomber en panne, et parfois des liens ou des routeurs peuvent être ajoutés. L'inondation permet à un routeur de diffuser un message dans tout le réseau sans que les routeurs n'aient connaissance de la topologie du réseau. L'algorithme d'inondation peut s'exprimer ainsi :

declenche_inondation() {
  un routeur qui veut diffuser un message M émet ce message sur chacun de ses liens de communication.
}
reception_inondation() {
  un routeur qui reçoit un message d'inondation M pour la première fois le retransmet sur chacun de ses liens de communication.
}

L'inondation nécessite que chaque routeur puisse décider s'il a déjà reçu le message. Pour cela, le protocole spécifiant les échanges de messages entre les routeurs prévoit généralement que le message contienne un identifiant unique, composé de l'identifiant du routeur origine du message et d'un numéro de séquence qui est incrémenté chaque fois que le routeur origine envoie un nouveau message.

L'inondation est une version distribuée de parcours. Avoir reçu le message revient pour un routeur à être marqué « visité ». Envoyer le message à d'autres routeurs revient à les marquer comme « découverts ». Le coût d'une inondation revient à l'envoi d'un message par arc du réseau. Une optimisation simple consiste à ne pas renvoyer le message sur le lien depuis lequel il a été reçu.

Lors d'une inondation, l'ordre de visite des routeurs est difficilement contrôlable puisque les retransmissions se font en parallèle. À l'inverse, de nombreux algorithmes de graphes sont dérivés de la méthode de parcours en contrôlant minutieusement l'ordre dans lequel les sommets sont visités. En stockant les sommets découverts dans une pile[1], on obtient l'algorithme de parcours en profondeur dans lequel on visite d'abord le sommet le plus récemment découvert. Un parcours à partir de Bordeaux du graphe de Renater 3 de la figure 1.1 visitera par exemple les sommets dans l'ordre Bordeaux, Pau, Toulouse, Montpellier, Lyon, Clermont-Ferrand, Limoges comme illustré par la figure 1.2. Le prochain sommet visité sera alors forcément le dernier découvert, c'est-à-dire Poitiers, et ainsi de suite...

Le *parcours en profondeur* est sans doute le premier algorithme de parcours de graphe inventé (les premières versions remontent à la fin du dix-neuvième siècle dans les travaux de Charles Trémaux et ceux de Gaston Tarry sur l'exploration de labyrinthes). Il peut s'écrire très simplement sous forme d'une fonction récursive, ce qui évite de manier directement la structure de pile. C'est sans doute Robert Tarjan (1972) qui a montré tout le potentiel de cette technique qui permet de calculer efficacement les *composantes fortement connexes* d'un graphe orienté (les composantes maximales telles qu'il existe un chemin orienté de tout sommet vers tout autre), ou encore les *composantes biconnexes* (les composantes maximales d'un graphe non orienté qui restent connexes après retrait d'un sommet quelconque). Notons que la biconnexité est une propriété intéressante pour un réseau car elle garantit que le réseau ne sera pas déconnecté par la panne d'un routeur.

En utilisant plutôt une file[1] pour stocker les sommets découverts, on obtient l'algorithme de *parcours en largeur* dans lequel on visite les sommets dans l'ordre de leur découverte. Le parcours en largeur permet de calculer des plus courts chemins comme nous allons le voir dans la section suivante.

3 Plus courts chemins et routage

Le problème du routage dans un réseau consiste à calculer des chemins efficaces pour acheminer les données qui transitent dans le réseau, c'est-à-dire des chemins courts. Définissons pour l'instant un plus court chemin comme un chemin utilisant un nombre minimal d'arcs.

Le routage par paquet consiste à acheminer chaque paquet de données indépendamment. Un flot de données est donc découpé en paquets, chacun contenant une en-tête avec les informations nécessaires au routage, notamment l'adresse de la destination. Un routeur qui reçoit un paquet doit donc décider très rapidement à quel voisin le faire passer. Il utilise pour cela une table de routage qui indique pour une destination donnée le voisin à qui retransmettre le paquet. Un *protocole de routage* est un algorithme distribué d'échange de messages de contrôle permettant à chaque routeur de construire sa table de routage.

D'un point de vue algorithmique, les protocoles de routage les plus simples sont les protocoles dits à « *états de liens* ». Par contre, la spécification de tels protocoles est généralement as-

[1] Les piles et les files sont décrites au chapitre ££Structures de données££.

sez longue car de nombreux échanges d'informations différents doivent être décrits. Le plus célèbre est sans doute OSPF qui est largement utilisé dans l'Internet (voir le chapitre ££Protocoles réseaux££). L'idée générale de ce type de protocole est de découvrir l'état des liens : cassé, unidirectionnel ou en bon état par exemple. Chaque routeur échange pour cela des message réguliers avec ses voisins. Il diffuse de plus régulièrement cette information dans tout le réseau au moyen du mécanisme d'inondation décrit dans la section précédente. Chaque routeur apprend ainsi toute la topologie du réseau et peut utiliser un algorithme de calcul de plus courts chemins pour calculer sa table de routage à partir de sa vision la plus récente du réseau : étant donné un chemin permettant d'atteindre une destination, la table de routage stockera le voisin apparaissant au début du chemin comme le routeur à qui faire passer un paquet pour la destination.

Utiliser des plus courts chemins possède deux avantages. Le premier est d'économiser les liens du réseau en les encombrant le moins possible. Le deuxième est de garantir qu'un paquet se rapprochera toujours de sa destination : même si les routeurs empruntés consécutivement par un paquet ne sont pas d'accord sur le chemin à emprunter pour atteindre la destination, il suffit que chaque routeur ait effectivement calculé un plus court chemin pour assurer que le routeur suivant se trouve un peu plus proche de la destination. Les protocoles de routage par états de liens ramènent donc le calcul des tables de routage à celui de plus courts chemins dans le graphe connu par le routeur.

Comme nous l'avons vu précédemment, la notion de parcours est liée à l'existence de chemins : un sommet v sera visité par un parcours à partir de u s'il existe un chemin reliant u à v. En notant depuis quel sommet chaque sommet a été découvert pour la première fois, la méthode permet de plus de construire de tels chemins. Si l'on utilise une file pour stocker les sommets découverts non encore visités, on obtient des plus courts chemins (en nombre d'arcs) depuis le sommet source u du parcours. Il s'agit de l'algorithme de *parcours en largeur* attribué à Edward Moore à la fin des années 50, époque à laquelle la plupart des algorithmes de plus courts chemins furent inventés. (Voir la procédure *largeur*(u) ci-après.)

La distance d'un sommet v peut être définie comme le nombre d'arcs d'un plus court chemin reliant u à v. L'idée de l'algorithme est de visiter d'abord u, puis les sommets à distance 1, puis les sommets à distance 2, et ainsi de suite. La structure de file permet de visiter les sommets dans l'ordre de leur découverte. Ainsi, un parcours en largeur à partir de Bordeaux du réseau Renater 3 (de la figure 1.1) visitera par exemple dans l'ordre les sommets Bordeaux, Toulouse, Poitiers, Pau, Montpellier, Paris, Nantes, *etc...*

```
largeur(u) {  /* Parcours en largeur à partir du sommet u. */
  initialiser une file F vide.
  marquer le sommet u comme « découvert » et enfiler u dans F.
  définir la distance de u comme 0.
  tant que il existe un sommet marqué « découvert » (F non vide) faire
    défiler de F le premier sommet v.
    si v n'est pas marqué « visité » alors
      marquer v comme « visité ».
      pour chaque voisin w de v non marqué comme « découvert » faire
        marquer w comme « découvert » et enfiler w dans F.
        définir v comme le père de w.
        définir la distance de w comme la distance de v plus 1.
      fin pour
    fin si
  fin tant que
}
```

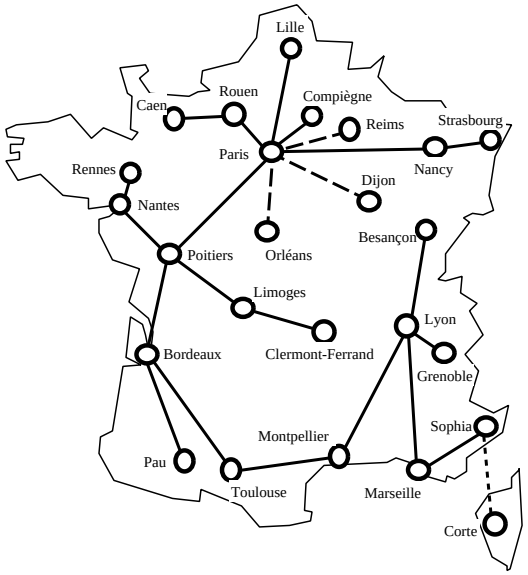

FIG. 1.3 – Un arbre couvrant du réseau Renater 3

En remontant de père en père depuis un sommet v jusqu'à u, on obtient un plus court chemin reliant u à v. La figure 1.3 donne le résultat d'un calcul de plus courts chemins sur le réseau Renater 3 de la figure 1.1 à partir de Bordeaux. Seuls les liens entre un sommet et son père sont affichés, ce qui donne un arbre couvrant du graphe

dont il faut considérer ici Bordeaux comme la racine (la structure d'arbre est définie au chapitre ££Structures de données££).

Cependant, définir les plus courts chemins en ne tenant compte que du nombre d'arcs n'est pas suffisant pour l'acheminement des données dans un réseau. En effet, il faut aussi tenir compte de divers paramètres comme la bande passante ou la charge des liens. Un cadre algorithmique général consiste à donner à chaque arc du graphe un poids que nous interpréterons ici comme une longueur (l'inverse de la bande passante du lien par exemple). On définit alors la *longueur d'un chemin* comme la somme des longueurs des arcs qui le constituent. Diverses métriques peuvent entrer dans la définition de la longueur d'un arc offrant ainsi la possibilité d'optimiser les chemins suivant différents critères avec le même algorithme.

Calculer des chemins de longueur minimale est un problème sensiblement plus difficile que de calculer des chemins possédant un nombre minimal d'arcs. L'algorithme de Dijkstra le résout au moyen d'un parcours avec une file de priorité. Une estimation de distance est tenue à jour pour chaque sommet découvert, et la file de priorité permet de sélectionner efficacement celui de distance estimée minimale.

```
dijkstra(u) {          /* Chemins de longueur
minimale à partir du sommet u. */
  initialiser une file de priorité F vide.
  marquer le sommet u comme « découvert »
  et enfiler u dans F.
  définir la distance de u comme dist(u) = 0
  et la distance des autres sommets comme
  infinie.
  tant que il existe un sommet marqué « dé-
  couvert » (F non vide) faire
    défiler de F le sommet v de distance mi-
    nimale.
    marquer v comme « visité ».
    pour chaque voisin w de v faire
      si w n'est pas marqué comme « décou-
      vert » alors
        marquer w comme « découvert » et
        enfiler w dans F.
      fin si
      si dist(v) + l(v,w) < dist(w) alors
        définir v comme le père de w.
        définir la distance de w comme
        dist(w) = dist(v) + l(v,w).
        mettre à jour F pour tenir compte de
        la nouvelle estimation de distance de
        w.
      fin si
    fin pour
  fin tant que
}
```

La distance $dist(v)$ d'un sommet v calculée par l'algorithme est la longueur minimale d'un chemin de u à v. L'algorithme repose sur le fait qu'un chemin de longueur minimale $u, u_1, \ldots, u_k, v$ de u à v se compose forcément d'un chemin de longueur minimale de u à u_k plus un arc. L'algorithme présuppose de plus que les arcs ont tous une longueur positive. En particulier l'arc (u_k, v) étant positif, u_k sera visité avant v. La complexité de l'algorithme dépend de la structure de données utilisée pour gérer la file de priorité. Une structure très simple telle qu'une liste par exemple permet d'obtenir une complexité quadratique. En utilisant un tas, on obtient une complexité linéaire à un facteur logarithmique près.

L'utilisation de métriques différentes sur les liens peut nécessiter de modifier le calcul de distance dans l'algorithme. Ainsi, on peut définir le goulot d'étranglement d'un chemin comme son arc le plus long. Cette définition est intéressante pour trouver des chemins de bande passante maximale par exemple. La bande passante obtenue sur un chemin sera en effet celle de son lien de plus bas débit. En définissant alors la longueur d'un chemin comme la longueur de son goulot d'étranglement, et en utilisant le max au lieu de la somme calcul de distance, l'algorithme de Dijkstra permet de trouver des chemins de goulot d'étranglement de longueur minimale.

Les premiers protocoles de routage d'Internet (comme RIP) reposent sur un algorithme vraiment distribué mais plus instable vis-à-vis de la dynamique du réseau que les protocoles à états de liens. Il s'agit des protocoles de routage par « *vecteur de distances* ». Du point de vue de l'échange de messages, ces protocoles sont très simples : chaque routeur se contente d'échanger avec ses voisins le vecteur de ses distances estimées avec les autres routeurs du réseau. Ces protocoles sont basés sur une version asynchrone de l'algorithme de Bellman-Ford. Pour se rapprocher de la problématique du routage, nous en donnons ici la version où l'on calcule la distance *vers* une destination u donnée (ce qui revient à calculer la distance *depuis* u dans le graphe où l'on a renversé les arcs) :

```
belfor(u) { /* Distances à une destination u.
*/
  initialiser la distance de chaque sommet
  v ≠ u à dist(v) = ∞.
  dist(u) = 0.
  pour i = 1 à n faire
    pour chaque sommet v faire
      dist(v) =
          min_{w voisin de v} (l(v,w) + dist(w)).
    fin pour
  fin pour
}
```

Le voisin qui réalise la distance minimale peut être utilisé comme « père » pour calculer effectivement les chemins de longueur minimale. L'avantage de cet algorithme est de pouvoir fonctionner même si certains arcs ont des longueurs négatives, ce qui ne pose pas de problème tant qu'il n'y a pas de circuit de longueur négative. La boucle principale est répétée n fois car n est une borne supérieure du nombre d'arcs dans un chemin de longueur minimale. La version asynchrone des protocoles de routage par vecteur de distances devient :

```
distvect(v) {     /* Routage distance vecteur
pour le routeur v. */
  boucle
    si un voisin w de v lui envoie son vecteur
    distance alors
      pour toute destination u faire
        stocker la distance dist_u(w) de w à
        u estimée par w.
      fin pour
    fin si
    pour toute destination u ≠ v faire
      dist_u(v) =
        min_{w voisin de v} (l(v,w) + dist_u(w)).
    fin pour
    dist_v(v) = 0.
    v envoie son vecteur de distances à ses
    voisins.
  fin boucle
}
```

La convergence des distances estimées vers les distances exactes, et donc celle des routes induites par les tables de routage vers des routes optimales, est loin d'être évidente a priori. En effet, chaque itération s'effectue de manière asynchrone sur chaque routeur et avec des délais de transmission entre routeurs variables. Un routeur peut très bien effectuer une itération avec un vecteur distance récent pour un voisin et obsolète pour un autre. Cependant, il est possible de prouver la convergence si chaque routeur arrive à transmettre son vecteur distance régulièrement à chacun de ses voisins (Bertsekas et Gallager, 1992). Mais le temps de convergence peut être assez long et même très coûteux en messages. Par exemple, si un routeur u tombe en panne, l'estimation de distance des autres routeurs vers u va croître d'une longueur d'arc à chaque itération jusqu'à une distance considérée comme infinie. C'est le problème du « comptage à l'infini » qui a donné naissance aux protocoles à états de liens plus robustes à la dynamique du réseau. Une version plus résistante que l'échange de vecteurs de distances repose sur l'échange de vecteurs de chemins, ce qui permet de détecter plus rapidement ce genre de pannes et d'éviter les boucles dans le routage. Cette version est à la base du protocole BGP utilisé entre systèmes autonomes par les routeurs de haut niveau de l'Internet (voir le chapitre ££Protocoles réseaux££).

Notons enfin le lien entre les chemins d'un graphe et le calcul matriciel. Tout graphe peut en effet être représenté sous sa forme matricielle M telle que $M_{uv} = l(u,v)$ s'il existe un arc entre u et v et $M_{uv} = \infty$ sinon. Pour un graphe non valué, on considère $l(u,v) = 1$ pour tout arc (u,v). En effectuant le produit de matrice M^k sur le semi-anneau[2] $(\mathbb{R}, \min, +)$, on obtient la matrice de distances en k sauts : M^k_{uv} est la longueur minimale d'un chemin de k arcs de u à v. Cette approche donne lieu au calcul de chemins de longueur minimale entre tout couple de sommets par l'algorithme de Floyd-Warshall qui permet aussi de calculer la fermeture transitive d'un graphe ou la somme des puissances itérées d'une matrice booléenne. L'algorithme repose sur une optimisation de type programmation dynamique du calcul matriciel. Utiliser un autre semi-anneau permet de calculer d'autres métriques sur les chemins. Le semi-anneau $(\mathbb{R}, \min, \max)$ permet, par exemple, de calculer des chemins de goulot d'étranglement minimal.

4 Diffusion optimisée et communication multipoint

L'algorithme d'inondation décrit au paragraphe 2 permet de diffuser un message dans tout un réseau mais au prix d'une transmission par lien. En considérant que chaque transmission a un coût, l'optimisation de la diffusion revient à calculer un arbre couvrant de poids minimal. Le poids d'un arbre est en effet défini comme la somme des poids de ses liens et le poids d'un lien est alors interprété comme le coût d'une transmission par ce lien. L'algorithme de Prim construit un tel arbre de manière similaire à l'algorithme de Dijkstra décrit à la section 3. L'algorithme part d'une racine quelconque et parcourt le graphe en visitant d'abord les sommets atteignables par un lien de coût minimal. D'un autre côté, l'algorithme de Kruskal se rapproche du calcul de composantes connexes (voir la définition de *kruskal*() ci-après).

Dans cet algorithme, plusieurs arbres croissent en parallèle jusqu'à s'unifier finalement en un arbre couvrant de poids minimal. Une complexité quasiment linéaire peut être obtenue en utilisant la technique d'« *union-find* » qui permet de tester

[2] Dans le semi-anneau $(\mathbb{R}, \min, +)$, min tient lieu d'opération d'addition et + d'opération de multiplication (il faut vérifier que + est bien distributive par rapport à min : $a + \min(b,c) = \min(a+b, a+c)$ pour tous $a, b, c \in \mathbb{R}$).

efficacement au cours du calcul si deux sommets sont dans un même arbre (et donc reliés par T). Le principal coût de l'algorithme provient alors du tri des liens. Une version distribuée de cet algorithme est, par exemple, utilisée pour construire un arbre de diffusion utilisant les liens les moins chargés du réseau.

```
kruskal() {   /* Arbre de poids minimal. */
  initialiser l'ensemble des liens de l'arbre à
  T = ∅.
  pour chaque lien (u, v) pris par coût crois-
  sant faire
    si u et v ne sont pas reliés dans T alors
      rajouter (u, v) dans T.
    fin si
  fin pour
  renvoyer T.
}
```

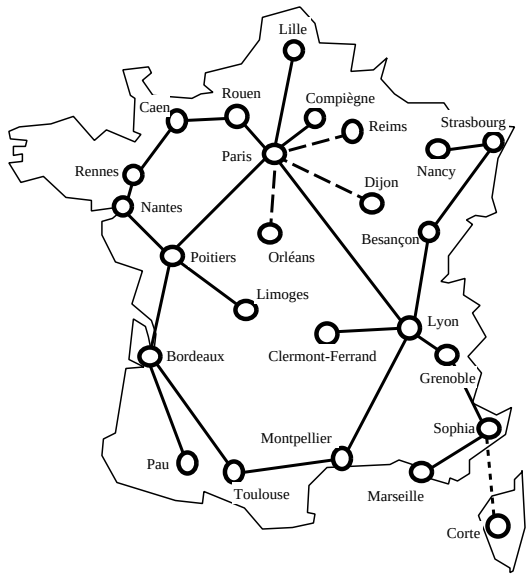

FIG. 1.4 – Un 4-couvrant du réseau Renater 3

Si un arbre couvrant de poids minimal permet d'optimiser le coût d'une diffusion en terme de trafic dans le réseau, il peut considérablement rallonger le délai de la diffusion. Par exemple, en utilisant l'arbre couvrant de la figure 1.3, une diffusion depuis Besançon atteindra Strasbourg avec un délai huit fois supérieur à une inondation sur tous les liens. Pour en prendre en compte ce problème, David Peleg introduit la notion de « *k-couvrant* » (« *k-spanner* » en anglais) dans un graphe où les liens sont valués à la fois par des poids et des longueurs. Un k-couvrant d'un graphe G est un sous graphe H de poids minimal tel que la distance $d_H(u, v)$ entre deux sommets u et v dans ce sous-graphe est au plus k fois la distance $d_G(u, v)$ dans le graphe (la distance étant définie comme la longueur d'un chemin de longueur minimale). Le graphe H étant alors forcément connexe, il suffit d'appliquer l'algorithme d'inondation sur les seuls liens de H pour diffuser de manière optimale avec un délai borné. La figure 1.4 donne un exemple de 4-couvrant, symétrique en l'occurrence, pour le réseau Renater 3 de la figure 1.1. La distance entre Paris et Nancy a par exemple été multipliée par 4 par rapport au réseau non tronqué. Il n'est de plus pas possible de retirer un lien sans perdre la propriété de 4-couvrant.

Trouver un couvrant est un problème NP-difficile[3], mais l'algorithme de Kruskal ci-dessus se généralise facilement en une heuristique permettant de calculer en temps polynomial un $\log n$-couvrant contenant $O(n)$ liens et dont le poids est à un facteur $O(\log n)$ du poids d'un arbre couvrant de poids minimal (Peleg et Schäffer, 1989) :

```
couvrant(k) {   /* Heuristique de calcul d'un
k-couvrant. */
  initialiser l'ensemble des liens du couvrant
  à H = ∅.
  pour chaque lien (u, v) pris par coût crois-
  sant faire
    si d_H(u, v) > k l(u, v) alors
      rajouter (u, v) dans H.
    fin si
  fin pour
  renvoyer H.
}
```

Il est assez aisé de vérifier que cette heuristique calcule bien un k-couvrant. En effet, quand l'algorithme termine on a $d_H(u, v) \leq kl(u, v)$ pour tout couple de sommets u et v. Un chemin de longueur L dans le graphe pourra donc être remplacé par un chemin de longueur au plus $k\,L$ dans H. L'approximation à un facteur $O(\log n)$ près avec $O(n)$ liens est obtenue pour $k = \log n$.

En ce qui concerne l'optimisation de la diffusion dans un réseau radio, il pourra être plus pertinent de considérer un coût par routeur participant à la diffusion plutôt que par lien. En effet, dans un réseau radio, un routeur peut transmettre un message à tous ses voisins par une seule émission. Ce problème consiste à construire un ensemble connexe *dominant* de poids minimal, c'est-à-dire un ensemble de sommets E qui forme un sous-

[3] Plus précisément, décider si un graphe possède un k-couvrant de poids inférieur à une borne donnée est NP-complet (voir le chapitre ££Modèles de machines££).

graphe connexe tel que tout sommet du graphe est soit dans E, soit voisin d'un sommet de E, et tel que la somme des coûts associés aux sommets de E soit minimale. Ce problème NP-difficile s'apparente au problème de la couverture de sommet (« *set-cover* » en anglais) pour lequel il existe des heuristiques permettant de s'approcher à un facteur $\log n$ de l'optimal. Un résultat de complexité remarquable indique que trouver une heuristique avec un facteur d'approximation meilleur que $c \log n$ (pour un certain $c > 0$) est en soi un problème NP-difficile.

Un cas plus général de diffusion consiste à envoyer un message à un sous-ensemble de routeurs du réseau. Il s'agit des communications multipoints (ou « *multicast* » en anglais). Une adresse IP peut permettre d'identifier un groupe multicast. Le problème du routage multipoint consiste à acheminer un paquet ayant une telle adresse pour destination à tous les routeurs membres du groupe. Du point de vue de l'algorithmique des graphes, la structure optimale pour effectuer une telle diffusion restreinte consiste à calculer un arbre de *Steiner*, c'est-à-dire un arbre de poids minimal qui contient tous les membre du groupe, plus éventuellement d'autres. Ce problème NP-difficile admet des heuristiques approchant l'optimal à un facteur constant près en temps polynomial. Cependant, les protocoles de routage multipoint proposés en pratique pour les réseaux ne se confrontent pas à ce problème d'optimisation. En effet, ils reposent sur l'utilisation des tables de routage point à point et construisent donc des arbres de diffusion par union de plus courts chemins.

Dans le cas d'un réseau de taille raisonnable, la solution généralement adoptée consiste à construire un arbre de diffusion pour chaque source en faisant l'union des plus courts chemins de la source à chaque membre du groupe. Avec un protocole à états de liens, un routeur intermédiaire peut construire cet arbre en reproduisant le calcul de plus courts chemins qu'effectue la source. Avec un protocole de type vecteur de distances, la technique du « *reverse path forwarding* » permet d'optimiser l'algorithme d'inondation de manière à diffuser selon un arbre : un routeur ne retransmet un message multipoint d'une source s que s'il provient du routeur inscrit dans sa table de routage pour atteindre s par un plus court chemin. Des messages de contrôle sont de plus échangés pour élaguer les branches inutiles.

Dans le cas d'un multicast à grande échelle, un point de rendez-vous, le créateur du groupe par exemple, est utilisé pour construire un unique arbre qui sert à la diffusion dans le groupe quelle que soit la source. L'arbre est constitué de l'union des plus courts chemins de chaque membre du groupe au point de rendez-vous. Pour envoyer un paquet aux membres du groupe, il suffit de l'envoyer vers le point de rendez-vous. Dès qu'un nœud de l'arbre du groupe est rencontré, l'algorithme d'inondation optimisé est utilisé sur les liens de l'arbre. Cet algorithme peut être optimisé en choisissant comme point de rendez-vous le membre du groupe qui offre le meilleur arbre, mais un compromis doit être trouvé pour ne pas changer trop souvent de point de rendez-vous.

5 Flots et trafic

Dans les réseaux haut débit de haut niveau dans l'Internet, les trafics sont agrégés dans des flots statistiques. Un flot donné peut alors être acheminé selon plusieurs chemins dans le graphe du réseau. Cette approche initiée par Lestor Ford et Delbert Fulkerson (1962) permet de modéliser de nombreux problèmes algorithmiques de graphes et de réseaux. Nous supposerons ici que chaque arc du réseau possède une *capacité* $c(u, v) \geq 0$. Une source s de trafic et un puits t sont donnés. Le réseau sera supposé connexe (ce qui implique $m \geq n$). Un *flot* est une fonction f qui à tout couple de sommets u, v associe une quantité de trafic $f(u, v)$ transitant de u à v respectant les contraintes suivantes :

- *contrainte de capacité* : pour tout couple de sommets u et v ; $f(u, v) \leq c(u, v)$ ($c(u, v) = 0$ si l'arc (u, v) n'existe pas).
- *contrainte de symétrie* : pour tout couple de sommets u et v, $f(u, v) = -f(v, u)$ (le signe indique dans quel sens transite le trafic entre deux sommets) ;
- *conservation du flot* : pour tout sommet u différent de s et t, $\sum_v f(u, v) = 0$ (le flot entrant au sommet est égal au flot sortant).

La valeur du flot est le flot sortant à la source $|f| = \sum_v f(s, v)$. On verra que $|f|$ est aussi le flot entrant au puits.

Le problème du *flot maximal* consiste à trouver un flot de valeur maximale. L'algorithme de Ford-Fulkerson résout ce problème lorsque les capacités et les valeurs de f sont entières : il part d'un flot nul qui est augmenté selon des chemins trouvés incrémentalement. Pour cela, il faut définir le *réseau résiduel* associé à un flot f comme le réseau de mêmes sommets ayant un arc de capacité $c_f(u, v) = c(u, v) - f(u, v)$ pour tout couple de sommets u et v. Un chemin augmentant p est un chemin de s à t utilisant des arcs de capacité strictement positive dans le réseau résiduel. La capacité résiduelle $c_f(p)$ d'un chemin p est définie comme la plus petite capacité résiduelle des arcs de p. L'algorithme de Ford-Fulkerson (1962) peut alors s'exprimer ainsi :

```
fordfulkerson() {          /* Calcul d'un flot
maximal. */
  initialiser un flot f à zéro.
  tant que il existe un chemin augmentant p
  dans le réseau résiduel associé à f faire
    pour chaque arc (u, v) de p faire
      f(u, v) = f(u, v) + c_f(p)
      f(v, u) = f(v, u) − c_f(p)
      mettre à jour les capacités résiduelles
      de (u, v) et (v, u).
    fin pour
  fin tant que
  renvoyer f.
}
```

En utilisant un algorithme linéaire de parcours pour trouver un chemin augmentant, la complexité de cet algorithme est $O(m|f^*|)$ où m est le nombre d'arcs dans le réseau et $|f^*|$ la valeur d'un flot maximal. L'algorithme d'Edmonds-Karp utilise plus spécifiquement un parcours en largeur pour trouver un chemin augmentant. On peut alors prouver que la complexité de l'algorithme est en $O(nm^2)$. De nombreuses améliorations de cet algorithme ont été proposées jusqu'à l'algorithme de Goldberg et Tarjan dont la complexité est en $O(nm \log(n^2/m))$.

Une notion fondamentale concernant les flots est celle de coupe. Une *coupe* est une partition des sommets en deux ensembles disjoints X et $\overline{X}$ telle que $s \in X$ et $t \in \overline{X}$. On définit la capacité d'une coupe comme $c(X, \overline{X}) = \sum_{u \in X, v \in \overline{X}} c(u, v)$. Une *coupe minimale* est une coupe de capacité minimale. Étant donné un flot f, le flot à travers la coupe est $f(X, \overline{X}) = \sum_{u \in X, v \in \overline{X}} f(u, v)$. On constate facilement que le flot à travers une coupe est toujours égal à la valeur du flot. En effet, la contrainte de symétrie implique $\sum_{u \in X, v \in X} f(u, v) = 0$ d'où $f(X, \overline{X}) = \sum_{u \in X, v} f(u, v)$, or l'équation de conservation implique $\sum_{u \in X, v} f(u, v) = \sum_v f(s, v) = |f|$. Ceci permet entre autre de montrer que le flot entrant au puits vaut aussi $|f|$. De plus, ce résultat permet d'affirmer que la valeur d'un flot maximal est inférieure à la capacité d'une coupe minimale. Le théorème du flot maximal et de la coupe minimale établit que ces deux quantités sont en fait toujours égales (Ford et Fulkerson, 1962).

Ce résultat permet d'établir la validité de l'algorithme de Ford-Fulkerson puisqu'il implique que le graphe résiduel d'un flot contient un chemin augmentant si et seulement si le flot n'est pas maximal : si le flot est maximal, il existe une coupe de capacité résiduelle nulle, interdisant ainsi l'existence d'un chemin augmentant ; et sinon toute coupe a une capacité résiduelle strictement positive, ce qui implique l'existence d'un plus court chemin augmentant. Il est intéressant de noter que la donnée d'un flot maximal permet de calculer une coupe $X, \overline{X}$ minimale en temps linéaire $O(m)$. Il suffit pour cela de définir X comme l'ensemble des sommets atteignables depuis s dans le graphe résiduel et $\overline{X}$ comme son complémentaire.

Le problème d'un administrateur de réseau de haut niveau est plus complexe : il doit traiter une matrice de flots entre ses divers points de connexions avec d'autres réseaux. Il s'agit donc d'acheminer plusieurs flots distincts par le même réseau. Ce problème est NP-difficile si les capacités et les valeurs des flots sont entières. L'algorithme ci-dessus permet en effet de décider s'il est possible d'acheminer une certaine quantité de flot entre deux points du réseau : il faut que cette quantité soit plus petite que la valeur flot maximal. En revanche, décider si deux flots distincts (ou plus) sont transportables en même temps dans le réseau est NP-complet. Un problème connexe consiste à concevoir un réseau de coût minimal pour pouvoir acheminer une matrice de trafic donnée. Les problèmes de ce type sont généralement résolus avec des méthodes numériques par une approche de programmation linéaire.

6 Coloration, routage optique et allocation de fréquences

De nombreux problèmes d'optimisation de réseau se rapprochent du problème de la coloration d'un graphe. Une *coloration* d'un graphe consiste à attribuer une couleur à chaque sommet de sorte que deux sommets reliés par un lien n'ont jamais la même couleur. Si d est le degré maximal d'un sommet, une heuristique très simple permet de colorier les sommets avec $d + 1$ couleurs :

```
coloration() {  /* Coloration d'un graphe */
  pour chaque sommet u du graphe pris dans
  un ordre quelconque faire
    attribuer à u la plus petite couleur non
    attribuée à l'un de ses voisins.
  fin pour
}
```

Comme un sommet a au plus d voisins, $d + 1$ couleurs suffisent forcément. Trouver une coloration minimale, c'est-à-dire utilisant un minimum de couleurs est un problème NP-difficile. Il est même difficile de trouver des solutions approchées. Colorier un graphe à n sommets avec moins de n^c couleurs pour un certain $c < 1/7$ est en effet un problème NP-difficile.

Les réseaux de très haut débit utilisent actuellement une technologie à base de fibre optique, ce sont les réseaux optiques. Pour obtenir de plus grands débits, ils utilisent le multiplexage en longueur d'onde qui consiste à transmettre plusieurs signaux sur une même fibre. Pour éviter les

brouillages entre signaux, il est nécessaire d'utiliser des longueurs d'onde différentes. Un administrateur de réseau optique voit donc se rajouter au problème de l'acheminement d'une matrice de flots celui de l'attribution des longueurs d'onde aux chemins empruntés par chaque flot. Si l'administrateur commence par résoudre le problème du routage, c'est-à-dire le découpage de chaque flot en un ensemble de chemins permettant de le transporter, il reste à résoudre un problème de coloration : assigner une longueur d'onde à chaque chemin de sorte que deux chemins de même longueur d'onde ne passent jamais par le même arc du réseau. Ce problème est exactement celui de la coloration dans le graphe dont les sommets sont les chemins obtenus par la résolution du problème de routage et où deux chemins sont reliés s'ils empruntent un même arc du réseau. Il est à noter qu'une solution technologique est venue résoudre le problème avec l'apparition de routeurs optiques capables de changer la longueur d'onde d'un signal entrant sur un lien avant de le retransmettre sur un autre lien.

Les réseaux radio offrent eux aussi des problèmes d'assignation de longueur d'onde. La téléphonie cellulaire recouvre le territoire par des cellules au centre desquelles se trouvent des stations de base. Un téléphone mobile communique ainsi par une liaison radio avec la station de base de la cellule dans laquelle il se trouve, le reste de la transmission s'effectuant par un réseau filaire classique. Une station de base utilise plusieurs canaux de fréquences radio différentes afin de gérer plusieurs mobiles simultanément : il s'agit du multiplexage en fréquence utilisé par le GSM par exemple. Pour éviter les interférences entre cellules, les ensembles de canaux réservés pour des cellules proches doivent être disjoints. Ce problème est sensiblement plus complexe que celui de la coloration d'un graphe pour deux raisons. Tout d'abord, l'exclusion est plus forte pour des cellules plus proches : la règle généralement adoptée consiste à utiliser des canaux séparés par au moins un canal intermédiaire entre deux cellules contiguës et des canaux simplement distincts pour deux cellules séparées par une cellule. Deux cellules plus éloignées peuvent utiliser les même canaux. Ces règles sont bien sûr une simplification de la réalité physique du phénomène d'interférences radio. D'autre part, il faut allouer un certain nombre de canaux par cellule et non une seule couleur. Notons cependant que l'heuristique de coloration proposée plus haut se généralise facilement dans ce contexte.

Si l'algorithmique des graphes constitue un outil indispensable pour les réseaux, des ajustements sont bien souvent nécessaires pour s'intégrer au contexte spécifique d'un problème donné.

7 Pour en savoir plus

La plupart des algorithmes de graphe décrits dans ce chapitre sont présentés en détail dans la copieuse introduction à l'algorithmique de Cormen, Leiserson et Rivest (1994). L'ouvrage de Bertsekas et Gallager (1992) fournit une approche algorithmique assez poussée des réseaux. Le panorama très général proposé par Tanenbaum (2003) fournit une introduction plus abordable au monde des réseaux. Les résultats d'inapproximabilité des problèmes difficiles d'algorithmique des graphes sont tirés du livre de Ausiello, Crescenzi, Gambosi, Kann, Marchetti-Spaccamela et Protasi (1999). Les auteurs tiennent à jour un site Internet avec les résultats les plus récents en la matière. Le livre de Vazirani (2001) fournit une approche plus algorithmique des problèmes d'approximation.

Bibliographie

Ausiello G., Crescenzi P., Gambosi G., Kann V., Marchetti-Spaccamela A. et Protasi M. (1999), *Complexity and Approximation, Combinatorial Optimization Problems and their Approximability Properties*, Springer Verlag.

Bertsekas D. et Gallager R. (1992) *Data Networks*, Prentice Hall.

Cormen T., Leiserson C. et Rivest R. (1994) *Introduction à l'algorithmique*, Paris, Dunod, traduit de l'américain par X. Cazin de *Introduction to Algorithms*, MIT Press, 1990.

Ford L. et Fulkerson D. (1962), *Flows in Networks*, Princeton University Press.

Peleg D. et Schäffer A. (1989) "Graph spanners", *Journal of Graph Theory*, Volume 13, p.99-116.

Tanenbaum A. (2003), *Réseaux*, Paris, Pearson Education, traduit de l'anglais par F. Sofor, V. Warion et M. Dreyfus de *Computer Networks*, Prentice Hall, 1996.

Tarjan R. (1972), "Depth first search and linear graph algorithms", *SIAM Journal of Computing*, Volume 1, Number 2, p.146-160.

Vazirani V. (2001), *Approximation Algorithms*, Springer Verlag, Berlin.

Index

algorithme
    de graphes, 1–10
    de réseaux, 1–10

BGP, 6

graphe
    algorithmes de, 1–10
    coloration, 9
    composante connexe, 2
    connexe, 2
    connexité, 2
    flot maximal, 8
    modélisation de réseaux, 1
    non orientés, 1
    parcours en largeur, 3, 4
    parcours en profondeur, 3
    spanner, 7
    symétriques, 1

réseau
    algorithmes de, 1–10
    modélisation par des graphes, 1