Université Du Havre, Master : Mathématiques-Informatique (Matis)

MIS7 : Modèles discrets: évolution, auto-organisation

2010/2011

Nakechbandi M.

moustafa.nakechbandi@univ-lehavre.fr

Chapitre 1
Algèbre de chemins

0. Introduction

Le problème du plus court chemin, consistant, étant donné un ensemble de sommets reliés par des arêtes pondérées par leur « coût de franchissement », à trouver un chemin le moins coûteux possible entredeux d'entre eux, est fondamental en théorie des graphes.

Plusieurs algorithmes classiques permettent d'y répondre efficacement, mais ils se contentent en général de traiter le cas où le « coût de franchissement » des arêtes est un réel ou un réel positif. Or, dans de nombreuses situations concrètes, on voudrait déterminer un plus court chemin alors que ces « coûts » dépendent du temps ou bien des déplacements précédents (par exemple, dans un réseau de transports en commun où les déplacements sont conditionnés par des horaires).

Le but de cet chapitre est de montrer qu'on peut généraliser les algorithmes classiques de plus court chemin en exprimant ce problème comme un système linéaire par changement d'algèbre sous-jacente : on introduit pour cela une structures algébriques dite Algèbre de chemins, obtenues par substitution des opérations usuelles (la somme et le produit) par des lois aux propriétés plus faibles (en particulier min et max, pour lesquelles les réels ne disposent pas de symétriques), mais permettant de définir les opérateurs linéaires.

1. Semi-anneau idempotent (ou dioïde idempotent)

Définition 1 : Un ensemble K muni de deux opérations binaires associatives $\oplus$ et $\otimes$ et de deux éléments distingues $\varepsilon(0)$ et $e(1)$ est un semi-anneau si

1. $(K, \oplus, 0)$ est un monoïde commutatif, c'est à dire $a \oplus b = b \oplus a$ pour $a, b \in K$ et $a \oplus 0 = 0 \oplus a = a$ pour tout a dans K ;

2. $(K, \otimes, 1)$ est un monoïde et $a \otimes 1 = 1 \otimes a = a$ pour tout a dans K ;

3. l'opération $\otimes$ est distributive par rapport à $\oplus$, est à dire :

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$$

et $(a \oplus b) \otimes c = (a \otimes c) \oplus (a \otimes b)$;

4. l'élément 0 est un absorbant pour l'opération : $0 \otimes a = a \otimes 0 = 0$.

On note $(K, \oplus, \otimes, 0, 1)$ un semi-anneau.

Exemple 1 : Les structures $(N, +, *, 0, 1)$,
$(N, \min, +, \infty, 0)$,
$(N, \max, +, -\infty, 0)$

sont des exemples triviaux de semi-anneau par contre (ensemble des mots sur un alphabet, union, concaténation, Ø, le mot vide) est un autre exemple moins trivial.

Le tableau de annexe B donne d'autres exemples **classiques d'algèbre** de chemin.

Définition 2 (**Algèbres d'endomorphismes**).
On suppose que (K, $\oplus$, $\otimes$, ε, e) est un semi-anneau. On définit alors F comme ensemble des applications $f : K \to K$ qui vérifient les propriétés suivantes :

(i) $f(a \oplus b) = f(a) \oplus f(b) \quad \forall a, b \in K, f \in F.$

(ii) $f(\varepsilon) = \varepsilon \quad \forall f \in F.$

On munit F de la loi $\oplus$, définie par : $\forall f,g \in F, \forall x \in K,$
$(f \oplus g)(x) = f(x) \oplus g(x)$ dont le neutre est l'application $f^{\varepsilon}: x \to \varepsilon$.
Dans la suite f^{ε} sera notée 0.

Enfin, on utilise la composition $\circ$ (de fonction) comme seconde loi(produit)

$h \otimes g = h \circ g$

qui est associatif mais non commutatif, dont l'élément neutre est l'application identité(unité) noté I.

Théorème : $(F, \oplus, \otimes, 0, I)$ est un semi-anneau.

La démonstration s'appuie sur le fait que l'élément neutre f^{ε} de $\oplus$ est un élément absorbant pour l'opération $\otimes$, car $(g \otimes f^{\varepsilon})(a) = f^{\varepsilon}(g(a)) = \varepsilon \Rightarrow g \otimes f^{\varepsilon} = f^{\varepsilon}$

La classe précédente de Semi-anneau est une classe très importante permettant l'étude de très nombreux problèmes, en particulier des problèmes de cheminements complexes comme le plus court chemin avec **longueur** des arcs **dépendant du temps** cité plus haut, ou le problème de cheminement sous **contraintes sur les arcs**, ou le plus courts chemins dans les graphes probabiliste(au lieu d'associé aux arcs des valeurs certaines, on associe une variable aléatoire.

Dans la suite, on se place dans un Semi-anneau $(\mathbf{K},\oplus,\otimes,\varepsilon(0),e(1))$. On donne une définition «valuation » d'un graphe :

Définition 3 (**graphe**). Un graphe $G=(S,\ \delta)$ est la donnée d'un ensemble fini de sommets $S=\{1,...,n\}$ et d'une application de valuation $\delta : S \times S \rightarrow \mathbf{K}$, telle que $\delta(i,j)$ représente la valuation de l'arc reliant i à j ou 0 si un tel arc n'existe pas. Pour simplifier, on prendra souvent $\delta(i,i)=0$, ce qui intuitivement signifie que rester sur le sommet i ne « coûte » rien.

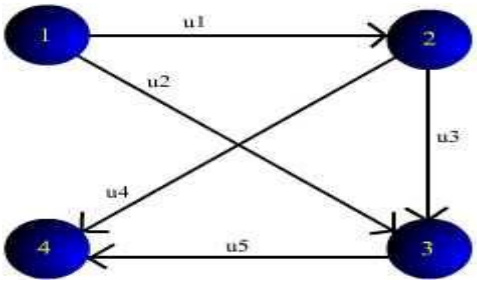
Cette définition met en évidence l'association canonique entre un graphe et sa matrice d'adjacence généralisée :

Définition 4 (**matrice d'adjacence**). On appelle matrice d'adjacence la matrice $A=(a_{i,j})$; $(i,j)\in[1..n]^2$ définie par $a_{i,j}=\delta(i,\ j)$;

Réciproquement, soit $A \in M_n(K)$ une matrice carré $n\times n$ à coefficients dans un semi-anneau K, on note $G(A)$ le graphe orienté valué associé à A.

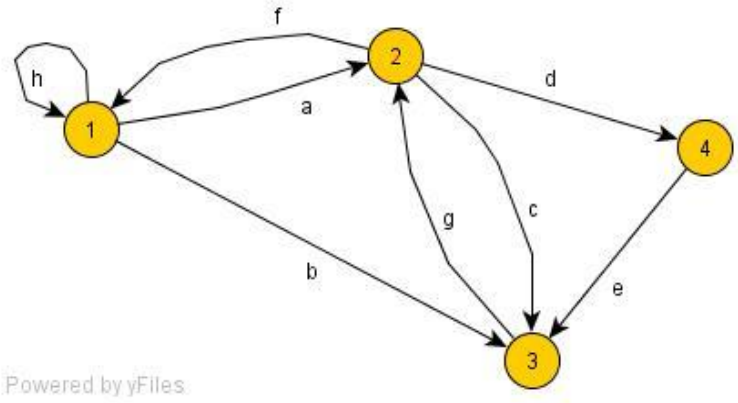
$a_{i,j}$ est donc un libellé associé à l'arc ij correspondant à la **distance** entre i et j ; $a_{i,j}=0$ s'il n'existe pas d'arc reliant i à j.

Exemple 2 : $K=(0,\ 1,\ et,\ ou)$ est ici l'algèbre est l' algèbre de Boole.



i \ u	1	2	3	4
1	0	1	0	1
2	0	0	1	1
3	0	0	0	0
4	0	0	1	0

Exemple 3 : On considère le graphe ci-dessous :



La matrice d' adjacence A associée à ce graphe est :

h	a	b	0
f	0	c	d
0	g	0	0
0	0	e	0

On définie l'addition et la multiplication des matrices à partir des lois ⊕,⊗.

Ainsi pour $A=(a_{i,j})$, $B=(b_{i,j})$:

$$C = A \oplus B = (c_{i,j}) \quad \Leftrightarrow \quad c_{i,j} = a_{i,j} \oplus b_{i,j}$$

et $C = A \otimes B = (c_{i,j}) \quad \Leftrightarrow \quad c_{i,j} = \sum a_{i,k} \otimes b_{k,j}$

L'élément neutre pour l'opération ⊕ matricielle est la matrice (0) (zéro par tout) et

l'élément neutre pour opération ⊗ matricielle l'élément neutre est
$E = (e_{i,j}) \mid e_{i,j} = 0$ si $i \neq j$ et $e_{i,i} = 1$ au diagonal.
E est donc la matrice d'identité de $M_n(K)$.

On notera $A^0 = E$.

2. Calcul de chemin

2.1 Poids d'un chemin

Par définition **le poids d'un chemin** dans le graphe est la multiplication des poids de ses arcs.

Le problème du plus court chemin consiste à calculer, pour tout sommet de départ i et tout sommet d'arrivée j, la somme (si elle existe) des poids des chemins reliant i à j. Ceci correspondent à l'idée intuitive que l'on a d'un plus court chemin dans l'algèbre(min,+) : c'est le minimum pris sur les chemins de la somme des valuations des arcs empruntés.

Par exemple, si l'on considère le graphe de l 'exemple 3. Soit x le poids du chemin (1, 2, 4, 3) ;

$x = a \otimes d \otimes e = a + d + e$. Soit y le poids du chemin (1, 2, 3) ;
$y = a \otimes c = a+c$
$x \oplus y = \min(x, y) = \min((a + d + e), (a + c))$

On remarque que la longueur du plus court chemin entre 1 et 3 correspond à la somme des poids des chemins possibles entre 1 et 3.

Comme on a supposé la somme sélective (la valeur d'une somme est égale à un de ses termes), on cherche aussi un chemin dont le poids est minimum.

2.2 Puissances de la matrice d'adjacence

Montrons à présent que la résolution de problème de calcul de chemin est équivalente à un calcul de puissance de la matrice d'adjacence.

Théorème (**interprétation du produit matriciel**) : Chaque terme (i, j) de la $k^{ième}$ puissance de la matrice A associée au graphe G est la somme des poids des

chemins reliant i à j en k arcs exactement.

Exemple 4 (d'Illustration) **:** On ici considère les données de l'exemple 3
Le calcul de la première ligne de A^1 , A^2 , A^3 donne :

A_1^1 = (h, a, b, 0)

A_1^2 = (h + af, bg, Ac, Ad)

A_1^3 = (h + af + bgf, acg, ade, Bgd)

Exercice 1 : calculer A_1^4

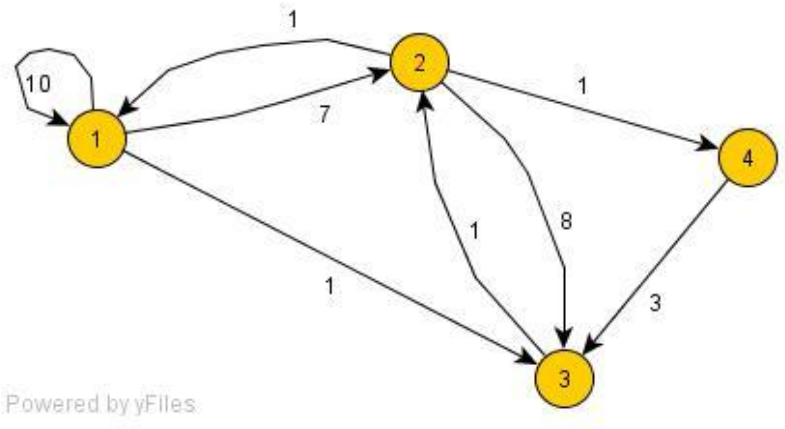
A_1^4=

Lemme : La somme $A^{(k)} = E \oplus A^1 \oplus A^2 \ \ldots \ \oplus A^k$ contient la somme des poids de tous les chemins dont la longueur en nombre d'arcs est $\leq k$. (par exemple dans le cas de problème de plus court chemin $A^{(k)}$ contient les valeurs des plus court chemin de k arcs au maximum).

Exercice 2 : Calculer $A_1^{(3)} = E \oplus A_1^1 \oplus A_1^2 \oplus A_1^3 \oplus A_1^4$

E_1	1	0	0	0
$E_1 \oplus A_1^1$	h	a	b	0
$E_1 \oplus A_1^1 \oplus A_1^2$	h + af	a + bg	b + ac	ad
$E_1 \oplus A_1^1 \oplus A_1^2 \oplus A_1^3$	h + af + bgf	acg + a + bg	ac + b + ade	ad + bgd
$E_1 \oplus A_1^1 \oplus A_1^2 \oplus A_1^3 \oplus A_1^4$				

Exemple 5 :
0n considère un problème de plus court chemin dont le graphe de celui de l'exemple 3 avec les distances suivantes :
h=10, a=7, b=1, f=1, c=8, d=1, g=1, e=3.



Dans ce cas les élément neutres sont e = ∞, ε = 0 et la matrice A est :

10	7	1	∞
1	∞	8	1
∞	1	∞	∞
∞	∞	3	∞

Le calcul de $A_1^{(4)}$ donne :

$E_1 = (0 \quad \infty \quad \infty \quad \infty)$

$E_1 \oplus A_1^1 = (10 \quad 7 \quad 1 \quad \infty)$

$E_1 \oplus A_1^1 \oplus A_1^2 = (8 \quad 2 \quad 1 \quad 8)$

$E_1 \oplus A_1^1 \oplus A_1^2 \oplus A_1^3 = (3 \quad 2 \quad 1 \quad 3)$

$E \oplus A_1^1 \oplus A_1^2 \oplus A_1^3 \oplus A_1^4 = (3 \quad 2 \quad 1 \quad 3)$

On note $A^* = \sum A^{(k)} \mid k = 1..\infty$

(les sommations ci-dessus s'entendent sens de $\oplus$)

Dans l'absence d'un circuit absorbant (ou négatif) on peut démontrer que

$A^* = \sum A^i \mid i=0..n$; où n est le nombre de sommets.

Un grand nombre de problème de cheminement se ramène au calcul de **A^* (cf. annexe)**

2.3 Équation de point fixe.

On peut facilement démontrer que **$A^* = A \otimes A^* \oplus E$**

En effet, $A \otimes A^* \oplus E = A \otimes (A_1^0 \oplus A_1^1 \oplus ... \oplus A_1^n) \oplus E$

$= A_1^1 \oplus A_1^2 \oplus .. A_1^n \oplus A_1^{n+1} \oplus E$

$= A_1^1 \oplus A_1^2 \oplus .. A_1^n \oplus E$

$= A^*$

La recherche de A^* revient donc à la résolution d'un système d'équations

$$\mathbf{X = A \otimes X \oplus E} \quad \text{(ou de } Y = Y \otimes A \oplus E^T\text{)}.$$

Ainsi la plus part des algorithmes de résolution du problème de cheminement peut s'interpréter comme des variantes de méthodes connues en algèbre linéaire classique.

3. Algorithmes de résolution

Les structures algébriques étudiées plus haut vont permettre de définir des variantes de la plus part des algorithmes classique de résolution de systèmes linéaires (Algorithme de Jacobi, de Gauss-Seidel, de Gauss, etc…).

Ces structure auront également un rôle unificateur considérable puis'un même algorithme permettra alors de résoudre plusieurs dizaines de problèmes de cheminement de nature différents.

3.1 Algorithmes directs

Les conditions d'existence de A^* ne font pas partie de ce cours. On suppose dans la suite que **A^* existe** et que $A^* = A^{(n)}$. C'est à dire le graphe ne contient pas de circuits absorbants (négatif)

3.1.1 Approche naïve (calcul directe)

Une manière évidente de calculer **A^*** est de procéder par somme et produits matriciels successifs. Comme chaque produit matriciel nécessite $O(n^3)$ opérations $\oplus$ et $\otimes$, cette méthode a une complexité en $O(n^4)$.

On peut améliorer cette méthode de deux façons : d'une part en utilisant un algorithme d'exponentiation rapide, et d'autre part en utilisant un algorithme de produit matriciel rapide. De même, si l'on ne cherche qu'une ligne i de **A^*** (plus court chemin à origine unique), on peut la construire itérativement en $O(n3)$ par la récurrence (Voir l'exemple 4).

Cet approche est connu également sous le nom d'algorithme Dantzig

3.1.2. Méthode de Gauss-Jordan

Connu également sous le nom **Algorithme Warshall-Roi** ou encore **Algorithme Floyd-Warshall.**

```
procedure Floyd_Warshall (A : matrice n × n)
W = A
for k := 1 to n
    for I := 1 to n
        for j := 1 to n
            W_ij = W_ij ⊕ W_ik ⊗ W_kj ;
```

La complexité en temps de cet algorithme est O(n3).

Exemple 6 (d'illustration) **:** 0n considère le graphe de l'exemple 5

$W^{(0)} = A$

$W^{(o)} =$

10	7	1	∞
1	∞	8	1
∞	1	∞	∞
∞	∞	3	∞

Rappelons que + est l'opération min
. est l'addition réelle

k = 1 i = 1 pivot $W_{ik}^{(o)} = 10$

$W_{lj}^{(1)} = W_{lj}^{(o)} + W_{ll}^{(o)} W_{lj}^{(o)}$

Pas de changement à cause de l'absorption

i = 2 pivot $W_{ik}^{(o)} = 1$

2ème ligne = min (2ème ligne, 1 + 1ère ligne)

i = 3 pivot $W_{ik}^{(o)} = \infty$: pas de changement

i = 4 pivot $W_{ik}^{(o)} = \infty$: pas de changement

$W^{(1)} =$

10	7	1	∞
1	8	2	1
∞	1	∞	∞
∞	∞	3	∞

On a obtenu les plus courtes distances, en passant éventuellement par 1.

$k = 2$ $i = 1$ pivot $W_{ik}^{(1)} = 7$

lère ligne = min (lère ligne, 7 + 2ème ligne)

$i = 2$ Pas de changement si $i = k$

$i = 3$ pivot $W_{ik}^{(1)} = 1$

3ème ligne = min (3ème ligne, 1 + 2ème ligne)

$i = 4$ pivot $W_{ik}^{(1)} = \infty$: pas de changement

$W^{(2)} =$

8	7	1	8
1	8	2	1
2	1	3	2
∞	∞	3	∞

$k = 3$ $i = 1$ pivot $W_{ik}^{(2)} = 1$

lère ligne = min (lère ligne, 1 + 3ème ligne)

$i = 2$ pivot $W_{ik}^{(2)} = 2$

2ème ligne = min (2ème ligne, 2 + 3ème ligne)

$i = 3$ pas de changement si $i = k$

$i = 4$ pivot $W_{ik}^{(2)} = 3$

4ème ligne = min (4ème ligne, 3 + 3ème ligne)

$W^{(3)}$ =

3	2	1	3
1	3	2	1
2	1	3	2
5	4	3	5

k = 4 i = 1 pivot $W^{(3)}_{ik}$ = 3

1ère ligne = min (1ère ligne, 3 + 4ème ligne)

i = 2 pivot $W^{(3)}_{ik}$ = 1

2ème ligne = min (2ème ligne, 1 + 4ème ligne)

i = 4 pas de changement si i = k

$W^{(4)}$ =

3	2	1	3
1	3	2	1
2	1	3	2
5	4	3	5

3.1.3. Algorithme de Dijkstra

On cherche ici à calculer la ligne A^*_r de A* d'indice r quelconque (r ϵ [1..n])

a) Initialisation **:**

Poser $\pi(r) \leftarrow e$;

et $\pi(i) \leftarrow \varepsilon$ pour $i \in [1..n] \setminus \{r\}$;

$T \leftarrow \{1,2,\ldots, n\}$;

b) étape courante :

b1) Déterminer l'indice $i \in T$ par :

$$\pi(i) = \sum_{j \in T} \pi(j)$$

puis faire $T \leftarrow T \setminus \{i\}$;

Si $T = \Phi$, fin de l'algorithme /* le vecteur $(\pi(1), \pi(2), \ldots, \pi(n))$ est /* la $r^{ème}$ ligne de **A***

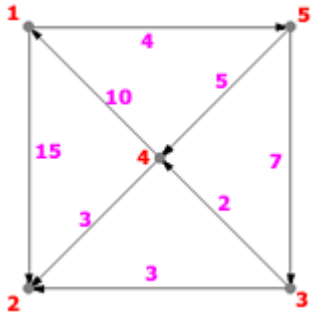
Si $T \neq \Phi$ aller en (b2) ;

b2) Pour tout $j \in T$ faire

$\pi(j) = \pi(j) \oplus \pi(i) \otimes a_{ij}$;

et retourner en (b)

Exercice 3 : Soit le graphe



la matrice A correspondant est :

$$\begin{matrix} 0 & 15 & \infty & \infty & 4 \\ \infty & 0 & \infty & \infty & \infty \\ \infty & 3 & 0 & 2 & \infty \\ 10 & 3 & \infty & 0 & \infty \\ \infty & \infty & 7 & 5 & 0 \end{matrix}$$

Appliquer l'algorithme de Dijkstra pour calculer $\mathbf{A}^*_1$, vérifier que la solution est : $\mathbf{A}^*_1 = (0, 12, 11, 9, 4)$;

Remarque : Dans l'algèbre (min,+), les algorithmes classiques vus précédemment calculent les poids des plus courts chemins, mais aussi, lors des sommes, l'indice minimal permettant effectivement de **construire** un plus court chemin :
Par exemple dans l'algorithme de Dijkstra, il suffi d'utiliser un tableau correspondant au prédécesseur de chaque sommet dans le meilleur parcourt, ce tableau est initialisé à NIL (p(j)=NIL, pour 1 ≤j ≤n) puis mettre ce tableau à jour dans la partie (b2). On obtient ainsi une nouvelle version de Dijkstra :
Une nouvelle version de l'Algorithme de Dijkstra permettant de construire le meilleur chemin :

a) Initialisation :

Poser $\pi(r) \leftarrow e$;

et $\pi(i) \leftarrow \varepsilon$ pour $i \in [1..n]\backslash\{r\}$;

$T \leftarrow \{1,2,\ldots, n\}$;

Pour $j \in [1..n]$

$p(j) \leftarrow$ nil ;

b) étape courante :

b1) Déterminer l'indice $i \in T$ par :

$$\pi(i) = \sum_{j \in T} \pi(j)$$

puis faire $T \leftarrow T\backslash\{i\}$;

Si $T = \Phi$, fin de l'algorithme

Si $T \neq \Phi$ aller en (b2) ;

b2) Pour tout $j \in T$ faire

$x = \pi(j)$;

$\pi(j) = \pi(j) \oplus \pi(i) \otimes a_{ij}$;

si $\pi(j) \neq x$ alors $p(j) \leftarrow i$;

et retourner en (b) ;

Exercice 4

Dans l'exemple de l'exercice n° 3 précédent vérifier que l'application de la nouvelle version de Dijkstra donne : `p = (NIL, 4, 5, 5, 1).`

Par exemple le chemin minimal de `1` à `4` par exemple est de coût `9`. C'est le chemin `1-5-4`, car `p(4)=5` et `p(5)=1`.

Exercice 5

Démontrer que pour un graphe à n sommets, l'algorithme de Dijkstra s'exécute en $O(n^2)$ opérations.

3.2 Algorithmes itératifs

On suppose dans la suite qu'il existe un `K` tel que $\mathbf{A^* = A^{(K)} = A^{(K+1)} = \ldots}$

3.2.1/ Algorithme de Jacobi généralisé

On cherche ici à calculer la première ligne de la matrice $\mathbf{A^*}$ (correspondant à la meilleur distance entre le sommet 1 et les autres sommets).

(a) poser $y^0 = b^T = (e, \varepsilon, \ldots \varepsilon)$; $t \leftarrow 0$;

(b) À l'itération t, soit y^t la solution courante. Calculer

$$y^{t+1} = y^t \otimes A \oplus b^T$$

Si $y^{t+1} = y^t$, l'algorithme se termine et $y^t = b^T \otimes A^*$ (la première ligne de A^*).

Si $y^{t+1} \neq y^t$ et $t \leq K$ alors faire $t \leftarrow t+1$ et retourner en (b).

Si $y^{t+1} \neq y^t$ et $t = K$ alors interrompre les calculs : $A^* \neq A^{(K)}$.

L'algorithme précédent est ce n'est pas autre chose que l'**Algorithme Bellman**

3.2.2/ Algorithme de Gauss-Seidel généralisé

Par analogie avec la version classique de l'algorithme de Gauss-Seidel, l'idée consiste à décomposer la matrice A en l'exprimant comme la somme d'une matrice triangulaire inférieure L et d'une matrice triangulaire supérieure U. Pour simplifier la présentation nous supposerons pour commencer que les éléments diagonaux de A sont tous égaux à ε. On a donc :

$$A = U \oplus L$$

L'itération de base de l'algorithme de Jacobi généralisé se met alors sous la forme :

$$y^{t+1} = b^T \oplus y^t \otimes U \oplus y^t \otimes L$$

Ainsi, pour toute composante j = 1,... n, le calcul de y_j^{t+1} se fait par la relation :

$$y_j^{t+1} = b_j \oplus \sum_{i=1}^{j-1} y_i^t \otimes a_{ij} \oplus \sum_{i=j+1}^{n} y_i^t \otimes a_{ij}$$

Lors du calcul de y_j^{t+1}, tous les termes $y_1^{t+1}, y_2^{t+1}, \ldots, y_{j-1}^{t+1}$ ont déjà été déterminés. L'idée de l'algorithme de Gauss-Seidel est alors de remplacer les valeurs y_j^t dans la première sommation par les nouvelles valeurs y_j^{t+1} déjà déterminées, ce qui conduit à la récurrence :

$$y_j^{t+1} = b_j \oplus \sum_{i=1}^{j-1} y_i^{t+1} \otimes a_{ij} \oplus \sum_{i=j+1}^{n} y_i^t \otimes a_{ij}$$

ou encore, sous forme matricielle :

$$y^{t+1} = b^T \oplus y^{t+1} \otimes U \oplus y^t \otimes L$$

Ceci conduit à l'algorithme suivant :

Détermination de la première ligne de A^* ou preuve que $A^{(K)} \neq A^*$

(a) Poser $b^T = (e, \varepsilon, \ldots \varepsilon)^T$ et $y^0 = (\varepsilon, \varepsilon, \varepsilon, \ldots \varepsilon)$; $t \leftarrow 0$;

(b) À l'itération t, soit y^t la solution courante. Calculer :

$$y^{t+1} = b^T \oplus y^{t+1} \otimes U \oplus y^t \otimes L$$

Si $y^{t+1} = y^t$, l'algorithme se termine et $y^t = b^T \otimes A^*$ (la première ligne de A^*).
Si $y^{t+1} \neq y^t$ et $t \leq K$ alors, faire $t \leftarrow t+1$ et retourner en (b).
Si $y^{t+1} \neq y^t$ et $t = K$ alors interrompre les calculs : $A^* \neq A^{(K)}$.

4. Quelques applications

4.1 Plus court (plus long) chemin

Dans l'algèbre (min,+), les algorithmes classiques, vus précédemment, calculent les poids des plus courts (plus long) chemins, mais aussi, lors des sommes, l'indice minimal permettant effectivement de **construire** un plus court chemin. (cf. la nouvelle version de l'algorithme de Dijkstra)

4.2 Accessibilité

Tous ces algorithmes peuvent être mis en œuvre directement dans l'algèbre de Boole $(B=\{0,1\}, \vee, \wedge)$. A^* est alors appellée fermeture transitive du graphe $G(A)$, dont les termes $(A^*)_{i,j}$ valent 1 s'il existe un chemin entre i et j et 0 sinon.

4.3 Fiabilité d'un réseau

On associe à chaque arc un booléenne et une probabilité d'existence de l'arc, on veut déterminer la probabilité que deux sommets soient reliés par un chemin.

4.4 Cheminement avec temps de parcours variables

On a vu que les endomorphismes d'un semi-anneau forment un semi-anneau.

On choisira ici $S=(R^+, \min, +, \infty, 0)$. L'ensemble H sera l'ensemble des endomorphismes h de S vérifiant h (+∞) = + ∞.
Comme ici $\oplus$ = Min, l'adition d'endomorphisme s'écrit :
$h\ (t \oplus t') = h(\min\{t, t'\}) = \min\{h(t), h(t')\} = h(t) \oplus h(t')$

À chaque arc (i,j) du graphe on associe $\varphi_{ij} \in H$ ayant la signification suivante:
$t_j = \varphi_{ij}(t_i)$ est l'instant d'arrivée au sommet j lorsqu'on part de i à l'instant t_j en empruntant l'arc (i,j).

Si l'on part du sommet i_0 choisi comme origine à l'instant t_0 on veut déterminer, pour chaque somme $j \neq i_0$ un chemin optimum permettant d'arriver en j le plus tôt possible. Il s'agit donc de déterminer toutes les valeurs $\varphi^*_{ij}(t_0)$ qui correspondent à la ligne i_0 de la matrice $(\varphi_{ij})^*(t_0)$.

l'exemple d'un **réseau routier** soumis à des **variations de trafic**. À chaque route reliant i à j, on associe une fonction $f(i,j)$ croissante qui à une heure de départ associe une heure d'arrivée. Cette fonction permet de prendre en compte d'éventuelles heures creuses ou de pointe.

Au graphe G, on associe donc une matrice $F=(f_{i,j})$ à valeurs dans un semi-anneau sélectif dans lequel la fonction identité (temps de parcours nul) est minimale pour $\oplus$.

Les algorithmes vue plus haut (paragraphe 3) permettent donc le calcul du temps de cheminement entre deux sommets quelconques, pour une heure de départ $t0$

fixée, et éventuellement de l'itinéraire associé. Cet itinéraire dépend de `t0` : il est possible qu'à certaines heures un itinéraire soit meilleur qu'un autre. On trouve dans [4] quelques exemples d'illustration.

4.5 Plus court chemin avec contraintes sur les arcs.

Chercher un plus court chemin compatible avec des contraintes qui sont définies sur les arcs.

À chaque arc (i, j) d'un graphe G, on associe :
– une durée $d_{ij} > 0$ mesurant le temps de parcours de l'arc (i, j),
– un ensemble d'intervalles $V_{ij} \subset [0, +\infty[$ représentant l'ensemble des instants de départ possibles du sommet i vers le sommet j par l'arc (i, j).

À chaque sommet i, on associe un ensemble d'intervalles $W_i \subset [0, +\infty[$ représentant l'ensemble des instants où le parking est autorisé au sommet i.

Le problème est de rechercher entre deux sommets x et y le plus court chemin (au sens des temps de parcours) compatible avec les contraintes temporelles définies par les V_{ij} (sur les arcs) et les W_i (sur les sommets).

L'information associée à un sommet i est l'ensemble E_i des instants d'arrivée possible en i à partir de l'origine x. Un élément de S sera donc un ensemble d'intervalles $\subset [0, +\infty[$. On définit dans S l'opération $\oplus$ (union de 2 ensembles d'intervalles) par :

$$a \oplus b = \{t/t \in a \text{ ou } t \in b\} \qquad \forall\, a, b \in S.$$

L'ensemble vide ø est l'élément neutre de $\oplus$. Pour définir les endomorphismes φ_{ij}, nous définirons la *transition* entre i et j en plusieurs étapes :
– Si E_i correspond à l'ensemble des instants d'arrivée possible en i, alors l'ensemble D_i des instants de départ possibles de i sera :

$$D_i = E_i \perp W_i$$

où l'opération $\perp$ est définie de la façon suivante :

$$\text{Si} \quad E_i = \left\{[\alpha_1, \alpha_1'], [\alpha_2, \alpha_2'], \ldots, [\alpha_p, \alpha_p']\right\}$$

$$W_i = \left\{[\beta_1, \beta_1'], [\beta_2, \beta_2'], \ldots, [\beta_q, \beta_q']\right\}$$

alors :

$$D_i = \left[\gamma_1, \gamma_1'\right] \oplus \left[\gamma_2, \gamma_2'\right] \oplus \ldots \oplus \left[\gamma_p, \gamma_p'\right]$$

avec pour k de 1 à p :

$$\left[\gamma_k, \gamma_k'\right] = \begin{cases} \left[\alpha_k, \alpha_k'\right] & \text{si} \quad \alpha_k' \notin W_i \\ \left[\alpha_k, \beta_j'\right] & \text{si} \quad \alpha_k' \in \left[\beta_j, \beta_j'\right] \end{cases}$$

– L'ensemble des instants de départ possibles de i vers j par l'arc (i, j) sera alors : $D_i \cap V_{ij}$.
– Définissons sur S une opération externe $\top$ (translation) par :

$$a \in S, \tau \in \mathbb{R}^+, \tau \top a = \{t + \tau/t \in a\}.$$

L'ensemble des instants d'arrivée possible en j à partir de i par l'arc (i, j) sera donc :

$$d_{ij} \top (D_i \cap V_{ij}).$$

– Finalement on définira φ_{ij} par :

$$\varphi_{ij}(E_i) = d_{ij} \top [(E_i \perp W_i) \cap V_{ij}].$$

On vérifie que φ_{ij} est un endomorphisme de $(S, \oplus)$.

L'application de l'un des algorithmes du paragraphe 3 dans l'algèbre des endomorphisme précédente permet donc de résoudre ce problème.

Références

1. A. Delignat-Lavaud, Algèbres tropicales et plus court chemin, Quadrature n◦ 72 (2009) 22–28, EDP Sciences, 2009.

2. M. Gondran et M. Minoux, Graphes, dioïdes et semi-anneaux, Nouveaux modèles et algorithmes, Ed. Tec & Doc, Paris, 2001.

3. M. Gondran and M. Minoux. Graphes et algorithmes. Eyrolles, Paris, 3e édition, 1995.

4. Fallou Gueye, Christian Artigues et Marie José Huguet, Planification d'itinéraires en transport multimodal, edsys2009, Toulouse, mai 2009. `http://www.laas.fr/EDSYS/contents/congres/2009/contents/documents/edsys2009_submission_23.pdf`

5. Projet MaxPlus, INRIA, `http://www-rocq.inria.fr/MaxplusOrg`.

6. M. Trehel, Algorithmes de cheminement, polycopié de DEA, Université de Besançon, 1976.

Annexe

Exemples d'algèbre de chemins

Types de problèmes résolus	Problèmes résolus	E	$\oplus$	$\otimes$	ε	e
Existence	Les problèmes de connexité	$\{0,1\}$	max	min	0	1
Énumération	Énumération des chemins élémentaires	$\mathcal{P}(X^*)$	Union	Multiplication latine	ø	X
	Problèmes multicritères	$\mathcal{P}(\overline{\mathbb{R}}^p)$	Ensemble des chemins efficaces de l'union	Ensemble des chemins efficaces de la somme	$(+\infty)^p$	$(0)^p$
	Génération des langages réguliers (Kleene)	Ensemble des mots	Union	Concaténation	ø	Le mot vide
Optimisation	Chemin de capacité maximum	$\mathbb{R}^+ \cup \{+\infty\}$	max	min	0	$+\infty$
	Abre minimum Classification hiérarchique	$\overline{\mathbb{R}}$ $\overline{\mathbb{R}}^+$	min	max	$+\infty$	$-\infty$ 0
	Chemin de nombre d'arcs minimum	$\mathbb{N} \cup \{+\infty\}$	min	+	$+\infty$	0
	Plus court chemin	$\mathbb{R} \cup \{+\infty\}$	min	+	$+\infty$	0
	Plus long chemin	$\mathbb{R} \cup (+\infty\}$	max	+	$-\infty$	0
	Chemin de fiabilité maximum	$\{a \mid 0 \leq a \leq 1\}$	max	×	0	1
	Fiabilité d'un réseau	Polynômes idempotents	Différence symétrique	×	0	1
Dénombrement	Dénombrement de chemins	$\mathbb{N}$	+	×	0	1
	Chaînes de Markov	$\{a \mid 0 \leq a \leq 1\}$	+	×	0	1
Optimisation post-optimisation	Problème du $k^{ième}$ chemin	Cône de $\mathbb{R}^{-k}$	k plus petits termes des 2 vecteurs	k plus petits termes des sommes de couples	$(+\infty)^k$	$(0, +\infty, \ldots, +\infty)$
	Chemins η-optimaux	Suite ordonnée de termes de $\mathbb{R}$ d'amplitude η	Suite des plus petits termes à η près des 2 suites	Suite des plus petits termes à η près des sommes de couples	$(+\infty)$	(0)

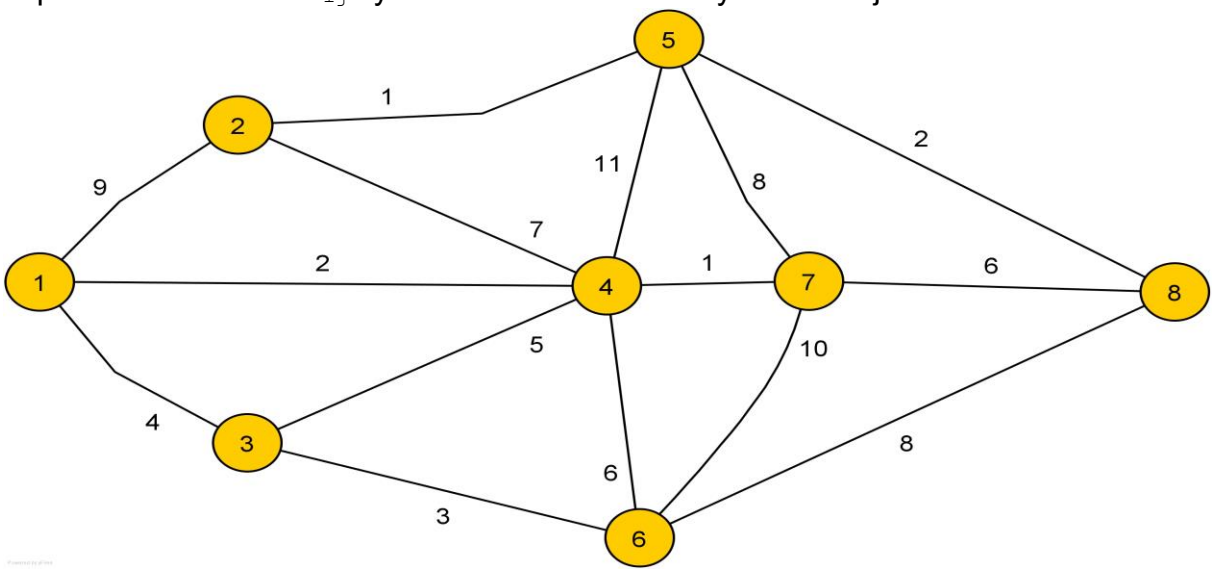
Master 2 Matis Année 2010-2011

MIS 7 - Sujet d'étude n°1

Ce travail est à rendre pour lundi 3 janvier 2011 par mail à : nakech@free.fr

Il fera l'objet d'un examen individuel oral aux horaires indiqués plus tard.

L'objectif de cet exercice est de trouver les chemins de capacité maximum.
Le graphe ci-dessous est un graphe non orienté ($a_{ij} = a_{ji}$) dont les arcs sont libellés par des capacités. L'élément a_{ij} symbolise le débit d'un tuyau reliant j à i.



On définie la capacité d'un chemin comme la plus petite capacité de tous ces arcs, car le débit d'une canalisation étant limité par le minimum des débits des tuyaux qui la composent, par exemple la capacité du chemin (1,4,5) est égale à 2, et la capacité maximum entre le sommet 1 et le sommet 5 est égale à 6 correspondant à la capacité du chemin (1,2,4,5).
L'algèbre max-min, avec le maximum comme addition, et le minimum comme multiplication, $(R^+, \max, \min, \infty, 0)$ permet résoudre ce problème de cheminement (Calculer A^n dans l'algèbre max-min revient à trouver les canalisations de débit maximal parmi celles de longueur n).

1. Démontrer que, $(R^+, \max, \min, \infty, 0)$ est un Semi-anneau idempotent. Donner la matrice d'adjacence A (Vérifier que ce matrice est symétrique).

2. Réaliser (ou récupérer ou adapter) un petit programme permettant d'appliquer l'algorithme de **Warshall** pour calculer A^* correspondant à la capacité maximal entre toute paire de sommets.

3. Dessiner le graphe de A^*

4. Appliquer l'algorithme de **Dijkstra** pour trouver les chemins de capacité maximale entre le sommet 1 et tous les autres sommets.

5. Appliquer l'algorithme de **Jacobi** pour calculer la capacité maximal entre le sommet 1 et tous les autres sommets (la première ligne de A^*).

6. Trouver une nouvelle version de l'algorithme de **Jacobi** permettant de trouver les chemins de capacité maximal entre le sommet 1 et tous les autres sommets.
   Indication : Dans le paragraphe 3.1.3 voir l'amélioration dans la nouvelle version l'algorithme de **Dijkstra**.

Université Du Havre, Master : Mathématiques-Informatique (Matis)

M2-S1-CPX "Complexité spatiale et dynamique"
2012/2013

Nakechbandi M.

moustafa.nakechbandi@univ-lehavre.fr

Chapitre 3 : Algorithmes itératifs Asynchrones

3.1/ Méthodes itératives/directes pour le calcul numérique

3.1.1/ Méthodes directes :

Calculent la solution exacte (en négligeant les erreurs d'arrondis voir annexe A)

Gourmandes en mémoire (dimension n => stockage n^3)

En général, pour les problèmes non-linaire les méthodes directes n'existe pas.

3.1.2/ Méthodes itératives :

Approximation de la solution par les itérations : $y^{k+1} = f(y^k)$

Plus facilement parallélisable

Moins gourmandes en mémoire

Sont indispensable pour résoudre problèmes non-linaire

3.1.3/ Exemple : ***La résolution d'un système linaire par une méthode itérative***

Suppose that it is desired to find the solution to the equation system

$$\begin{bmatrix} 2 & 1 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 3.1 \\ 5.1 \\ 5.1 \\ 5.1 \\ 5.1 \\ 2.9 \end{bmatrix}$$

Since the system is sparse, an iterative method will be used to solve the system. From this equation system it can be seen that the Gauss–Seidel iteration steps will be

$$x_1 = \frac{3.1 - x_2}{2}$$

$$x_i = \frac{5.1 - x_{i-1} - x_{i+1}}{2}, \qquad \text{for} \quad i = 2, 3, \ldots, 5$$

$$x_6 = \frac{2.9 - x_5}{2}$$

These relationships are included in the following BASIC program. This program uses an initial guess of $x_i = 1$ for $i = 1, 2, \ldots, 6$ and is written to accommodate different choices for w in the overrelaxation relationship. From this program it is obvious that the amount of working storage required to accommodate the iteration process is roughly two times the dimensional size of the coefficient matrix. To do this same problem with the Gauss–Jordan elimination method would require working storage on the order of twice the square of the dimensional size of the coefficient matrix. Thus the space saving becomes obvious for large systems that are sparse.

```
1000  REM **********************
1010  REM *THIS PROGRAM FINDS  *
1020  REM *THE SOLUTION TO A   *
1030  REM *SET OF SIMULTANEOUS *
1040  REM *LINEAR ALGEBRAIC    *
1050  REM *EQUATIONS BY        *
1060  REM *ITERATION USING THE *
1070  REM *METHOD OF SUCCESSIVE*
1080  REM *OVER RELAXATION.    *
1090  REM **********************
1100 :
1110 :
1120  REM **SET INITIAL VALUES**
1130 :
1140  DIM X(7),Y(7)
1150  FOR I = 1 TO 6
1160 X(I) = 1
1170  NEXT I
1180 :
1190 :
1200  REM *BEGIN ITERATIVE**
1210  REM *PROCESS         **
1220 :
1230 NCT = 0
1240 W = 1.
1250  PRINT "STARTING VALUES"
1260  GOSUB 1690
1270  FOR I = 1 TO 6
1280 Y(I) = X(I)
1290  NEXT I
1300 NCT = NCT + 1
1310  IF NCT > 50 THEN  GOTO 1600
1320 XBAR = (3.1 - X(2)) / 2.
1330 X(1) = X(1) + W * (XBAR - X(
     1))
1340  FOR I = 2 TO 5
1350 XBAR = (5.1 - X(I - 1) - X(I
      + 1)) / 2.
1360 X(I) = X(I) + W * (XBAR - X(
     I))
1370  NEXT I
1380 XBAR = (2.9 - X(5)) / 2.
1390 X(6) = X(6) + W * (XBAR - X(
     6))
1400  FOR I = 1 TO 6
1410  IF  ABS (Y(I) - X(I)) > 0.0
     001 THEN  GOTO 1270
1420  NEXT I
1430 :
1440 :
1450  REM **CONVERGENCE  **
1460  REM **PRINT SUMMARY**
1470 :
1480  PRINT "W=";W
1490  PRINT "CONVERGENCE REACHED"
1500  PRINT NCT;" ITERATIONS REQU
     IRED"
1510  PRINT "FINAL VALUES ARE"
1520  GOSUB 1690
1530  END
1540 :
1550 :
1560  REM **NONCONVERGENCE **
1570  REM **PRINT SUMMARY  **
1580 :
1590  PRINT "W=";W
1600  PRINT "NO CONVERGENCE "
1610  PRINT "AFTER 50 ITERATIONS"
1620  PRINT "LAST VALUES USED "
1630  GOSUB 1690
1640  END
1650 :
1660 :
1670  REM **PRINT SUBROUTINE**
1680 :
1690  PRINT "--------------------
     "
1700  FOR I = 1 TO 6
1710  PRINT "X(";I;")=";X(I)
1720  NEXT I
1730  PRINT "--------------------
     "
1740  PRINT : PRINT : PRINT
1750  RETURN
```

```
STARTING VALUES
--------------------
X(1)=1
X(2)=1
X(3)=1
X(4)=1
X(5)=1
X(6)=1
--------------------

W=1
CONVERGENCE REACHED
33 ITERATIONS REQUIRED
FINAL VALUES ARE
--------------------
X(1)=.785496963
X(2)=1.52892425
X(3)=1.25674647
X(4)=1.05749999
X(5)=1.72831339
X(6)=.585843305
--------------------
```

This program required less than 10 seconds to complete the 33 iterations on an Apple II computer. If the relaxation factor w is changed to 1.4, the program uses only 14 iterations and the run time is reduced to 5 seconds. For this case, the output is

```
STARTING VALUES
--------------------
X(1)=1
X(2)=1
X(3)=1
X(4)=1
X(5)=1
X(6)=1
--------------------

W=1.4
CONVERGENCE REACHED
14 ITERATIONS REQUIRED
FINAL VALUES ARE
--------------------
X(1)=.785686222
X(2)=1.52859928
X(3)=1.25711714
X(4)=1.05715737
X(5)=1.7285651
X(6)=.585716582
--------------------
```

3.1.4/ **Les schémas d'exécution d'un algorithme itératif :**

Schéma algorithmique Jacobi

pour $t=0,1,2\ldots$
  pour $i \in \{1,\ldots,n\}$
    $x_i^{t+1} = f_i(x^t)$
  fpour
fpour

Schéma algorithmique Gauss-Seidel

pour t = 0,1,2 …
  pour i ϵ {1,2, …, n}
    $x_i^{t+1} = f_i(x_1^{t+1}, x_2^{t+1}, \ldots x_i^{t+1}, x_{i+1}^{t} \ldots, x_n^{t+1})$
  fpour
fpour

Schéma algorithmique chaotique

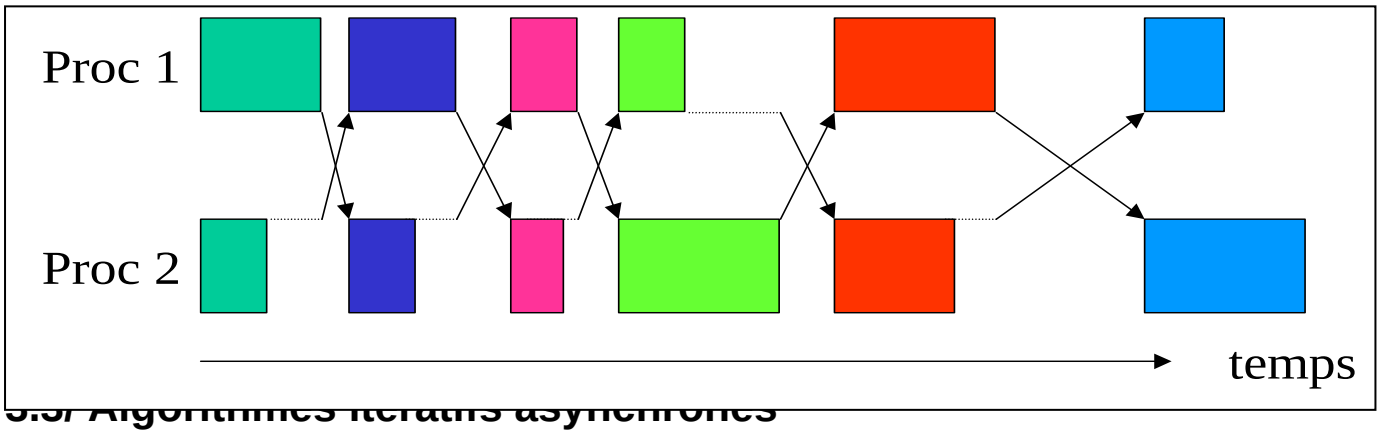
pour $t=0,1,2\ldots$
  pour $i \in \{1,\ldots,n\}$
    si $i \in J(t)$ *alors*
      $x_i^{t+1} = f_i(x^t)$
    sinon
      $x_i^{t+1} = x_i^t$
    fsi
  fpour
fpour

Schéma algorithmique chaotique à retard

pour $t=0,1,2\ldots$
  pour $i \in \{1,\ldots,n\}$
    si $i \in J(t)$ *alors*
      $x_i^{t+1} = f_i(x_1^{t-r_1^i(t)}, x_2^{t-r_2^i(t)}, \ldots, x_i^t, \ldots, x_n^{t-r_n^i(t)})$
    sinon
      $x_i^{t+1} = x_i^t$
    fsi
  fpour
fpour

3.2/ Algorithmes itératifs synchrones

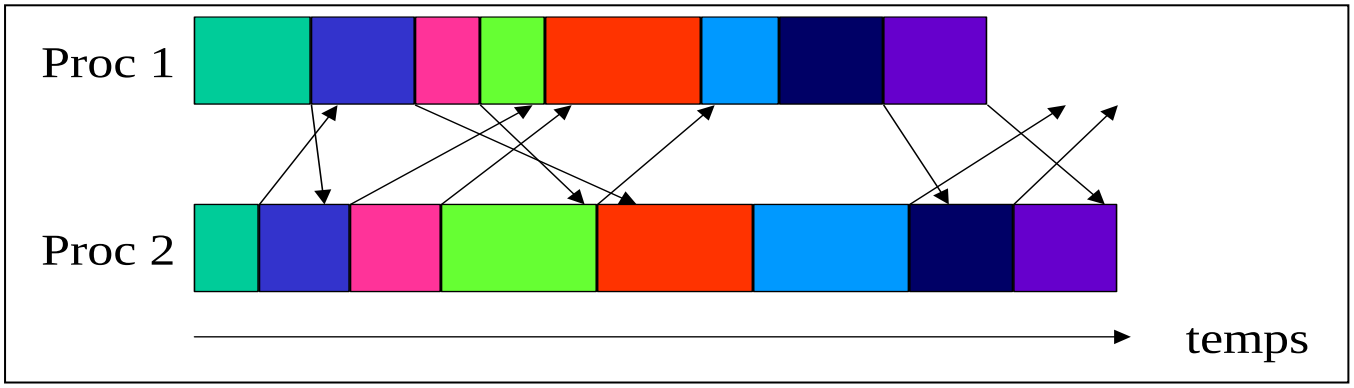
Les processeurs commencent la même itération au même moment



3.3/ Algorithmes itératifs asynchrones

3.3.1/ Principe

Les processeurs ne calculent pas forcément la même itération à un instant t ⇒ plus de période d'inactivité



3.3.2/ Schéma algorithmique général

```
AnciennesValeurs = Données calculées localement à l'itération précédente
NouvellesValeurs = Données calculées localement à l'itération courante

Initialisation du système
Initialisation des données dans AnciennesValeurs
répéter
    Calcul de NouvellesValeurs en utilisant AnciennesValeurs
    Envois asynchrones des nouvelles données
    "Copie" de NouvellesValeurs dans AnciennesValeurs
    Détection de la convergence globale
jusqu'à Convergence globale détectée
Affichage ou sauvegarde des données locales
Arrêt du système
```

3.3.3/ Exemples

Exemple 1 : Méthode de newton [7]

Considérons la **méthode de Newton** pour résoudre `F(x)=0` où `F` est une fonction non-linéaire, `F,x ∈ R`^{n}.

Algorithme

La recherche d'un zéro pour la fonction `F(x)` peut se faire en utilisant l'itération de Newton : `Xn+1 = Xn - F(Xn)/F'(Xn)..... (1)`

Cette méthode produit une séquence d'approximations, en commençant d'une valeur initiale `X0`. On s'arrête quand `|Xn+1 - Xn| < epsilon.`

Supposons :

- Que pour un x donné, le calcul de `F(x)` nécessite `t1` unités du temps ;
- Que pour un y donné, le calcul de `F'(y)` nécessite t2 unités du temps ;
- Et que pour `a,b,c, le` calcul de `d=a-b/c` nécessite t3 unités du temps; si `|d-a| > epsilon` alors `d` est envoyé à P1 et P2 sinon on a la convergence et `d` correspond au résultat (le point fixe de l'équation `(1).`

Pour réaliser le calcul précédent on peut envisager 2 modes d'exécution :

Mode Séquentiel

Le meme processeur calcule Successivement : `F(Xn), F'(Xn),` puis `Xn+1` en utilisant `(1).` D'où le temps exécution d'une itération est = `t1+t2+t3.` Si k

itérations sont nécessaires pour la convergence alors le temps total sera = `k(t1+t2+t3)`.

Supposons maintenant que nous avons 3 processeurs

Mode parallèle synchrone.

P1 et P2 calculent `F(Xn)` et `F'(Xn)` simultanément et quand les deux terminent le calcul des valeurs `F(Xn)` et `F'(Xn)` P3 les utilise pour calculer `Xn+1`
D'où le temps d'exécution d'une itération est = `max( t1,t2)+t3` ⇒ le temps total dans ce cas sera `k[max(t1,t2)+t3]` ou `k` est le nombre d'itérations .

Mode parallèle asynchrone.

P1 et P2 commencent à calculer dès qu'une nouvelle valeur d'entrée est rendue disponible par P3 et ils sont prêts à le recevoir,
P3 calcule une nouvelle valeur en utilisant `(1)` dès que P1 ou P2 fournissent une nouvelle entrée.

Le temps par itération est au plus `min(t1, t2) t3` (mais peut être aussi bas que t3) par contre nous ne pouvons pas prédire combien d'itérations seront nécessaires. Supposons `t1=2, t2=3` et `t3=1`.

Le tableau suivant illustre ce mode d'exécution :

	2 unités	3 unités	1 unités
temps	**P1**	**P2**	**P3**
2	F(X0)	C0	-
3	-	F'(X0)	-
4	-	-	X1 = X0 - F(X0)/F'(X0)
5	C1	C1	-
6	F(X1)	C1	-
7	-	F'(X1)	X2 = X1 - F(X1)/F'(X0)
8	C2	C2	X3 = X2 - F(X1)/F'(X1)
9	F(X2)	C2	-
10	C3	F'(X2)	X4 = X3 - F(X2)/F'(X1)
11	F(X3)	C3 or C4	X5 = X4 - F(X2)/F'(X2)
12	C4 or C5		X6 = X5 - F(X3)/F'(X2)
13	.	.	.
etc.			

– indique le processeur est au repos
`Ci` indique que le processeur utilise `Xi` dans son calcul

NOTE : Au temps `11` P2 a le choix d'utiliser `X3` ou `X4` pour calculer `F'(X)`.

Exemple 2 : Le problème :

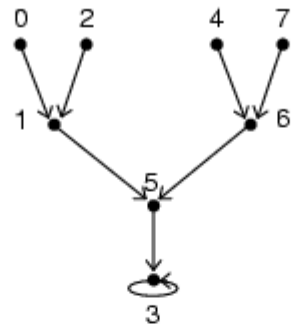
Parallel synchronous evolution of 3 booleans processors (automatas):

$x_i \in \{0,1\}$, $(x_1, x_2, x_3) \xrightarrow{F} (y_1, y_2, y_3)$

State	X	$F(X)$
0	000	001
1	001	101
2	010	001
3	011	011
4	100	110
5	101	011
6	110	101
7	111	110

Parallel synchronous iterations

Algorithme itératif asynchrone :

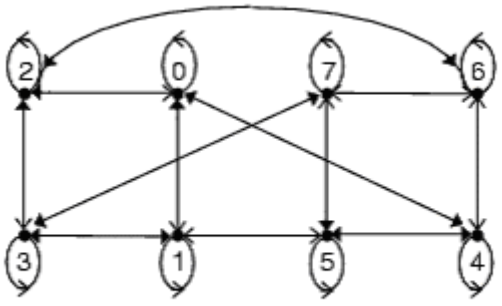


Componentwise iterations chaotic iterations

(at each iteration, only one component is updated).

State	$F_1(X),X_2,X_3$	$X_1,F_2(X),X_3$	$X_1,X_2,F_3(X)$
0	000	000	001
1	101	001	001
2	010	000	011
3	011	011	011
4	100	110	100
5	001	111	101
6	110	100	111
7	111	111	110

Chaotic iterations



Some asynchronous iterations

3.3.3/ Avantages

- *Itérations totalement désynchronisées*

  ⇒ *pas de synchronisations pénalisantes*

  ⇒ *pas d'ordonnancement*

- *Convergence sous conditions, satisfaites dans les applications scientifiques.*

- *Tolérance de pertes de messages.*

- *Tolérance aux pannes ou délais des communications*
  ⇨ Adapté au global computing (Metacomputing ou encore grid computing)

  *Rappelons que le **Metacomputing** (calcul sur le grid) est un calcul faisant usage de collections distribuées de plates-formes hétérogènes (la hétérogénité est aussi bien au niveau des processeurs que au niveau des réseaux de connection).*

3.3.4/ Restes à résoudre

- *La convergence*

- *Équilibrage de charge*

3.4/ Model mathématique

B_i, $i \in \{1, ..., \alpha\}$ Banach spaces $\Rightarrow$ $|| \cdot ||_i$
The fixed point problem

$$u = Gu,\ u \in B = \prod_{i=1}^{\alpha} B_i$$

The asynchronous algorithm (G, u^0, J, S) is defined by

$$\begin{cases} Given\ u^0 = (u_1^0, ..., u_\alpha^0) \\ \quad for\ p = 1, 2... \\ \quad\quad for\ l = 1, ..., \alpha \\ u_l^{p+1} = \begin{cases} G_l(u_1^{s_1^l(p)}, ..., u_\alpha^{s_\alpha^l(p)})\ if\ l \in J(p) \\ u_l^p\ if\ l \notin J(p) \end{cases} \end{cases}$$

- $J(p)$ is the set of components to be updated at step p
- $p - s_j^l(p)$ is the delay of the l^{th} proc. w.r.t. the l proc. computing the j^{th} block at the p^{th} iteration
  - If $s_i^l(p) = p$ then: synchronous algorithms.
  - $$\begin{cases} s_i^l(p) = p\ for\ p = 1, 2, ...\ and\ i = 1, 2, ... \\ J(p) = \{1, 2..., \alpha\}\ \ for\ p = 1, 2, ... \end{cases}$$
    $\Rightarrow$ the block Jacobi algorithm.
  - $$\begin{cases} s_i^l(p) = p\ for\ p = 1, 2, ...\ and\ i = 1, 2, ... \\ J(p) = 1 + p(mod\ \alpha)\ for\ p = 1, 2, ... \end{cases}$$
    $\Rightarrow$ the block Gauss Seidel algorithm.

Hypotheses

- $\forall i,\ \{p \in \mathbb{N}, i \in J(p)\}$ is infinite
- $\forall i, l \in \{1, ..., \alpha\}, \forall p \in \mathbb{N}, s_i^l(p) \leq p$
- $\forall i, l \in \{1, ..., \alpha\}, \lim_{p \to \infty} s_i^l(p) = +\infty$

3.5/ La convergence

General convergence theorems

1) Miellou - El Tarazi's Theorem (contraction)

If
(a) $D(G) = \prod_{i=1}^{\alpha} D_i(G)$
(b) $G(D(G)) \subset D(G)$
(c) $\exists u^* \in D(G)$, such that $u^* = G(u^*)$
(d) $||G(u) - u^*||_\gamma \leq \beta ||u - u^*||_\gamma, (0 < \beta < 1)$

$$|| \cdot ||_\gamma = \max_{1 \leq l \leq \alpha} \frac{|| \cdot ||_l}{\gamma_l}, (\gamma \gg 0)$$

then any asynchronous algorithm (G, u^0, J, S) converges to the fixed point u^* of G.

Applications du théorème de Miellou (1975, 1981) **:** Problèmes numérique de point fixe.

2) Bertsekas Theorem

B_i given sets and B their Cartesian product

$$B = \prod_{i=1}^{\alpha} B_i$$

If there exit a sequence of nonempty sets $\{B(k)\}$ with

$$... \subset B(k+1) \subset B(k) \subset ... \subset B$$

(a) Synchronous Convergence Condition: $f(x) \in B(k+1), \forall k$ and $x \in B(k)$
(b) Box Condition: $\forall k$, there exist sets $B_i(k) \subset B_i$ such that

$$B(k) = B_1(k) \times B_2(k) \times ... \times B_\alpha(k)$$

then any asynchronous algorithm (G, u^0, J, S) where $u^0 \in B(0)$ converges to the fixed point u^* of G

3.6/ Applications du théorème de Bertsekas (1989) [2] **:**

Problèmes non numérique de point fixe.

Tous les problèmes qui se formulent par une équation de point fixe sur un espace algébrique fini.

- **Algèbre de chemin**

  ⇨ Routage dynamique et routage résistant aux pannes dans les réseaux [3]

- **D'autres problèmes combinatoires**

  ⇨ Programmation dynamique sur un espace discret [2]

  ⇨ Problème d'Allocation de fréquence dans les réseaux sans fil [5]

  ⇨ Problème de routage lié au écoulement de paquet dans le réseau [6]

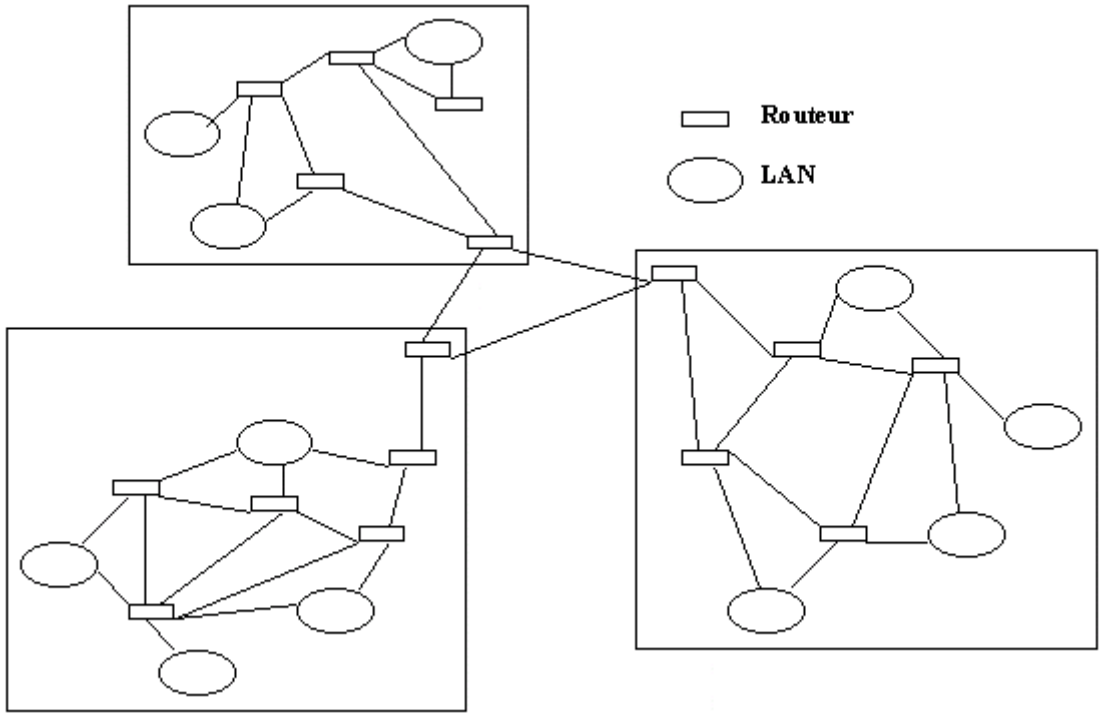
3.7/ Application au routage dans les réseaux

Dans cette partie, nous montrons comment certains algorithmes répartis auto-stabilisants comme le protocole RIP (Routing Information Protocol) peuvent être étudiés via itération asynchrone basée sur une équation de point fixe matricielle.

3.7.1/ Rappel : Le problème de routage dans le réseau d'internet

En général, le routage est le mécanisme par lequel des chemins sont sélectionnés dans un réseau pour acheminer les données d'un expéditeur jusqu'à un ou plusieurs destinataires.

Un routeur est un élément intermédiaire dans un réseau informatique assurant le routage des paquets. Son rôle est de faire transiter des paquets d'une interface réseau vers une autre. Un routeur doit être connecté à au moins deux réseaux informatiques pour être utile, sinon il n'aura rien à router.
L'appareil crée et/ou maintient une table, appelée table de routage, laquelle mémorise les meilleures routes vers les autres réseaux, via les métriques associées à ces routes.



Protocole de routage : **C'est le mécanisme par lequel les paquets d'un routeur sont acheminées jusqu'à leur destinataire**

Le protocole RIP **(Routing Information Protocol).**

Ce protocole est Issu des travaux de Bellman-Ford, il a été développé par l'université de Californie, simple, solide et a été l'un des premiers protocoles de routage à être développé.

Principe et Routage vecteur-distance (RIP**)**

RIP utilise une métrique à nombre de sauts, aussi qualifiée de métrique de distance. Le nombre de sauts indique le nombre de routeurs traversés pour atteindre la destination.

Si le chemin que doit prendre un paquet possède un nombre de saut égal à 5 dans la table de routage, cela signifie que le paquet empruntant ce chemin passera par 5 routeurs avant d'atteindre sa destination finale. Si il existe plusieurs chemins, il sera préféré celui avec le moins de sauts.

Le nombre de saut est la seule métrique utilisée par RIP, et l'inconvénient est que le nombre de saut est limité à 15. Un réseau situé à une distance de plus de 15 sauts ne pourra être atteignable.

Le routeur va utiliser la diffusion sur le réseau pour propager ses tables de routage (en broadcast pour RIP version 1). Cette mise à jour se fait toutes les 30 sec et envoie ces tables à tous les routeurs voisins (directement connecté), ce qui augmente le trafic sur le réseau.

Il n'y a pas d'accusé de réception des messages, si on ne reçoit aucun message durant 180 sec, la route silencieuse est marqué comme inaccessible. S'il n'y a toujours aucune mise à jour après 240 secondes, le protocole RIP enlève toutes les entrées dans sa table de routage correspondant au routeur qui ne répond pas.

Lorsqu'un routeur est arrêté par la procédure normale d'extinction, il envoie, à ses voisins, sa table avec tous les liens à 16, cette procédure permet une convergence **(**stabilisation des tables**) plus rapide vers la nouvelle situation.**

Table de routage

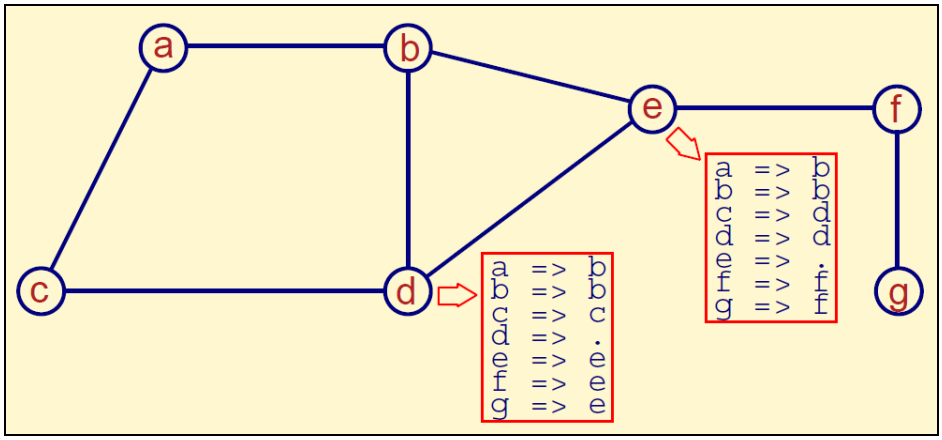
Chaque entrée de la table de routage fournit l'adresse IP d'un réseau, le prochain routeur sur le chemin, le nombre de sauts pour l'atteindre. La valeur 16 indique que la destination est inaccessible, la valeur 0 indique que le réseau est directement connecté au routeur.

RIP ne conserve que l'entrée correspondant à la meilleure route. Lorsqu'une information fournit une route plus intéressante, elle écrase l'ancienne. Destination Prochain routeur Nombre de sauts :

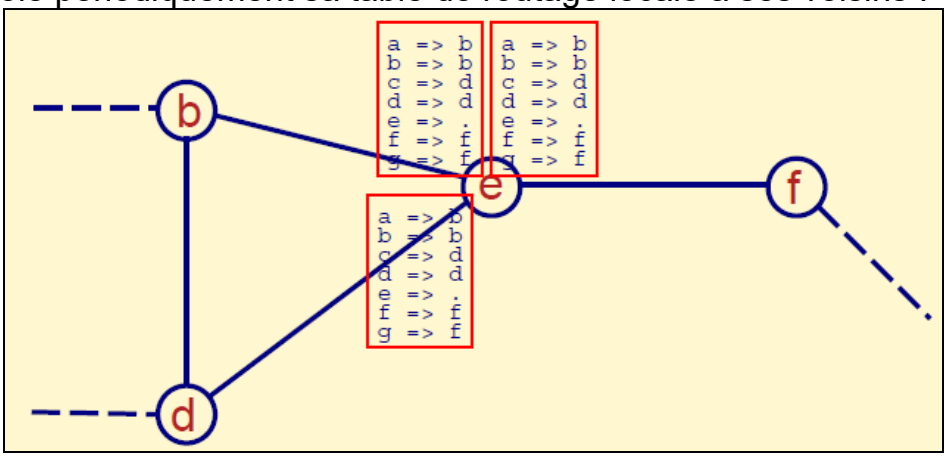
Destination	**Prochain routeur**	Nombre de sauts
IP Réseau 1	IP Routeur A	2
IP Réseau 2	IP Routeur B	4
IP Réseau 3	IP Routeur B	3
…		

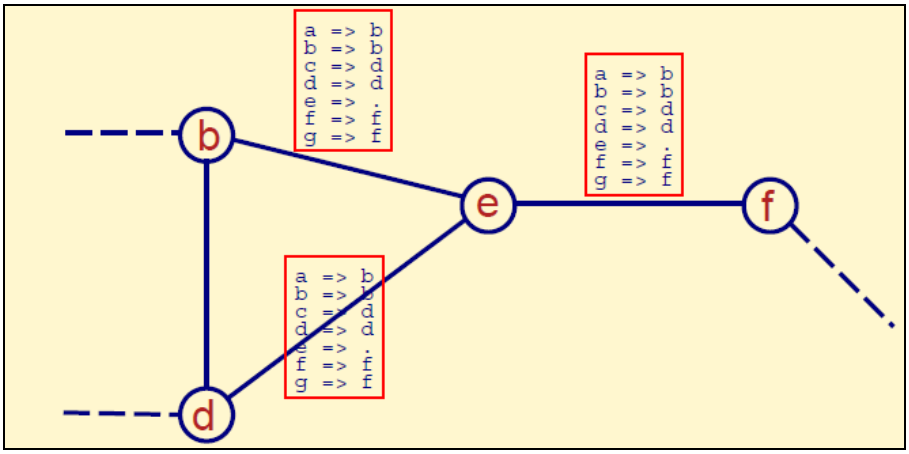
Illustration du protocole RIP :

Exemple d'une table de routage locale (table de d et de e)

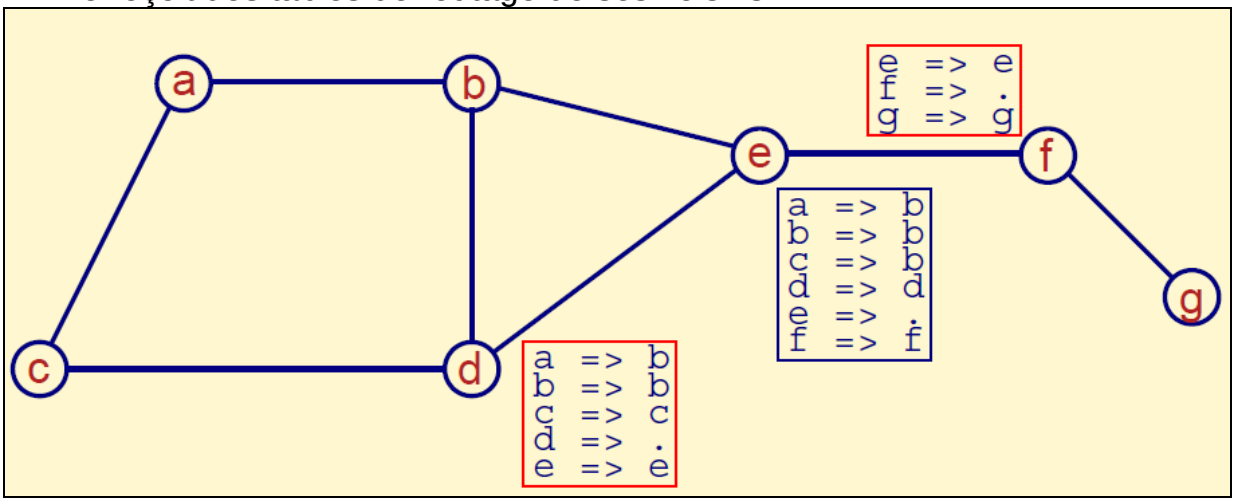


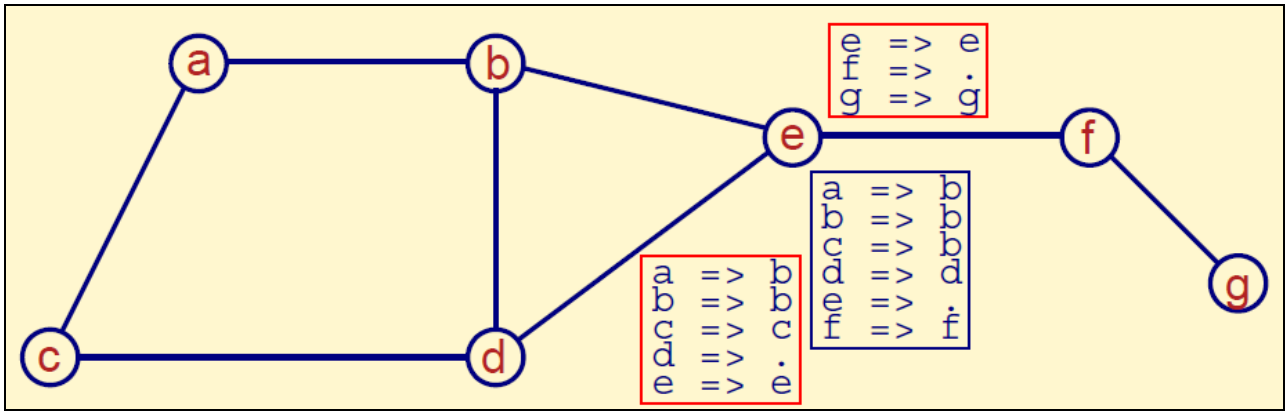
Un nœud envoie périodiquement sa table de routage locale à ses voisins :





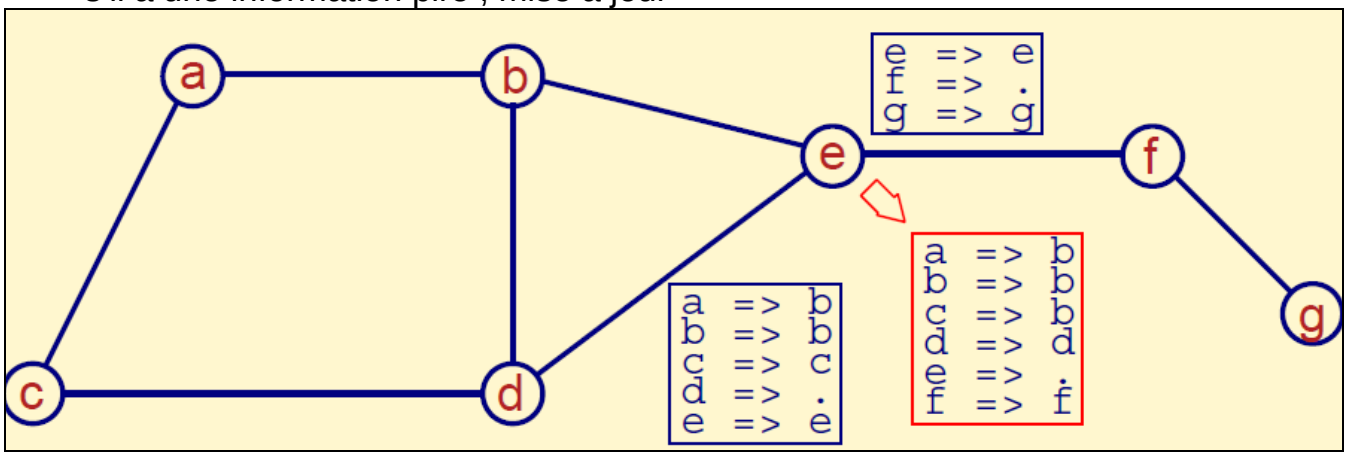
Le nœud e reçoit des tables de routage de ses voisins :



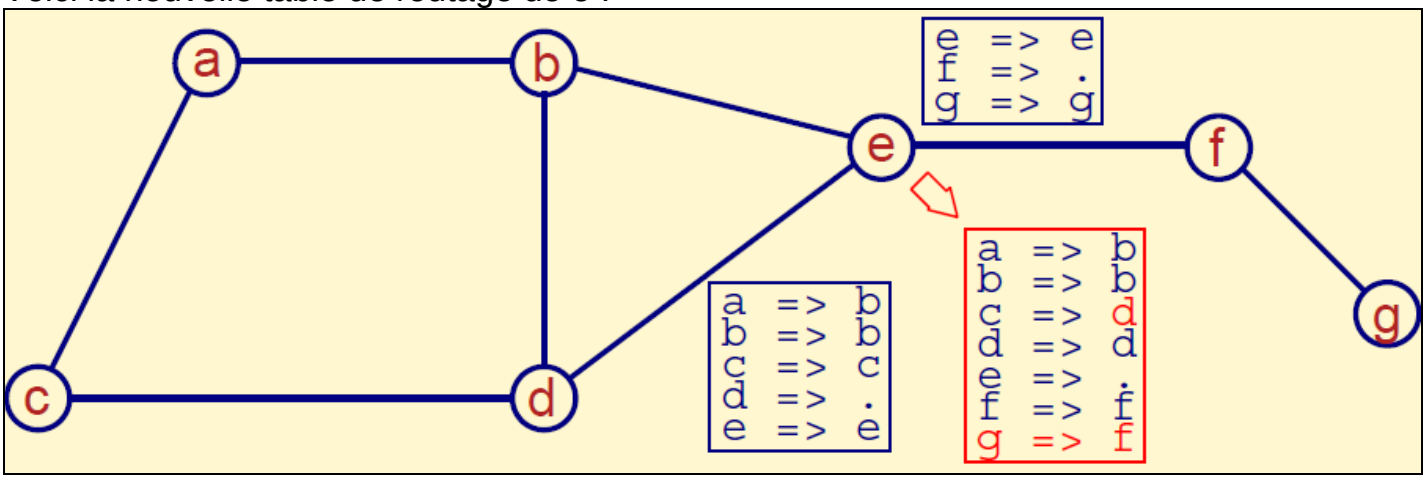


Quand le nœud e reçoit une information concernant un nœud de la destination, il vérifie :

- s'il n'a pas d'information pour ce nœud ; ajoutez
- s'il a une information pire ; mise à jour



Voici la nouvelle table de routage de e :



3.7.2/ Calcul de chemin :

Soient $G = (X,U)$ un graphe dont les arcs sont libellés par des éléments dans semi anneau idempotent K, et A la matrice d'incidence associée à G.

On a vue dans le chapitre 1 que $Z = A^k$ contient le poids des arcs de longueur k. C'est-à-dire $z_{i,j}$ correspond au poids d'un chemin de longueur k reliant i à j.

De même la somme $A^{(k)} = E \oplus A^1 \oplus A^2 \ \ldots \ \oplus A^k$ contient la somme des les poids de tous les chemins dont la longueur en nombre d'arcs est $\leq k$. (par exemple dans le cas de problème de plus court chemin somme = min donc

$A^{(k)}$ contient les valeurs des plus court chemin de k arcs au maximum).

Soit $\mathbf{A^* = A^{(k)} \mid k \rightarrow \infty}$, on a vue que dans l'absence d'un circuit absorbant (ou négatif) on peut démontrer que $A^* = A^{(n)}$ où n est le nombre de sommets.

Un grand nombre de problème de cheminement se ramène donc au calcul de A*

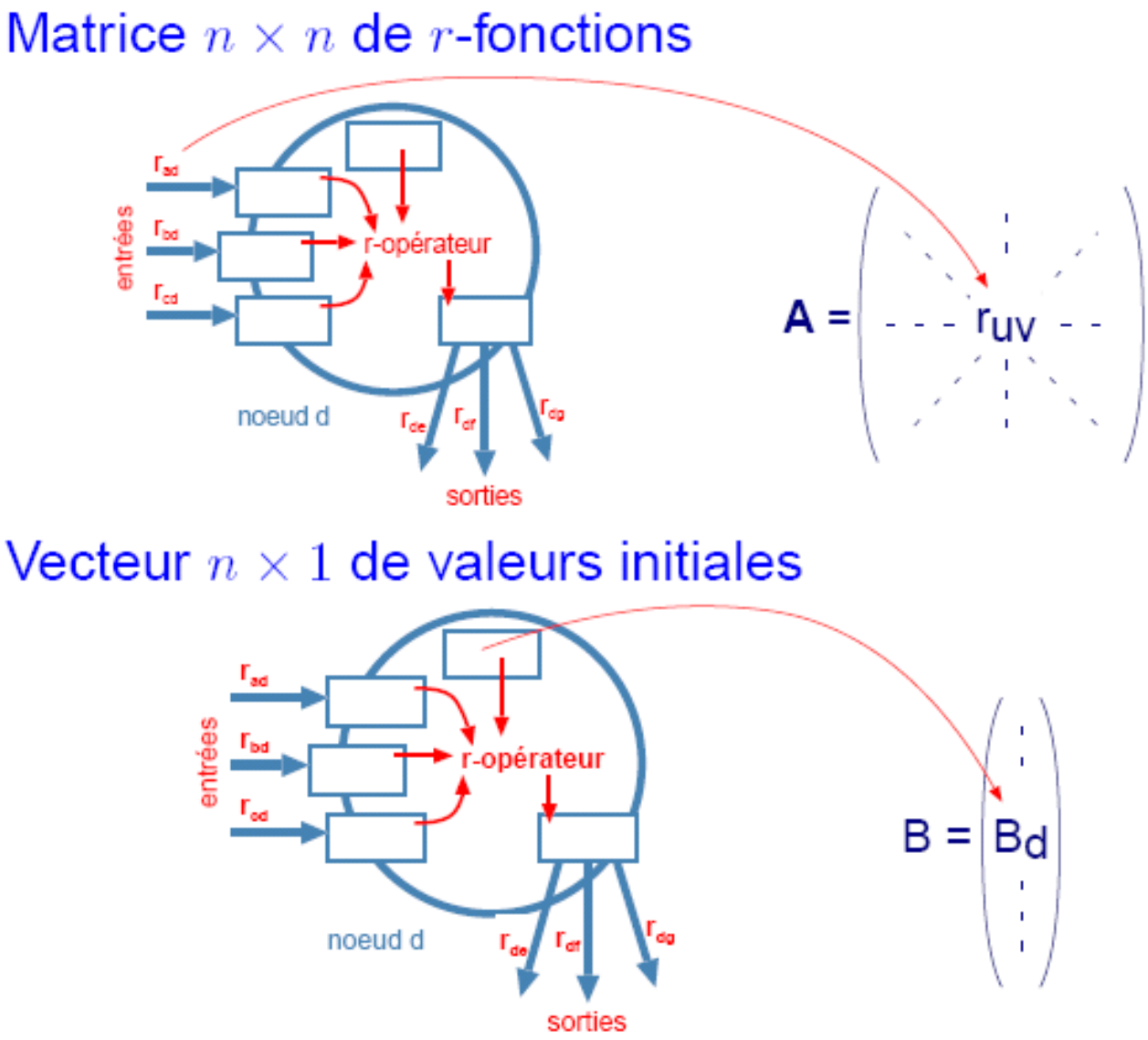
On a démontrer aussi que $\mathbf{A^* = A * A^* + E}$

La recherche de A^* revient donc à la résolution l'équation :

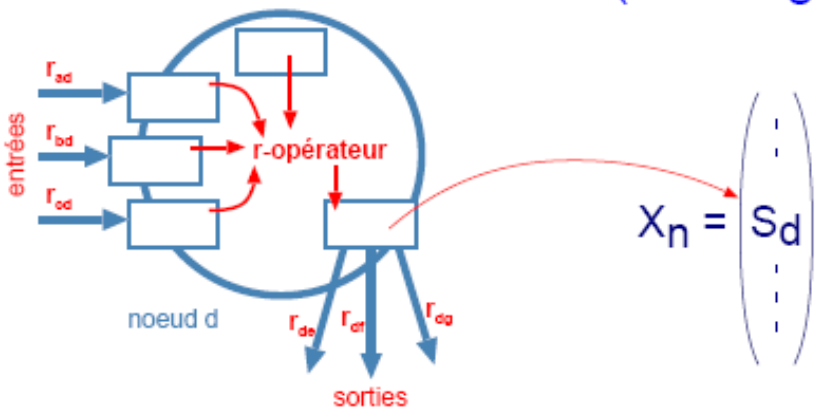
$\mathbf{X = A * X + E}$ (ou de $Y = Y \otimes A \oplus E^T$)

Ce qui nous permet de voir ça comme la recherche d'un point fixe et permet d'appliquer des algorithmes **itératifs** de type Jacobi ou Gauss-Seidel, ou encore d'appliquer des algorithmes plus originaux de type **itération asynchrone**.

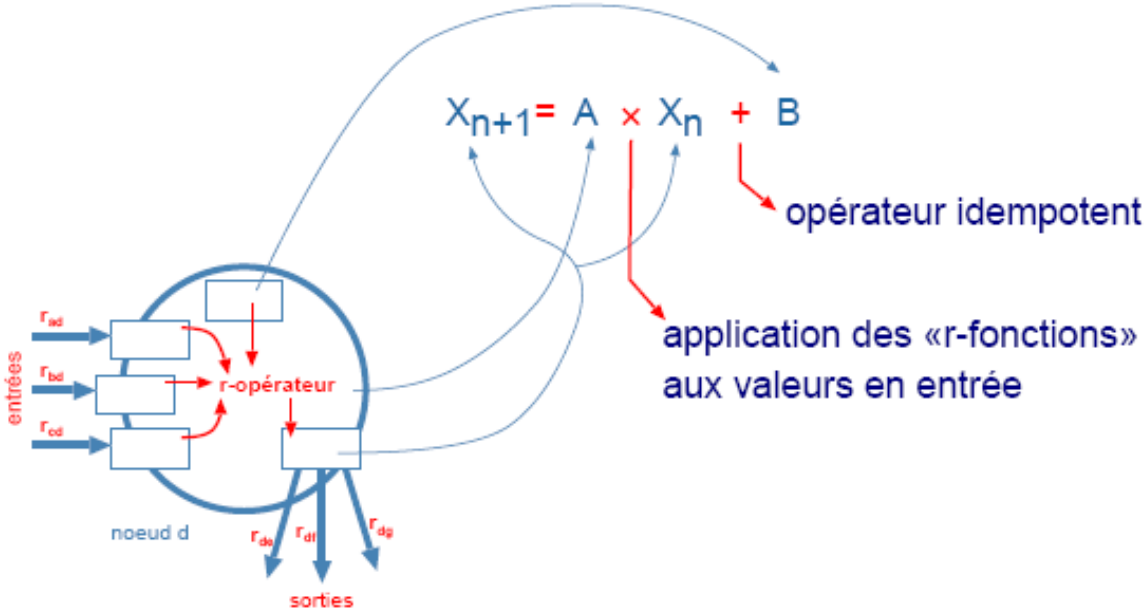
3.7.3/ Modélisation matricielle du problème de routage [3].



- Vecteur $n \times 1$ résultat ($\approx$ configuration)



- Algorithme réparti $\equiv$ multiplication matricielle



certains processeurs $P_i, i \in \mathcal{J}$ sont activés et
- lisent les valeurs de tous leurs antécédants
- calculent un nouveau résultat
- écrivent le résultat dans le registre de sortie

■ Itérations asynchrones

$$\mathrm{X}_{n+1}[i] = \begin{cases} \mathrm{X}_n[i] & \text{si } i \notin \mathcal{J}_n \\ (\mathrm{F}(\mathrm{X}_n))\,[i] = & \\ (\mathbf{A} \otimes \mathrm{X}_n \ \oplus \ \mathrm{B})\,[i] & \text{si } i \in \mathcal{J}_n \end{cases}$$

En insérant le délai de retard *D[i]*, on obtient :

$$\mathrm{X}_{n+1}[i] =$$

$$\begin{cases} \mathrm{X}_n[i] & \text{si } i \notin \mathcal{J}_n \\ \left(\mathrm{F}(\ (\mathrm{X}_{D_n[1]}[1], \ldots, \mathrm{X}_{D_n[N]}[N])^{\mathsf{t}}\ )\right)[i] = & \\ \left(\mathbf{A} \otimes (\mathrm{X}_{D_n[1]}[1], \ldots, \mathrm{X}_{D_n[N]}[N])^{\mathsf{t}} \oplus \mathrm{B}\right)[i] & \text{si } i \in \mathcal{J}_n \end{cases}$$

Références :

[1] "Asynchronous algorithmes Standard and non standard cas", J Bahi, LIFC(Besançon), 2001.
[2] "D.P. Bertsekas and J.N. Tsitsiklis, Parallel and Distributed Computation" : Numerical Methods. Englewood Cliffs, N.J.: Prentice Hall, 1989.
[3] "Self-stabilization with Path Algebra, Bertrand Ducourthial, Sébastien Tixeuil, Theoretical Computer Science, Vol. 293, No. 1, pp. 219-236, 2003. voir aussi :
- Approche système en algorithmique des télécommunications, Bertrand Ducourthial http://www.hds.utc.fr/~ducourth/BIBLIO/t-20ans.pdf
- Auto-stabilisation comme un problème de point fixe matriciel, Bertrand Ducourthial, 19/06/2002 Exposé aux Journées de l'Action Spécifique Auto-Telecom, au CNAM à Paris. http://www.hds.utc.fr/~ducourth/BIBLIO/t-as-autotelecom.pdf
[4] "Distributed Routing", Abhay Parekh, EECS 228, University of California, Berkeley, 2002. http://robotics.eecs.berkeley.edu/~wlr/228a02/Lecture%20Slides/routing2.pdf.

[5] "Convergence of a Quasistatic Frequency Allocation Algorithm", Jean Walrand, B. Preneel, Journal of High-Speed Networks, 5, 3-22, 1996. http://robotics.eecs.berkeley.edu/~wlr/Papers/bart.pdf
[6] "A parallel optimal routing algorithm". C. Ribeiro, Didier El Baz : Parallel Computing 18(12): 1393-1402 (1992)
[7] "Synch. And asynch. parallel algorithms for multiprocessor", H. T. Kung , Algorithms and complexity, ed. Traub, pp 153-200, 1976.
[8] Graphes, dioïdes et semi-anneaux - Nouveaux modèles et algorithmes, M.Gondran M.Minoux Editeur(s) : Tec et Doc, 2002.

Annexe A

Arrondissement de précision

Exemple illustrant comment l'utilisation d'une méthode directe pour la résolution d'un système d'équation linaire avec l'arrondissement de précision peut générer des résultats fausses :

Considérons pour fixer les idées une machine où les réels seraient arrondis à des flottants dont la mantisse ne comporte que trois chiffres significatifs en numérotation décimale, soit $\pm(0, d_1 d_2 d_3)\ 10^{\pm e}$, et confions-lui le système

$$\begin{aligned} 10^{-4} x_1 + x_2 &= 1 \\ x_1 + x_2 &= 2 \end{aligned}$$

dont la solution exacte vaut $x_1 = 1.000100010001\cdots$, $x_2 = 0.999899989998\cdots$.

Une élimination de x_1 sans permutation des équations mène au système triangulaire suivant

$$\begin{aligned} 10^{-4} x_1 + x_2 &= 1 \\ (1 - 10^4) x_2 &= 2 - 10^4 \end{aligned}$$

c'est-à-dire au niveau de la machine

$$\begin{aligned} (0,1)\ 10^{-3} x_1 + (0,1)\ 10\, x_2 &= (0,1)\ 10 \\ -(0,1)\ 10^5 x_2 &= -(0,1)\ 10^5 \end{aligned}$$

et elle fournit la solution $x_2 = 1$, $x_1 = 0$.