Table des matières

Introduction

Les travaux présentés dans ce manuscrit portent sur des techniques d'algorithmique parallèle pour la résolution d'une classe particulière de problèmes d'optimisation : les problèmes de flot dans les réseaux. Ces problèmes interviennent dans de nombreux cas pratiques. En effet, nombreuses sont les réalisations humaines qui peuvent être modélisées par des réseaux, et ceci dans des domaines aussi variés que les transports, la distribution, les télécommunications, la finance ...

Nous nous sommes plus particulièrement intéressés à deux types de problèmes de flot :

- les problèmes non linéaires de flot à coût minimum et à critère convexe qui portent sur l'optimisation de la quantité de flux circulant dans les arcs d'un réseau pour certains débits d'entrée et de sortie donnés. Ce problème concerne la gestion du trafic ou du produit écoulé. On peut notamment citer comme exemples : le routage d'information dans les réseaux à communication de paquets et les politiques de distribution de flux.

- les problèmes linéaires de flot maximum/coupe minimale qui portent sur la détermination du débit maximum que peut écouler un réseau entre deux points particuliers. Ce dernier problème concerne plus particulièrement tous les aspects de conception de réseaux. On peut citer notamment la conception de réseaux autoroutiers, informatiques, de circuits VLSI. Notons aussi que ce problème peut intervenir comme un sous-problème particulier dans la résolution de problèmes plus complexes comme le problème de flot généralisé par exemple.

Ces deux classes de problèmes d'optimisation concernent généralement des instances de très grande taille dont la résolution nécessite de consommer beaucoup de ressources de calcul. De nouvelles techniques de résolution des problèmes de flot, faisant appel au concept de parallélisme, se sont développées lors de ces dernières années. Ce domaine est actuellement en plein essor; on peut citer notamment les travaux de [Zenios et Lasken 88], [Bertsekas et Tsitsiklis 89], [Tseng 90], [Bertsekas et al. 94], [Alizadeh et Goldberg 93] et [Shiloach et Vishkin 82].

Il faut noter que la nature des applications et les caractéristiques des méthodes de résolution guident le choix vers certaines techniques du parallélisme. Dans ce travail, nous avons tenu à étudier deux classes de problèmes et les méthodes de résolution associées, correspondant à deux approches très différentes du parallélisme. Pour ce qui nous concerne, les deux facteurs principaux sont la régularité de l'application et la granularité des activités.

Nous verrons par la suite que ces paramètres, qui sont tout à fait différents pour les deux applications considérées, ont induit des choix de parallélisation très variés.

Le problème de flot à coût minimum a été traité à l'aide de méthodes itératives de type relaxation ou gradient qui ont fait leur preuve (cf. [Bertsekas et Tsitsiklis 89]). Ces techniques de résolution font apparaître des calculs à gros grain et de nature foncièrement régulière. Notre contribution a porté sur la mise en œuvre d'algorithmes parallèles synchrones ou asynchrones sur deux types d'architecture : une machine à mémoire distribuée TNode de Telmat avec 16 processeurs et une machine à mémoire partagée de type multiprocesseur symétrique (SMP), équipée de 4 processeurs.

Nous rappelons que les itérations asynchrones, pour lesquelles les composantes du vecteur itéré sont réactualisées dans un ordre quelconque sans synchronisation sur plusieurs processeurs, sont des méthodes très générales qui ont suscité un grand intérêt au cours de ces dernières années (cf. [Miellou 74], [Bertsekas et El Baz 87], [Chajakis et Zenios 91], [Giraud et Spiteri 91], [Frommer 91], [Bahi et Miellou 93] ...)

Nous nous sommes également intéressés à la mise en œuvre d'une nouvelle classe de méthodes itératives parallèles asynchrones : les *itérations asynchrones avec communication flexible*, pour lesquelles les valeurs partielles des composantes du vecteur itéré peuvent aussi être échangées entre les différents processeurs. Ces méthodes ont été introduites dans [Miellou et al. 94] dans un contexte lié aux $\mathcal{M}$-fonctions.

Notre contribution a également porté sur l'expérimentation et l'étude de performances des méthodes présentées ci-dessus.

Le problème de flot maximum/coupe minimale a été traité à l'aide de méthodes de type préflot (cf. [Goldberg 85], [Goldberg et Tarjan 88]) dont le concept a été introduit par Karzanov (cf. [Karzanov 74]), ainsi que des méthodes basées sur la chaîne améliorante (cf. [Dinic 70], [Ahuja et Orlin 91]), initiées par les travaux de Ford et Fulkerson (cf. [Ford et Fulkerson 56]). Ces méthodes font apparaître essentiellement des calculs de grain fin et de nature irrégulière.

Notre contribution a porté ici sur l'étude des performances des différentes méthodes séquentielles en vue de la parallélisation de l'approche la plus efficace, à savoir une méthode de type préflot. Nous avons proposé des éléments de parallélisation de la méthode préflot sur machine à mémoire partagée de type multiprocesseur symétrique. Notre approche repose sur l'utilisation d'une librairie de threads pour gérer le parallélisme à grain fin des activités, ceci afin de garantir un bon taux d'occupation des différents processeurs.

Le manuscrit de thèse est organisé de la manière suivante :

- La première partie est consacrée au problème (non linéaire) de flot de coût minimum à critère convexe et à sa résolution par plusieurs méthodes itératives parallèles.

  Dans le chapitre 1, la théorie de la dualité nous permet de transformer le problème (primal) en un problème dual. Ce dernier, qui est différentiable et sans contrainte, peut alors être résolu au moyen de différentes méthodes itératives de descente (notamment de type gradient et relaxation) qui se prêtent bien à la parallélisation. Nous considérons alors les méthodes itératives parallèles synchrones et asynchrones, puis introduisons une nouvelle classe de méthodes basées sur la notion de sous-applications. Ces méthodes nous permettent de bâtir une nouvelle classe de méthodes itératives parallèles asynchrones : les *itérations asynchrones avec communication flexible.*

Le chapitre 2 regroupe les détails de mise en oeuvre des méthodes introduites au chapitre précédent sur deux architectures parallèles : le TNode de Telmat et un multiprocesseur SMP. Nous analysons les résultats obtenus par l'application des méthodes itératives synchrones, asynchrones et asynchrones avec communication flexible sur divers problèmes.

– La seconde partie s'intéresse à l'étude du problème (linéaire) de flot maximum/coupe minimale et à la parallélisation d'une des méthodes de résolution.

  Le problème et les résultats théoriques fondamentaux sont rappelés dans le chapitre 3. Nous décrivons les algorithmes de base appartenant aux deux classes de méthodes : celles basées sur la chaîne améliorante (ou augmentante) et celles basées sur la notion de préflot.

  Le chapitre 4 présente tout d'abord des résultats sur les performances séquentielles de plusieurs algorithmes résolvant efficacement le problème de flot maximum. Puis, il introduit des éléments concernant la parallélisation d'une méthode de type préflot à l'aide de processus légers sous Unix (architecture SMP).

Première partie

Optimisation non linéaire dans les graphes : résolution de problèmes de flot de coût minimum

Chapitre 1

Problèmes de flot convexes et algorithmes

Introduction

Dans ce chapitre, nous nous intéressons à une classe particulière de problèmes d'optimisation : les problèmes de flot dans les réseaux à critère convexe. Ces derniers s'insèrent dans la classe des problèmes d'optimisation où le critère (ou *fonction de coût*) à minimiser/maximiser est non linéaire de nature convexe et les contraintes qui s'appliquent aux variables de décision (jouant sur la valeur du critère) sont linéaires.

Les problèmes de flot convexes dans les réseaux interviennent dans de nombreux domaines : distribution, modèles financiers, réseaux de communication et réseaux de transport. Ces problèmes requièrent généralement des calculs intensifs du fait de leur taille et de la non linéarité du critère (cf. [Zenios et Mulvey 88]). Aussi la résolution de ces problèmes de manière parallèle semble-t-elle particulièrement intéressante (cf. [Bertsekas et al. 94], [Tsitsiklis et Bertsekas 86] et [Zenios et Lasken 88]).

Dans un premier temps nous présentons le problème sous sa forme *primale* et utilisons la théorie de la dualité pour en extraire le problème *dual*. Ces deux problèmes sont intimement liés et la résolution de l'un ou de l'autre conduit à la solution optimale recherchée. Dans notre cas, le problème dual possède des propriétés intéressantes : il est sans contrainte, différentiable et bien adapté à une résolution mettant en jeu des méthodes itératives parallèles. Parmi les méthodes de *descente* qui s'appliquent aux problèmes d'optimisation sans contrainte, nous avons retenu la méthode de *relaxation* (coordinate descent) et la méthode de *gradient* (steepest descent) qui se prêtent à une mise en œuvre parallèle. Pour bénéficier des mêmes propriétés de convergence qu'en séquentiel, les mises en œuvres parallèles nécessitent généralement une synchronisation des processeurs. Pour éviter que certains processeurs passent une part importante de leur temps en attente de synchronisation, il est toutefois possible, sous certaines conditions, de supprimer ces synchronisations : l'algorithme itératif correspondant devient un algorithme itératif *asynchrone* (on parle également d'*itérations asynchrones*). Bertsekas et El Baz (cf. [Bertsekas et El Baz 87] et

[El Baz 96a]) ont montré en particulier que le problème dual peut être résolu au moyen d'algorithmes itératifs de relaxation et de gradient asynchrones. Après avoir rappelé le modèle des itérations asynchrones classique et présenté une nouvelle classe d'applications de point fixe permettant de résoudre le problème dual, nous proposons d'appliquer un nouveau type de méthode itératives asynchrones : les itérations asynchrones avec communication flexible. Ces méthodes ont été introduites dans [Miellou et al. 94] dans le contexte des $\mathcal{M}$-fonctions.

1.1 Le problème de flot à critère convexe

1.1.1 Formulation du problème

Tout au long de cette partie nous considérons un graphe connexe orienté $\mathcal{G}$ défini par un ensemble de sommets $\mathcal{N}$ et un ensemble d'arcs $\mathcal{A}$ (cf. Annexe pour une introduction sur les graphes). Ce graphe est noté $\mathcal{G} = (\mathcal{N}, \mathcal{A})$. Dans la suite du document, $|\mathcal{X}|$ dénote le cardinal d'un ensemble $\mathcal{X}$. Les sommets du graphe $\mathcal{G}$ sont numérotés de 1 à n où $n = |\mathcal{N}|$ dénote le nombre total de sommets. Le nombre d'arcs du graphe $\mathcal{G}$ est noté $m = |\mathcal{A}|$. A tout arc (i,j) est associée une fonction de coût $c_{ij} : \mathbb{R} \to (-\infty, +\infty]$, fonction du flot x_{ij} traversant l'arc (i,j). A chaque sommet i est associée une constante b_i représentant la quantité de produit, fournie par le sommet i si $b_i > 0$ (i est alors appelé sommet *producteur*) ou consommée par le sommet i si $b_i < 0$ (i est alors appelé sommet *consommateur*). Les sommets i tel que $b_i = 0$ sont couramment appelés sommets de *transit*.

Le problème consiste à établir le vecteur flot $x = \{x_{ij} | (i,j) \in \mathcal{A}\}$ qui minimise le coût total sous la contrainte de conservation de flot en chaque sommet du graphe, soit :

$$\boxed{\begin{aligned} &\min \sum_{(i,j)\in\mathcal{A}} c_{ij}(x_{ij}), && (1.1a)\\ &\text{sous} \sum_{j|(i,j)\in\mathcal{A}} x_{ij} - \sum_{k|(k,i)\in\mathcal{A}} x_{ki} = b_i \quad \text{pour tout } i \in \mathcal{N}. && (1.1b)\end{aligned}}$$

Le critère à minimiser est dit *séparable* car chacune des variables n'intervient que dans l'un des termes de la sommation. Un vecteur flot x dont les composantes satisfont les contraintes de conservation de flot (1.1b) est appelé *solution admissible*. Un vecteur flot admissible x qui minimise le critère est une *solution optimale*. Notons que la sommation des n contraintes (1.1b) implique que $\sum_{i\in\mathcal{N}} b_i = 0$. Cette dernière condition est nécessaire pour que le problème soit admissible.

Ce type de problème est illustré dans l'exemple de la figure (1.1). Les sommets 3 et 4 sont des sommets producteurs (identifiés par une flêche pointillée entrante) et les sommets 9 et 10 sont des sommets consommateurs (flêche pointillée sortante). Les autres sont des sommets de transit.

Avant de préciser les hypothèses concernant la fonction de coût c_{ij} en chaque arc (i,j) du graphe, nous rappelons, dans le paragraphe qui suit, quelques notions importantes

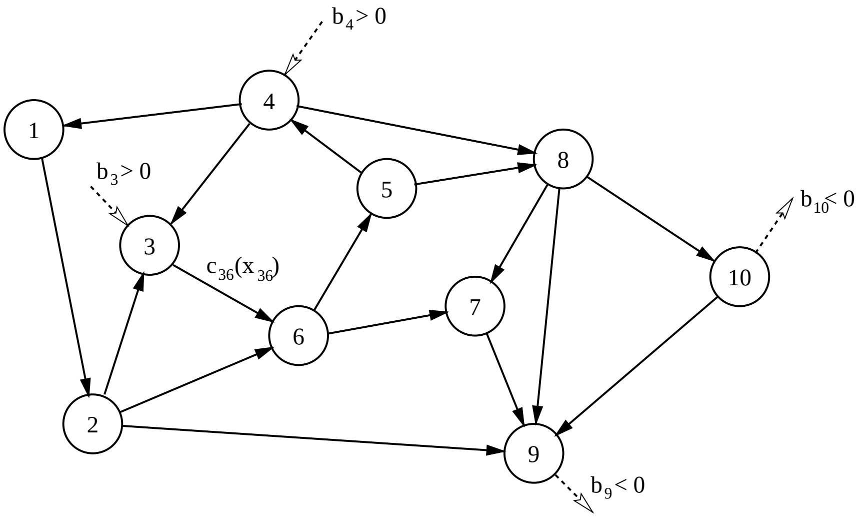

FIG. 1.1 – *Représentation graphique d'un problème de flot convexe dans un réseau.*

concernant la *convexité* d'une fonction.

1.1.2 Rappels sur les fonctions convexes

Nous présentons dans ce paragraphe les éléments essentiels de la théorie de la convexité utiles pour la suite. Le lecteur désirant plus de détails peut se reporter à l'ouvrage de référence de Rockafellar [Rockafellar 70] ainsi qu'à [Rockafellar 84].

Définition 1.1.1 ***Fonction convexe :*** *Soit f une fonction de Δ dans $(-\infty, +\infty]$, où Δ est un ensemble convexe (par exemple $\Delta = \mathbb{R}$). La fonction f est* convexe *si et seulement si*

$$f((1-\lambda)x + \lambda y) \leq (1-\lambda)f(x) + \lambda f(y), \quad \forall \lambda \in (0,1), \forall x, y \in \Delta \tag{1.2}$$

(cf. interprétation graphique sur la figure 1.2). Une fonction f est strictement convexe *si l'équation (1.2) est vérifiée avec la stricte inégalité.*

Remarque 1.1.1 La fonction f peut prendre pour valeur $+\infty$ mais pas $-\infty$, ce qui nous permet de définir la convexité sans ambiguité (on évite l'opération $\infty - \infty$) grâce à l'équation (1.2). Une telle fonction est communément appelée fonction *convexe propre.*

Dans la suite on considère $\Delta = \mathbb{R}$, et on définit $C = \{x \in \mathbb{R} | f(x) < +\infty\}$. L'ensemble C est obligatoirement un intervalle que l'on appelle *domaine effectif* de f. Les extrémités droite et gauche de C sont notées respectivement c^+ et c^-. Notons toutefois que l'intervalle C n'est pas forcément fermé et peut ne contenir ni c^+ ni c^-, même si ces valeurs sont finies (cf. figure 1.3).

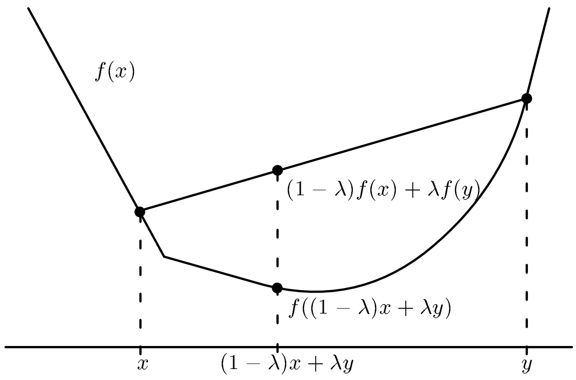

FIG. 1.2 – *Exemple de fonction convexe*

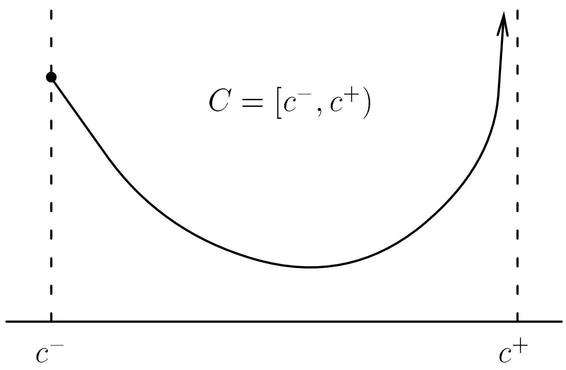

FIG. 1.3 – *Intervalle C non fermé à droite*

1.1.2.1 Propriétés sur les dérivées

Pour une fonction f de domaine effectif C, la convexité est équivalente à l'existence de dérivées à gauche et à droite croissantes. Si f'_- et f'_+ représentent respectivement les dérivées à gauche et à droite d'une fonction convexe f sur C, elles satisfont :

$$f'_-(x) \leq f'_+(x) \text{ pour tout } x \in C$$

$$f'_+(x') \leq f'_-(x'') \text{ pour tout } x', x'' \in C \text{ avec } x' < x''.$$

Les fonctions f'_- et f'_+ peuvent s'étendre à $\mathbb{R}$ tout entier en définissant

$$f'_-(x) = f'_+(x) = +\infty \text{ pour tout } x \notin C \text{ avec } x \geq c^+,$$

$$f'_-(x) = f'_+(x) = -\infty \text{ pour tout } x \notin C \text{ avec } x \leq c^-.$$

Rappelons que dans le cas où f est une fonction à valeur réelle deux fois continûment différentiable sur un intervalle ouvert (a, b) alors f est convexe si et seulement si sa dérivée seconde est positive sur (a, b).

1.1.2.2 Continuité sur $\mathbb{R}$

Définition 1.1.2 Semi-continuité inférieure :

Soit f une fonction quelconque de $\mathbb{R}$ dans $[-\infty, +\infty]$. On dit que f est semi-continue inférieure sur $\mathbb{R}$ si et seulement si l'ensemble $\{x|f(x) \leq \alpha\}$ est fermé pour tout $\alpha \in \mathbb{R}$.

Du fait de l'existence de dérivées à gauche et à droite finies en tout point x compris strictement entre c^- et c^+, une fonction convexe f est continue en tout point x vérifiant $c^- < x < c^+$. Il en est de même lorsque $x > c^+$ ou $x < c^-$ puisque f vaut $+\infty$. Le problème de continuité peut alors se poser aux extrémités finies c^+ ou c^- comme l'illustre la figure 1.4. En supposant qu'une fonction f est semi-continue inférieure, le cas de discontinuité qui peut apparaître en c^- ou c^+ disparaît et f est continue relativement à la fermeture de C.

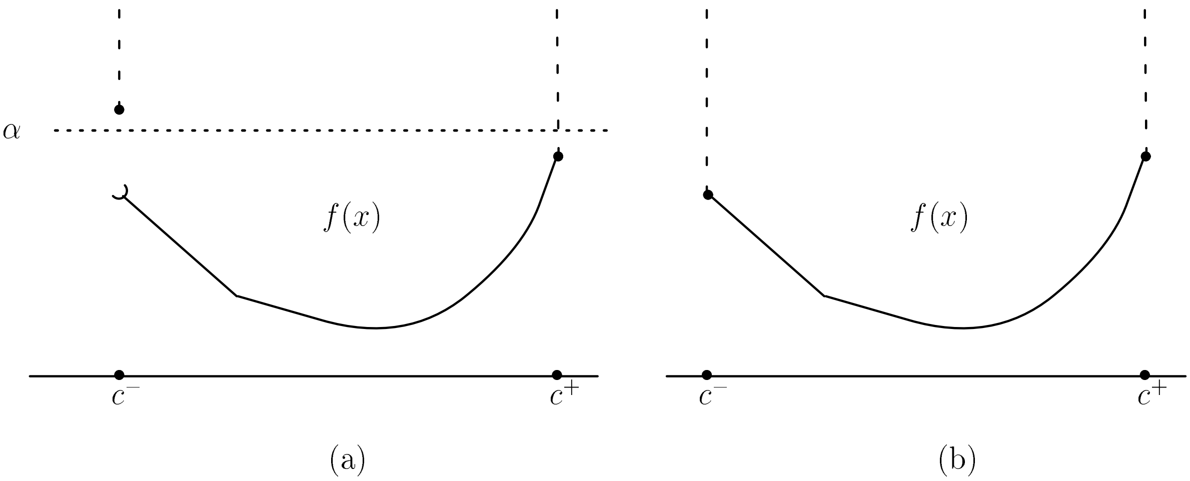

FIG. 1.4 – *Fonctions convexes de même domaine effectif* $[c^-, c^+]$ *: non semi-continue inférieure en* c^- *(a) et semi-continue inférieure (b)*

1.1.2.3 Courbe caractéristique et sous-gradient

Définition 1.1.3 *Soit f une fonction convexe, on appelle* courbe caractéristique *de f, l'ensemble Γ de points défini par :*

$$\Gamma = \{(x,t) \in \mathbb{R}^2 | f'_-(x) \leq t \leq f'_+(x)\}. \tag{1.3}$$

Cette courbe exprime l'information contenue dans les dérivées à gauche et à droite de f. Elle se rapproche de la représentation graphique d'une fonction croissante mais elle peut contenir des segments verticaux. La fonction f est en fait entièrement déterminée par la courbe caractéristique Γ, à une constante d'intégration près. La figure 1.5 représente une fonction strictement convexe et sa courbe caractéristique. Du fait de la stricte convexité, nous pouvons noter dans cet exemple que Γ n'est composée d'aucun segment de droite horizontal.

La courbe caractéristique Γ possède une courbe réciproque naturelle obtenue par symétrie par rapport à la première bissectrice et définie par :

$$\Gamma^{-1} = \{(t,x) \in \mathbb{R}^2 | (x,t) \in \Gamma\} \tag{1.4}$$

Cette courbe nous permet de déterminer l'abscisse x des points qui ont pour tangente une droite de pente t donnée. Nous remarquons que le point est unique si la fonction f est strictement convexe : il n'y a pas de segment de droite vertical dans Γ^{-1} qui est alors la représentation graphique d'une fonction croissante (cf. figure 1.6).

Définition 1.1.4 *On dit que t est un* sous-gradient *de la fonction convexe f en x si $f(x') \geq f(x) + t(x' - x)$ pour tout x'.*

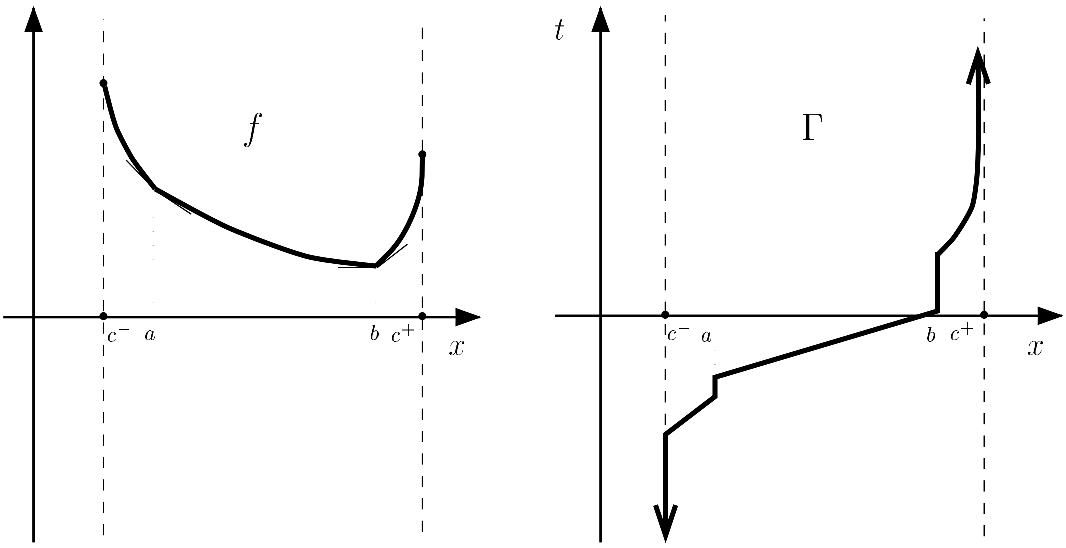

Fig. 1.5 – *Fonction f strictement convexe et sa courbe caractéristique* Γ

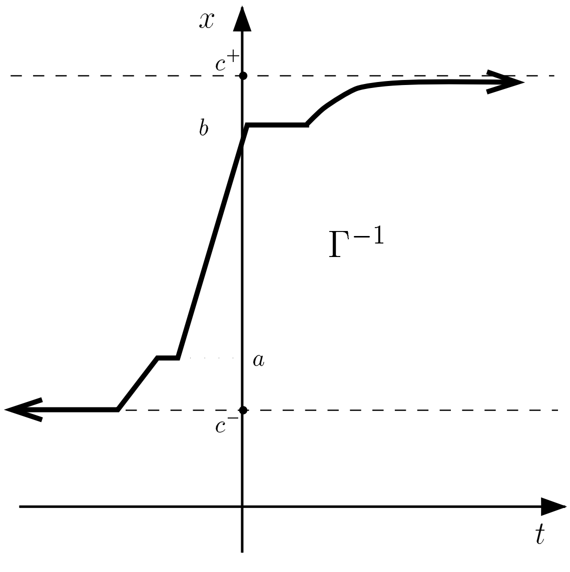

Fig. 1.6 – *Courbe caractéristique réciproque de f*

Géométriquement, la valeur t constitue la pente d'une tangente possible au point $(x, f(x))$. Nous noterons que lorsque f est dérivable en x, le sous-gradient t en x est unique alors qu'il existe un ensemble convexe fermé de sous-gradients aux points où f n'est pas différentiable (par exemple les points c^-, a et b de la figure 1.5).

1.1.2.4 Fonction convexe conjuguée

Définition 1.1.5 *Soit f une fonction convexe de domaine effectif C, on appelle fonction* convexe conjuguée *de f, la fonction f^* de domaine effectif D définie par :*

$$f^*(t) = \sup_{x\in\mathbb{R}}\{xt - f(x)\} = -\inf_{x\in\mathbb{R}}\{f(x) - xt\} \tag{1.5}$$

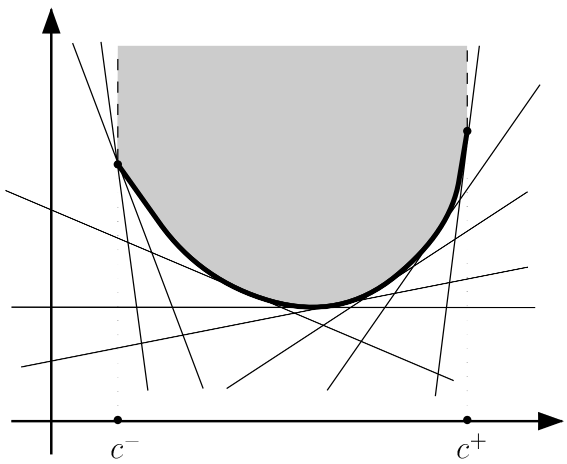

FIG. 1.7 – *Vue d'une fonction convexe par l'enveloppe de tangentes*

Interprétation graphique : En prenant la formulation de f^* avec l'*inf*, il est possible de donner une interprétation graphique de la fonction convexe conjuguée. Une fonction convexe f peut être non seulement caractérisée par son graphe, c'est à dire l'ensemble des points $(x, f(x))$ avec $x \in C$, mais aussi par son enveloppe de tangentes (cf. figure 1.7). Notons respectivement $t(x)$ et $b(x)$, la pente et l'ordonnée à l'origine d'une tangente en x. Il est possible d'associer à une fonction convexe f l'ensemble des points $\{(t(x), b(x))$ avec $x \in C\}$. Cherchons l'expression de b en fonction de t. Si on se fixe une droite de pente $\hat{t}$, passant par le point $(x, f(x))$, l'ordonnée à l'origine b de cette droite est égale à $f(x) - x\hat{t}$. A présent, si on se fixe une tangente de pente $\hat{t}$ l'ordonnée à l'origine de cette droite est égale à $\min_{x\in C}\{f(x) - x\hat{t}\}$ (voir figure 1.8). Soulignons que l'abscisse $\hat{x}$ du point de tangence peut être obtenue par lecture de la courbe caractéristique réciproque Γ^{-1} associée à la fonction convexe f. Finalement, nous pouvons caractériser la fonction convexe f par une fonction

$b(t) = \min_{x \in C}\{f(x) - xt\}$. Une telle fonction est *concave* (minimum de fonctions affines). En prenant son opposée, nous obtenons l'expression de la fonction convexe conjuguée de f soit $-b(t) = -\min_{x \in C}\{f(x) - xt\} = f^*(t)$.

Propriétés : Considérons une fonction convexe f et sa convexe conjuguée f^* alors :

- la fonction convexe conjuguée de f^* est f.
- si Γ est la courbe caractéristique de f, alors Γ^{-1}, est la courbe caractéristique de f^*.
- pour tout x et t, on a $f(x) + f^*(t) \geq xt$. De plus, $f(x) + f^*(t) = xt$ si et seulement si le point (x, t) est élément de la courbe caractéristique Γ.
- si f est strictement convexe alors f^* est continûment différentiable.

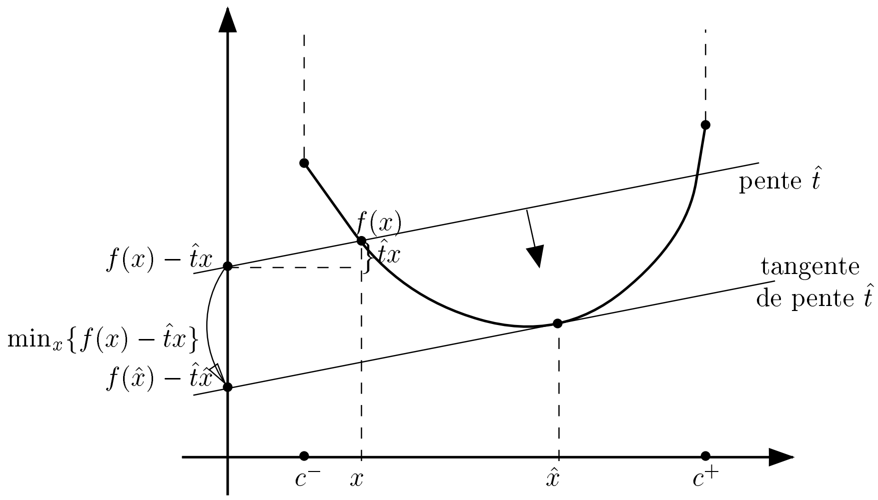

FIG. 1.8 – *L'ordonnée à l'origine de la droite de pente $\hat{t}$, tangente au graphe de f n'est autre que $-f^*(\hat{t}) = f(\hat{x}) - \hat{t}\hat{x}$. $\hat{t}$ est un sous-gradient de f en $\hat{x}$.*

1.1.3 Retour au problème : hypothèses sur les fonctions de coût c_{ij}

Nous supposons que le problème (1.1) a une solution admissible et nous considérons les hypothèses suivantes sur chacune des fonctions de coût c_{ij} :

Hypothèse 1.1.1 c_{ij} *est semi-continue inférieure;*

Hypothèse 1.1.2 c_{ij} *est une fonction strictement convexe;*

Hypothèse 1.1.3 *la fonction convexe conjuguée de c_{ij} définie par*

$$c^*_{ij}(t_{ij}) = \sup_{x_{ij}}\{x_{ij}.t_{ij} - c_{ij}(x_{ij})\}, \tag{1.6}$$

est à valeur réelle, c'est à dire que $-\infty < c^*_{ij}(t_{ij}) < +\infty$ pour tout réel t_{ij}.

L'hypothèse 1.1.1 nous assure que la fonction convexe c_{ij} est continue sur son domaine effectif. De par l'hypothèse 1.1.2, la courbe caractéristique réciproque Γ^{-1}_{ij} associée à c_{ij} est la représentation graphique d'une fonction croissante continue. Il en résulte que la fonction convexe conjuguée c^*_{ij} (de courbe caractéristique Γ^{-1}_{ij}) est continûment différentiable. Le gradient de c^*_{ij} au point t_{ij}, noté $\nabla c^*_{ij}(t_{ij})$, est alors l'unique x_{ij} qui atteint le suprémum dans l'équation (1.6) et le graphe de ∇c^*_{ij} n'est autre que Γ^{-1}_{ij}. Enfin l'hypothèse 1.1.3 implique que $\lim_{|x_{ij}|\to\infty} c_{ij}(x_{ij}) = +\infty$, par conséquent, les ensembles[1] circonscrits par les courbes de niveau de la fonction critère du problème 1.1 sont tous bornés (cf. [Rockafellar 70], chapitre 8). Il s'ensuit que le problème 1.1 possède une solution optimale qui est unique en raison de l'hypothèse 1.1.2.

Ces hypothèses sont peu restrictives hormis celle de stricte convexité. En effet, elles sont satisfaites en particulier dans les deux cas suivants (cf. [Bertsekas et El Baz 87]) :

– le cas *contraint* en capacité, où c_{ij} est de la forme

$$c_{ij}(x_{ij}) = \begin{cases} \hat{c}_{ij}(x_{ij}) & \text{si } x_{ij} \in [l_{ij}, u_{ij}], \\ +\infty & \text{sinon,} \end{cases} \tag{1.7}$$

  où l_{ij} et u_{ij} sont respectivement les bornes inférieures et supérieures du flot de l'arc (i, j) et $\hat{c}_{ij}(x_{ij})$ est une fonction strictement convexe à valeur réelle.

– le cas *non contraint* en capacité, où c_{ij} est une fonction strictement convexe à valeur réelle dont les dérivées à droite et à gauche c'_{ij+} et c'_{ij-} satisfont

$$\lim_{x_{ij}\to+\infty} c'_{ij+}(x_{ij}) = +\infty, \qquad \lim_{x_{ij}\to-\infty} c'_{ij-}(x_{ij}) = -\infty, \tag{1.8}$$

1.2 Le problème dual

1.2.1 Formulation du problème

La dualisation Lagrangienne du problème (1.1) par rapport aux contraintes de conservation de flot (1.1b) nous conduit à introduire le vecteur p de composantes p_i avec $i \in \mathcal{N}$. Chaque composante p_i est un multiplicateur de Lagrange associé à la i-ème contrainte. Le Lagrangien $L(x, p)$ est défini par :

1. Il s'agit d'ensembles non vides de la forme $\{x|f(x) \leq \alpha\}$ pour $\alpha \in \mathbb{R}$ où $x = \{x_{ij}|(i,j) \in \mathcal{A}\}$ et $f(x) = \sum_{(i,j)\in\mathcal{A}} c_{ij}(x_{ij})$.

$$
\begin{aligned}
L(x,p) &= \sum_{(i,j)\in\mathcal{A}} c_{ij}(x_{ij}) + \sum_{i\in\mathcal{N}} p_i \left(\sum_{k|(k,i)\in\mathcal{A}} x_{ki} - \sum_{j|(i,j)\in\mathcal{A}} x_{ij} + b_i \right) \\
&= \sum_{(i,j)\in\mathcal{A}} (c_{ij}(x_{ij}) - (p_i - p_j)x_{ij}) + \sum_{i\in\mathcal{N}} b_i p_i.
\end{aligned}
\tag{1.9}
$$

Le vecteur p est appelé vecteur *prix* ou *potentiel*. La valeur de la fonction duale $d(p)$ en un vecteur prix p s'obtient en minimisant $L(x,p)$ par rapport à x, soit :

$$
\begin{aligned}
d(p) &= \min_x L(x,p) = -\max_x \{-L(x,p)\} \\
&= -\max_{\substack{x_{ij} \\ (i,j)\in\mathcal{A}}} \left\{ \sum_{(i,j)\in\mathcal{A}} ((p_i - p_j)x_{ij} - c_{ij}(x_{ij})) \right\} + \sum_{i\in\mathcal{N}} b_i p_i \\
&= -\sum_{(i,j)\in\mathcal{A}} \max_{x_{ij}} \{(p_i - p_j)x_{ij} - c_{ij}(x_{ij})\} + \sum_{i\in\mathcal{N}} b_i p_i.
\end{aligned}
$$

Ce qui donne en faisant apparaître la fonction convexe conjuguée c_{ij}^* de c_{ij} :

$$
d(p) = -\sum_{(i,j)\in\mathcal{A}} c_{ij}^*(p_i - p_j) + \sum_{i\in\mathcal{N}} b_i p_i. \tag{1.10}
$$

Dans la théorie de la dualité, le problème dual consiste à maximiser la fonction duale $d(p)$ et ceci, sans contrainte sur le vecteur $p = \{p_i | i \in \mathcal{N}\}$. En posant $q(p) = -d(p)$, le problème dual[2] que nous allons résoudre s'écrit :

$$
\boxed{
\begin{aligned}
& \min_{p\in\mathbb{R}^n} q(p), && (1.11) \\
& \text{sans contrainte sur le vecteur } p = \{p_i | i \in \mathcal{N}\}, \\
& \text{avec } q(p) = \sum_{(i,j)\in\mathcal{A}} c_{ij}^*(p_i - p_j) - \sum_{i\in\mathcal{N}} b_i p_i. && (1.12)
\end{aligned}
}
$$

Les composantes du vecteur p sont les *variables duales*. Sous cette forme, nous voyons sans difficulté que la fonction $q(p)$ est une fonction convexe. De plus, d'après les caractéristiques des fonctions convexes conjuguées c_{ij}^*, $q(p)$ est une fonction *continûment différentiable* (cf. [Bertsekas et El Baz 87]).

1.2.2 Conditions d'optimalité

La dualité entre les problèmes (1.1) et (1.11) a été étudiée de manière très détaillée par Rockafellar dans le cadre de l'analyse convexe. Il a notamment établi la condition nécessaire

2. Noter que $\max_p d(p) = -\min_p q(p)$.

et suffisante d'optimalité d'une paire (x,p) (cf. [Rockafellar 70]), que nous rappelons ci-dessous :

Proposition 1.2.1 *un vecteur flot* admissible $x = \{x_{ij}|(i,j) \in \mathcal{A}\}$ *est optimal pour (1.1) et un vecteur prix* $p = \{p_i|i \in \mathcal{N}\}$ *est optimal pour (1.11) si et seulement si pour tout* $(i,j) \in \mathcal{A}$, *le point de coordonnée* $(x_{ij}, p_i - p_j)$ *est élément de la courbe caractéristique* Γ_{ij} *associée à la fonction de coût* c_{ij}. *Ceci est équivalent aux deux conditions qui suivent :*

$$p_i - p_j \text{ est un sous-gradient de } c_{ij} \text{ en } x_{ij} \text{ pour tout } (i,j) \in \mathcal{A} \tag{1.13}$$

$$x_{ij} = \nabla c^*_{ij}(p_i - p_j), \text{ pour tout } (i,j) \in \mathcal{A}. \tag{1.14}$$

Ces conditions d'optimalité, appelées aussi conditions des *écarts complémentaires*, jouent un rôle capital dans la théorie de la dualité. De plus, lorsqu'une méthode résoud le problème dual, elle fait en sorte de satisfaire ces conditions en permanence et cherche à vérifier les conditions d'admissibilité (primales).

1.2.3 Caractérisation des solutions duales

Avant de nous plonger dans la résolution du problème dual, analysons la structure de l'ensemble des solutions duales. Les hypothèses 1.1.1 – 1.1.3 nous assurent que le problème primal (1.1) possède une solution optimale unique.

Toutefois, il n'existe une solution optimale pour le problème dual (1.11) que si et seulement si le problème primal admet une solution *admissible régulière* (cf. [Rockafellar 84] p. 329 et p. 360). C'est à dire s'il existe un vecteur $x = \{x_{ij}|(i,j) \in \mathcal{A}\}$ admissible vérifiant $c'_{ij-}(x_{ij}) < +\infty$ et $c'_{ij+}(x_{ij}) > -\infty$ pour tout $(i,j) \in \mathcal{A}$, où c'_{ij-} et c'_{ij+} dénotent respectivement les dérivées à gauche et à droite de c_{ij}. Cette hypothèse supplémentaire implique que les composantes du vecteur optimal dual sont toutes bornées.

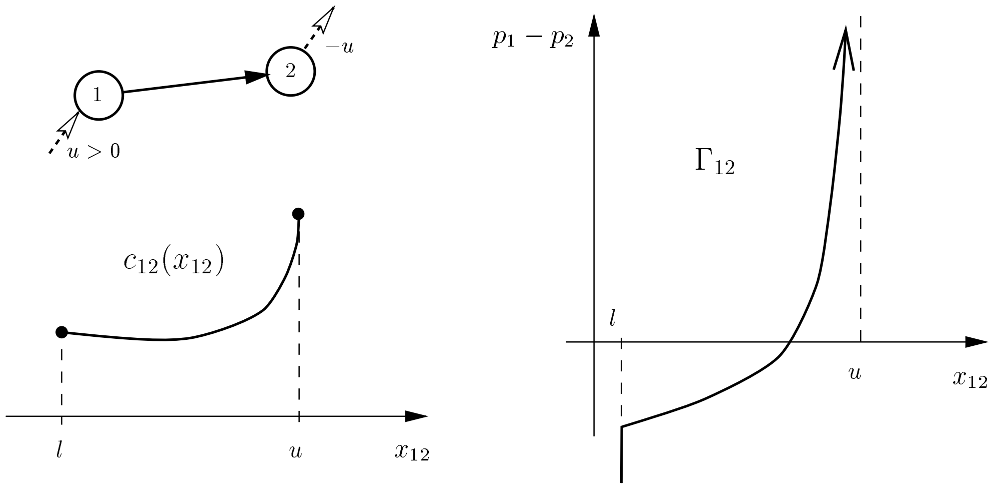

FIG. 1.9 – *Exemple de problème sans solution admissible primale régulière*

La figure 1.9 représente un cas simple où le problème primal n'admet pas de solution admissible régulière. En effet, la seule solution primale admissible, qui est aussi la solution optimale, vaut $x_{12} = u$ et la dérivée à gauche $c'_{12-}(u) = +\infty$. D'après la courbe caractéristique Γ_{12} associée à c_{12}, la solution optimale duale n'existe pas : le sous-gradient $p_1 - p_2$ associé à $x_{12} = u$ est en effet infini.

Par ailleurs, l'expression de $q(p)$ (1.12) (en utilisant le fait que $\sum_{i \in \mathcal{N}} b_i = 0$) révèle que si $\{p_i^* | i \in \mathcal{A}\}$ est un ensemble de prix optimaux, il en est de même pour l'ensemble $\{p_i^* + \delta | i \in \mathcal{N}\}$ où δ est une constante quelconque donnée. Par conséquent, la solution optimale du problème dual (lorsqu'elle existe) n'est pas unique : l'ensemble des solutions duales optimales est non borné. Afin de supprimer ce degré de liberté qui est dû au fait que seules $(n-1)$ contraintes de conservation de flot sont linéairement indépendantes, nous allons nous intéresser dans la suite à la résolution du problème dual en fixant une composante du vecteur p. Sans perte de généralité, nous supposerons fixé, le prix p_d d'un sommet destination d. Ceci conduit à la résolution d'un problème *dual réduit*.

1.2.4 Problème dual réduit et ensemble réduit de solutions duales

Pour limiter l'ensemble des solutions optimales et bénéficier de propriétés intéressantes pour la convergence asynchrone que nous verrons par la suite, nous contraignons le prix d'un sommet destination $p_d = 0$. Ceci revient à supprimer une composante du vecteur p (ou de manière équivalente à supprimer la d-ième contrainte de conservation de flot) et à travailler sur l'espace $\mathbb{R}^{n-1}$. Nous considérons alors maintenant le problème *dual réduit* suivant :

$$\boxed{\begin{array}{c} \min_{p \in P} q(p) \\ \text{avec } P = \{p \in \mathbb{R}^n | p_d = 0\} \end{array}} \tag{1.15}$$

L'ensemble des solutions duales optimales réduit P^* est défini par :

$$P^* = \{p' \in P | q(p') = \min_p q(p)\}. \tag{1.16}$$

Par la suite nous utiliserons l'hypothèse supplémentaire suivante :

Hypothèse 1.2.1 *L'ensemble des solutions duales optimales réduit P^* est non vide et compact.*

[Bertsekas et El Baz 87] ont montré que l'hypothèse 1.2.1 n'est pas très restrictive. En particulier, elle est vérifiée dans le cas d'un problème où le flot de chaque arc n'est pas contraint en capacité (cf. paragraphe 1.1.3). Dans le cas de flot contraint, le problème peut être converti en problème de flot non contraint en remplaçant les contraintes de capacités par des fonctions de pénalités non différentiables (cf. [Bertsekas 82a]).

Il résulte de l'hypothèse 1.2.1, que le problème dual réduit admet un ensemble de solutions P^* dont la structure est particulière (cf. [Bertsekas et El Baz 87], Proposition 2) :

$$\text{Il existe } \overline{p} \in P^* \text{ et } \underline{p} \in P^* \text{ tel que } \underline{p} \leq p \leq \overline{p} \text{ pour tout } p \in P^*. \tag{1.17}$$

Les solutions $\underline{p}$ et $\overline{p}$ sont appelées respectivement solution minimale et solution maximale de P^*. La relation '$\leq$' dénote la relation d'ordre partiel dans $\mathbb{R}^n$:

Définition 1.2.1 Ordre partiel *dans* $\mathbb{R}^n$ *:*
Soit x et y deux vecteurs de $\mathbb{R}^n$, on définit la relation

$$x \leq y \Longleftrightarrow x_i \leq y_i \textit{ pour tout } i \in \{1, \ldots, n\}.$$

1.3 Méthodes itératives séquentielles

Nous avons mis en évidence dans le paragraphe 1.2.1 que le problème dual (1.11) est un problème de minimisation *sans contrainte* et *différentiable.* Il en est de même pour le problème dual réduit si l'on se place dans $\mathbb{R}^{n-1}$. Il est donc possible de le résoudre par des méthodes itératives de descente (cf. [Luenberger 73]). Ce type de méthode consiste, à partir d'un point initial, à déterminer une direction de déplacement suivant une règle fixée puis à effectuer un certain déplacement. Ce déplacement fait descendre le critère suivant la direction sélectionnée. Un nouveau point est ainsi atteint, à partir duquel le processus est répété. La principale différence entre ces méthodes est la règle de sélection de la direction de déplacement. Nous appelons par la suite *recherche unidirectionnelle* l'opération qui consiste à déterminer un point minimum suivant une direction donnée.

Nous pouvons réécrire le problème dual réduit sous la forme d'un problème de point fixe :

$$p = F(p),$$

où $F : \mathbb{R}^{n-1} \to \mathbb{R}^{n-1}$ est l'opérateur ou application de point fixe dont les composantes F_i sont associées à la recherche d'un minimum suivant une direction donnée. Nous utilisons par la suite les propriétés de croissance monotone d'une application F et les notions de sous-solutions et sursolutions rappelées ci-dessous :

Définition 1.3.1 *Application* monotone croissante *:*
Une application f est dite monotone croissante sur $\mathbb{R}^n$ si pour tout $x, y \in \mathbb{R}^n$:

$$x \leq y \Longrightarrow f(x) \leq f(y).$$

Définition 1.3.2 Sursolution *et* sous-solution *de f :*

Un vecteur x de $\mathbb{R}^n$ tel que $x \geq f(x)$ est appelé sursolution de f.

Un vecteur x de $\mathbb{R}^n$ tel que $x \leq f(x)$ est appelé sous-solution de f.

Enfin, dans la suite du document, nous ferons usage des notations simplificatrices suivantes :

- $\mathcal{N}_{\bar{d}}$ représente l'ensemble $\mathcal{N} - \{d\}$,

- $(\hat{p}_i; p)$ est un vecteur de $\mathbb{R}^n$ dont la i-ème composante est égale à $\hat{p}_i$ et la j-ème composante est égale à p_j pour tout $j \in \mathcal{N}_{\bar{i}}$.

1.3.1 Méthode de relaxation

1.3.1.1 Principe de la relaxation

La différentiablité et la convexité de q nous assurent qu'il est possible, à partir d'un vecteur p non optimal, de faire décroître q en changeant simplement les composantes p_i de p une à une. Cette propriété nous permet d'envisager la résolution numérique du problème dual réduit par la méthode itérative de type Gauss-Seidel suivante : étant donné un vecteur prix p, un sommet $i \in \mathcal{N}_{\bar{d}}$ est sélectionné et son prix p_i est modifié de manière à minimiser le coût dual q par rapport à la i-ème composante, les autres prix étant inchangés. Il a été démontré (cf. [Powell 73]) qu'en sélectionnant de manière cyclique chaque composante p_i de p avec $i \in \mathcal{N}_{\bar{d}}$, la suite de vecteurs engendrés converge vers la solution optimale du problème dual réduit.

Parmi les méthodes de descente spécifiques à l'optimisation sans contrainte, l'intérêt de cette méthode itérative a été établi notamment par Bertsekas dans [Bertsekas 95b], [Bertsekas et al. 94] et [Bertsekas et Tsitsiklis 89]. D'une part, la forme séparable de la fonction duale :

$$q(p) = \sum_{(i,j)\in\mathcal{A}} c^*_{ij}(p_i - p_j) - \sum_{i\in\mathcal{N}} b_i p_i,$$

révèle qu'une composante p_i n'apparaît que dans les termes $c^*_{ij}, (i,j) \in \mathcal{A}$ et $c^*_{ji}, (j,i) \in \mathcal{A}$, limitant ainsi les calculs lorsque p_i varie. D'autre part, cette méthode est particulièrement intéressante pour le problème traité ici, en raison de la simplicité de mise en œuvre puisque le choix de la direction de recherche est immédiat et correspond au parcours séquentiel des $(n-1)$ axes de l'espace dual réduit.

Soit $g(p)$ le gradient de la fonction duale $q(p)$. En notant $g_i(p)$ la i-ème composante du gradient $g(p)$, il résulte de (1.12) que :

$$g_i(p) = \frac{\partial q(p)}{\partial p_i} = \sum_{(i,j)\in\mathcal{A}} \nabla c^*_{ij}(p_i - p_j) - \sum_{(m,i)\in\mathcal{A}} \nabla c^*_{mi}(p_m - p_i) - b_i, i \in \mathcal{N}. \quad (1.18)$$

Nous remarquons que $g_i(p)$ est uniquement fonction de valeurs locales, à savoir les prix des sommets adjacents au sommet i. De plus, $g_i(p)$ est une fonction continue non décroissante de p_i, puisque $g_i(p)$ est la dérivée partielle d'une fonction convexe différentiable (cf. [Tseng et Bertsekas 87]).

Lorsque les conditions d'optimalités sont satisfaites à tout instant, la i-ème composante du gradient de la fonction duale peut s'écrire à partir de (1.14) et (1.18) en fonction du vecteur x :

$$g_i(p) = \frac{\partial q(p)}{\partial p_i} = \sum_{(i,j)\in\mathcal{A}} x_{ij} - \sum_{(m,i)\in\mathcal{A}} x_{mi} - b_i, \quad i \in \mathcal{N}. \tag{1.19}$$

Cette expression n'est rien d'autre que le *déficit* au sommet i. Ce dernier doit être nul pour tout $i \in \mathcal{N}$ afin que les contraintes de conservation de flot soient satisfaites (rendant le vecteur x admissible). Nous pouvons alors reformuler le schéma itératif décrit précédemment de la manière suivante :

- étant donné un vecteur p, choisir un sommet $i \in \mathcal{N}_{\bar{d}}$ tel que le déficit $g_i(p)$ est non nul,
- rechercher une valeur $\hat{p}_i$ annulant le déficit du sommet i, les autres prix étant maintenus constants, c'est à dire telle que $g_i(\hat{p}_i; p) = 0$.

Une telle méthode est appelée méthode de *relaxation* car la minimisation de $q(p)$ suivant une composante p_i est équivalente à ajuster la solution primale x de manière à ce que la contrainte de flot au sommet i soit satisfaite, tout en *relachant* les contraintes de flot sur les autres sommets.

1.3.1.2 Applications de relaxation minimale et maximale

Nous venons de voir que le le problème dual réduit peut être résolu en minimisant de manière cyclique la fonction duale $q(p)$ suivant chaque axe de l'espace dual réduit [3]. Ceci revient à résoudre de manière itérative le système d'équations défini par :

$$g_i(p) = 0, i \in \mathcal{N}_{\bar{d}}, \tag{1.20}$$

où $g_i(p)$ est la i-ème composante du gradient $g(p)$ de la fonction duale $q(p)$ (ou déficit au sommet i). La recherche de la solution de ce système peut s'exprimer sous la forme d'une recherche de point fixe $\hat{p} = F(\hat{p})$ où $\hat{p}$ vérifie $g(\hat{p}) = 0$. Nous pouvons alors considérer l'application multivoque $F_i, i \in \mathcal{N}_{\bar{d}}$, qui associe à tout vecteur prix $p \in P$, l'ensemble des prix $\hat{p}_i$ qui minimisent le coût dual par rapport au i-ème prix soit :

$$\begin{aligned} F_i(p) &= \{\hat{p}_i \in \mathbb{R} | g_i(\hat{p}_i; p) = 0\} \\ &= \{\hat{p}_i \in \mathbb{R} | \sum_{(i,j)\in\mathcal{A}} \nabla c^*_{ij}(\hat{p}_i - p_j) - \sum_{(m,i)\in\mathcal{A}} \nabla c^*_{mi}(p_m - \hat{p}_i) - b_i = 0\}. \end{aligned} \tag{1.21}$$

Il est connu qu'une fonction convexe à valeur réelle pour laquelle un ensemble circonscrit par une courbe de niveau est compact, est telle que tous les ensembles circonscrits par des courbes de niveau sont compacts (cf. [Rockafellar 70], p. 70). Par conséquent, il résulte de

3. L'espace dual réduit est défini par les composantes p_i telles que $i \in \mathcal{N}_{\bar{d}}$.

l'hypothèse 1.2.1 que les ensembles $F_i(p), p \in P, i \in \mathcal{N}_{\bar{d}}$, sont des intervalles non vides et compacts. Il s'ensuit que les applications (univoques) *minimales* et *maximales* de relaxation $\underline{F}$ et $\overline{F}$, de composantes :

$$\underline{F}_i(p) = \min_{\hat{p}_i \in F_i(p)} \hat{p}_i \text{ et } \overline{F}_i(p) = \max_{\hat{p}_i \in F_i(p)} \hat{p}_i, \quad i \in \mathcal{N}_{\bar{d}}, \tag{1.22}$$

sont bien définies sur l'ensemble P. De plus, Bertsekas et El Baz ont montré que $\underline{F}$ et $\overline{F}$ sont des applications continues et monotones croissantes sur P (cf. [Bertsekas et El Baz 87]). Il s'agit d'un résultat primordial pour la démonstration de la convergence asynchrone.

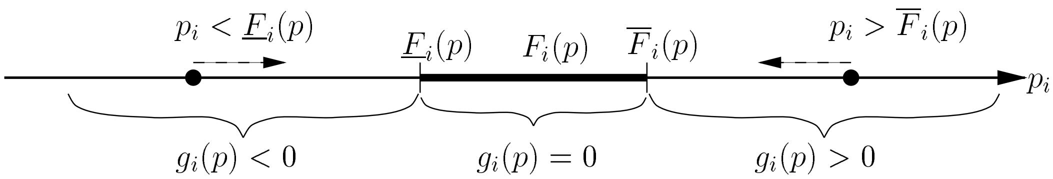

FIG. 1.10 – *Relaxation suivant p_i.*

La figure 1.10 illustre les effets de la relaxation sur la i-ème composante du vecteur p (les autres composantes sont maintenues constantes). Sur l'intervalle $[\underline{F}_i(p), \overline{F}_i(p)]$ définissant l'ensemble $F_i(p)$, le déficit $g_i(p)$ est nul par définition. Si $p_i < \underline{F}_i(p)$ alors $g_i(p) < g_i(\underline{F}_i(p); p) = 0$ du fait de la croissance monotone de g_i suivant la composante p_i. Par le même raisonnement, $g_i(p) > 0$ si $p_i > \overline{F}_i(p)$. Nous vérifions alors que si le déficit $g_i(p)$ du sommet i est positif (il sort plus de produit qu'il n'en rentre), il faut diminuer le prix p_i de manière à attirer du flot en i : décroître p_i augmente le flot des arcs entrants de i et réduit le flot des arcs sortants de i. Inversement, si le déficit est négatif, il faut augmenter p_i de manière à repousser le flot (en surplus) hors de i.

1.3.2 Méthode de relaxation approchée

Nous regroupons sous cette dénomination une classe de méthodes pour lesquelles la minimisation de $q(p)$ suivant la variable (direction) $p_i, i \in \mathcal{N}_{\bar{d}}$ s'effectue de manière inexacte (cf. [Tseng et Bertsekas 87], [Bertsekas et al. 87], [Bertsekas et Tsitsiklis 89]).

1.3.2.1 Principe de la relaxation approchée

Les applications de relaxation minimales et maximales $\underline{F}$ et $\overline{F}$ décrites précédemment peuvent dans certains cas s'avérer être des opérations coûteuses en temps de calcul. C'est notamment le cas lorsque la détermination précise du $\hat{p}_i$ annulant g_i est impossible de manière analytique. Ainsi, plutôt que de minimiser $q(p)$ de manière exacte suivant la composante p_i, on peut choisir de réduire l'amplitude du gradient $g_i(p)$ d'un certain facteur $\delta \in [0, 1)$ de la manière suivante :

- étant donné un vecteur p, choisir un sommet $i \in \mathcal{N}_{\bar{d}}$
- si $g_i(p) = 0$, on ne fait rien. Sinon, on calcule $\tilde{p}_i$ tel que

$$\delta g_i(p) \le g_i(\tilde{p}_i; p) \le 0 \text{ si } g_i(p) < 0,$$
$$0 \le g_i(\tilde{p}_i; p) \le \delta g_i(p) \text{ si } g_i(p) > 0.$$

Si on note $\tilde{F}$ les applications de relaxation inexacte (ou approchée), les composantes $\tilde{F}_i, i \in \mathcal{N}_{\bar{d}}$ sont alors définies, pour tout $p \in P$, par les relations suivantes :

$$\delta g_i(p) \le g_i(\tilde{F}_i(p); p) \le 0 \text{ et } \tilde{F}_i(p) \le \underline{F}_i(p) \text{ si } g_i(p) < 0, \tag{1.23a}$$
$$\tilde{F}_i(p) = p_i \text{ si } g_i(p) = 0, \tag{1.23b}$$
$$0 \le g_i(\tilde{F}_i(p); p) \le \delta g_i(p) \text{ et } \tilde{F}_i(p) \ge \underline{F}_i(p) \text{ si } g_i(p) > 0. \tag{1.23c}$$

1.3.2.2 Exemple de relaxation approchée

La relaxation approchée que nous décrivons est inspirée de [Chajakis et Zenios 91]. Elle suppose que les fonctions de coût $c_{ij}(x_{ij})$ sont différentiables. Dans ce cas, la condition nécessaire et suffisante d'optimalité (1.13) d'une paire (x, p) (cf. Proposition 1.2.1) est alors équivalente à :

$$p_i - p_j = \nabla c_{ij}(x_{ij}) \text{ pour tout } (i, j) \in \mathcal{A}. \tag{1.24}$$

La méthode procède en sélectionnant un sommet i de déficit $g_i(p)$ non nul tel que $|g_i(p)| > \epsilon'$ où ϵ' est une constante positive donnée. Puis, elle calcule le nouveau prix $\hat{p}_i$ de manière à écouler le déficit, en le répartissant en moyenne sur l'ensemble des arcs non saturés. La méthode se termine lorsque tous les sommets vérifient $|g_i(p)| \le \epsilon'$. La procédure de calcul du nouveau prix $\hat{p}_i$ est détaillée ci-dessous et est adaptée de [Tseng 90].

Nous considérons, dans un premier temps, que tous les arcs sont contraints en capacité : l_{ij} et u_{ij} dénotent respectivement les bornes inférieures et supérieures du flot d'un arc (i, j).

Supposons que le déficit $g_i(p)$ du sommet i sélectionné est positif. La diminution de $g_i(p)$ s'obtient en diminuant le prix p_i qui a pour effet d'abaisser le flot des arcs sortants de i et d'augmenter le flot des arcs entrants dans i. Soit $\mu \in (0, 1)$ un paramètre défini par l'utilisateur et soit n_i le nombre d'arcs sortants de i qui peuvent abaisser leur flot et d'arcs entrants dans i qui peuvent augmenter leur flot. Le principe de calcul est le suivant :

a) Pour les *arcs sortants* (i, j), le nouveau flot $\hat{x}_{ij}$ doit satisfaire :

$$x_{ij} - \frac{g_i(p)}{n_i} \le \hat{x}_{ij} \le x_{ij} - \mu \frac{g_i(p)}{n_i} \le x_{ij}.$$

En posant $a = x_{ij} - \frac{g_i(p)}{n_i}$ et $b = x_{ij} - \mu\frac{g_i(p)}{n_i} \leq x_{ij}$ et en utilisant l'équation (1.24), on déduit la valeur de $\hat{p}_i(j)$ pour l'arc (i,j) :

$$\hat{p}_i(j) = p_j + \begin{cases} \nabla c_{ij}(a) & \text{si } l_{ij} < a, \\ \nabla c_{ij}(l_{ij}) & \text{si } a < l_{ij} \leq b, \\ -\infty & \text{si } b < l_{ij}. \end{cases} \tag{1.25}$$

b) Pour les *arcs entrants* (j,i), le nouveau flot $\hat{x}_{ji}$ doit satisfaire :

$$x_{ji} < x_{ji} + \mu\frac{g_i(p)}{n_i} \leq \hat{x}_{ji} \leq x_{ji} + \frac{g_i(p)}{n_i}.$$

En posant $a = x_{ji} + \mu\frac{g_i(p)}{n_i}$ et $b = x_{ij} + \frac{g_i(p)}{n_i} \leq x_{ij}$ et en utilisant l'équation (1.24), on détermine la valeur de $\hat{p}_i(j)$ pour l'arc (j,i) :

$$\hat{p}_i(j) = p_j - \begin{cases} \nabla c_{ji}(b) & \text{si } b < u_{ij}, \\ \nabla c_{ji}(u_{ij}) & \text{si } a < u_{ij} \leq b, \\ +\infty & \text{si } u_{ij} < a. \end{cases} \tag{1.26}$$

Pour assurer que la diminution du flot des arcs sortants ne soit pas inférieure à $x_{ij} - \frac{g_i(p)}{n_i}$ et que l'augmentation du flot des arcs sortants ne dépasse pas $x_{ji} + \frac{g_i(p)}{n_i}$, le nouveau prix $\hat{p}_i$ doit satisfaire :

$$\hat{p}_i = \max\left\{ \max_{j/(i,j)\in\mathcal{A}} p_i(j), \max_{j/(j,i)\in\mathcal{A}} p_i(j) \right\}.$$

Le cas du déficit négatif $g_i(p) < 0$ conduit à un raisonnement identique et nous ne le détaillerons pas ici.

Remarque 1.3.1 Si les arcs ne sont pas contraints en capacité, le calcul de $\hat{p}_i$ se réduit au calcul de $\nabla c_{ij}(a)$ dans le cas a) et au calcul de $\nabla c_{ji}(b)$ dans le cas b) car on peut considérer que les relations $l_{ij} = -\infty < a$ et $b < u_{ts} = +\infty$ sont toujours vérifiées.

1.3.3 Méthode de gradient

Un autre moyen de déterminer le minimum de $q(p)$ consiste à choisir la direction de l'accroissement maximal d'une fonction. Cette direction est donnée par la direction du gradient $g(p)$. Si on cherche le minimum de la fonction $q(p)$, on doit obtenir une amélioration en déplaçant le point p déjà obtenu dans la direction opposée au gradient. L'itération de gradient est alors définie par :

$$p(k+1) = p(k) - \frac{1}{\alpha} g(p(k)),\ k = 0, 1, \dots, \tag{1.27}$$

où α est une constante positive et k dénote le numéro d'itération. Le terme $\frac{1}{\alpha}$ désigne le *pas* de gradient qui garde ici la même valeur à chaque itération (gradient à pas fixe). L'application de gradient associée $F' : \mathbb{R}^n \to \mathbb{R}^n$ est alors définie par :

$$F'(p) = p - \frac{1}{\alpha} g(p). \tag{1.28}$$

1.4 Méthodes itératives parallèles

1.4.1 Rappel sur les schémas de calcul parallèle

1.4.1.1 Schémas itératifs classiques

En analyse numérique, la recherche de la solution d'un système d'équations linéaires ou non linéaire se ramène généralement à la recherche d'un point fixe :

$$p^* = f(p^*), \tag{1.29}$$

où f est une application de $\mathbb{R}^n$ dans $\mathbb{R}^n$ et p^* un vecteur de $\mathbb{R}^n$ de composantes $(p_1^*, p_2^*, \ldots, p_n^*)$. La résolution s'effectue au moyen d'itérations de la forme :

$$p_i \leftarrow f_i(p_1, p_2, \ldots, p_n) \quad i = 1, 2, \ldots, n\,, \tag{1.30}$$

où p_i est la i-ème composante du vecteur p et f_i la i-ème composante de f. A partir d'un vecteur initial $p(0)$, une méthode itérative construit par approximations successives une suite $\{p(k)\}$ qui converge vers p^* sous certaines conditions. La suite $\{p(k)\}$ dépend non seulement de l'ordre dans lequel sont calculés les itérés successifs au moyen de la formulation (1.30) mais aussi de l'utilisation des itérés précédemment calculés.

A l'aide d'un calculateur monoprocesseur (ou par un calcul manuel), il est naturel d'utiliser un schéma d'actualisation dans lequel les composantes de p sont évaluées de manière cyclique en tenant compte, à chaque pas, des toutes dernières évaluations. Lorsque les composantes sont mises à jour dans l'ordre naturel, on obtient le classique schéma de calcul de Gauss-Seidel. En notant k le numéro de l'*itération* ($k \geq 0$), la formule (1.30) s'écrit :

$$\boxed{\begin{cases} p_1(k+1) = f_1(p_1(k), p_2(k), \ldots, p_n(k))\,, \\ p_2(k+1) = f_2(p_1(k+1), p_2(k), \ldots, p_n(k))\,, \\ \qquad\qquad \vdots \\ p_n(k+1) = f_n(p_1(k+1), p_2(k+1), \ldots, p_{n-1}(k+1), p_n(k)). \end{cases}} \tag{1.31}$$

Suivant la formulation (1.31), une itération correspond à l'ensemble des calculs permettant de réévaluer chaque composante une fois. Ce schéma peut être également vu comme une suite de n relaxations où la relaxation i consiste à mettre à jour p_i sans modifier les autres composantes :

$$\begin{cases} p_1(k+1) = f_1(p_1(k), p_2(k), \dots, p_n(k))\,, \\ p_j(k+1) = p_j(k) \quad \text{pour tout } j \neq 1\,, \end{cases}$$
$$\begin{cases} p_2(k+2) = f_2(p_1(k+1), p_2(k+1), \dots, p_n(k+1))\,, \\ p_j(k+2) = p_j(k+1) \quad \text{pour tout } j \neq 2\,, \end{cases}$$
$$\vdots$$
$$\begin{cases} p_n(k+n) = f_n(p_1(k+n-1), p_2(k+n-1), \dots, p_n(k+n-1))\,, \\ p_j(k+n) = p_j(k+n-1) \quad \text{pour tout } j \neq n. \end{cases}$$

Cette dernière forme met bien en évidence le fait qu'on utilise toujours les valeurs les plus récentes pour le calcul des itérés.

Il est intéressant de noter que, malgré sa nature séquentielle, le schéma de calcul Gauss-Seidel peut être réalisé dans certains cas en parallèle. C'est en effet possible lorsque plusieurs composantes du vecteur p sont indépendantes. Nous considérons deux composantes comme indépendantes lorsque la réactualisation de l'une ne fait pas intervenir l'autre et inversement. Dans ce cas, si nous détectons par exemple r $(r < n)$ sous-ensembles disjoints $l_1, l_2, \dots, l_r$ de composantes indépendantes, nous pouvons actualiser les composantes du sous-ensemble l_1, puis celles du sous-ensembles l_2 et ceci en séquence jusqu'au sous-ensemble l_r. La réactualisation des composantes d'un sous-ensemble peut s'effectuer en parallèle sur une architecture dotée de moins de n processeurs. On voit aisément que le schéma de calcul obtenu correspond alors au schéma de Gauss-Seidel que l'on obtiendrait sur un monoprocesseur en réactualisant séquentiellement les composantes dans l'ordre suivant : d'abord les composantes de l_1 puis celles l_2 jusqu'à celles de l_r. La résolution parallèle nécessite donc une synchronisation des processeurs pour éviter de commencer la réactualisation des composantes d'un sous-ensemble l_j avant d'avoir terminé la réactualisation des composantes appartenant à l_{j-1}. Il est évident que lorsque les composantes de p sont totalement inter-dépendantes (il y a alors n sous-ensembles contenant chacun une seule composante), seule la résolution séquentielle permet de retrouver le schéma de calcul de type Gauss-Seidel.

Rappelons enfin que les sous-ensembles disjoints peuvent être déterminés à partir d'un prétraitement, appelé *coloration*, appliqué au graphe représentant les dépendances entre composantes. Ce graphe non orienté est composé de n sommets (un sommet par composante) et ses arêtes symbolisent les dépendances entre composantes. Deux sommets non adjacents dans ce graphe représentent alors deux composantes indépendantes. Le but de la procédure de coloration est de partitionner les sommets du graphe en r sous-ensembles étiquetés par une couleur. Chacun des r sous-ensembles disjoints est composé de sommets non adjacents de même couleur, c'est à dire de composantes indépendantes pouvant être traitées en parallèles.

D'autres schémas de calcul ont été proposés pour lesquels une plus grande liberté de choix est laissée dans l'ordre de réactualisation des composantes. Nous pouvons citer notamment les méthodes à conduite libre (cf. [Ostrowski 61]).

Sur une architecture parallèle capable de mettre à jour simultanément les n composantes du vecteur p, il est plus naturel d'envisager une méthode cyclique dans laquelle la formule (1.30) s'écrit ($k \geq 0$) :

$$\boxed{\begin{cases} p_1(k+1) = f_1(p_1(k), p_2(k), \ldots, p_n(k))\,, \\ p_2(k+1) = f_2(p_1(k), p_2(k), \ldots, p_n(k))\,, \\ \qquad\qquad \vdots \\ p_n(k+1) = f_n(p_1(k), p_2(k), \ldots, p_n(k)). \end{cases}} \tag{1.32}$$

Il s'agit alors d'un schéma itératif de type Jacobi où une itération au sens de (1.32) correspond à la réactualisation en parallèle de l'ensemble des composantes du vecteur p. En l'interprétant sur une architecture séquentielle, ce schéma nécessite n itérations de relaxation définies par :

$$\begin{cases} p_1(k+1) = f_1(p_1(k), p_2(k), \ldots, p_n(k))\,, \\ p_j(k+1) = p_j(k) \quad \text{pour tout } j \neq 1\,, \end{cases}$$
$$\begin{cases} p_2(k+2) = f_2(p_1(k), p_2(k), \ldots, p_n(k))\,, \\ p_j(k+2) = p_j(k+1) \quad \text{pour tout } j \neq 2\,, \end{cases}$$
$$\vdots$$
$$\begin{cases} p_n(k+n) = f_n(p_1(k), p_2(k), \ldots, p_n(k))\,, \\ p_j(k+n) = p_j(k+n-1) \quad \text{pour tout } j \neq n. \end{cases}$$

Sous cette forme, nous pouvons mettre en évidence que les évaluations ne prennent pas en compte les valeurs des itérés les plus récemment calculés. A la n-ème itération de relaxation par exemple, le calcul de p_n fait intervenir les valeurs de composantes calculés n itérations auparavant : un *retard* apparaît dans l'utilisation des composantes connues. Par contre, sur un calculateur parallèle doté de n processeurs, chacun s'occupant de la réactualisation d'une composante de p, les évaluations successives s'effectuent sans retard dans l'utilisation des itérés déjà connus. En effet, une seule itération de relaxation par processeur suffit à construire le prochain élément de la suite de Jacobi. Notons que le calcul de l'itéré à l'étape $k+1$ ne peut avoir lieu qu'après que chaque processeur ait pris connaissance des $n-1$ composantes calculées par les autres processeur à l'étape k : les processeurs voient le même vecteur p à chaque début d'itération. Ce schéma nécessite donc une synchronisation à chaque itération sur une architecture parallèle composée de n processeurs.

1.4.1.2 Parallélisation de la relaxation et du gradient

Dans le paragraphe 1.3, nous avons présenté trois méthodes permettant de résoudre le problème dual réduit. Nous allons voir qu'elles se prêtent bien toutes trois à une mise en œuvre parallèle :

– Cas de la relaxation : nous avons vu que le schéma itératif est de type Gauss-Seidel et opère en sélectionnant de manière cyclique chaque composante $p_i, \in \mathcal{N}_{\bar{d}}$ pour déterminer le $\hat{p}_i$ tel que $g_i(\hat{p}_i; p) = 0$. Comme évoqué dans le paragraphe précédent, le schéma de calcul de Gauss-Seidel est préservé (ainsi que la propriété de décroissance du critère) dans une mise en œuvre parallèle si on réactualise simultanément un sous-ensemble de composantes indépendantes. Ces composantes sont telles que les sommets correspondants ne sont pas adjacents puisque, comme le montre l'équation (1.21), le calcul d'une composante $\hat{p}_i$ ne fait intervenir que le prix des sommets voisins de i. Il faut donc appliquer une procédure de coloration au graphe $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ pour extraire les sous-ensembles disjoints composés de sommets non adjacents de même couleur. Par ailleurs, la mise en œuvre nécessite une synchronisation lorsqu'un nouveau sous-ensemble de composantes indépendantes doit être réévalué en parallèle. Notons que le nombre de processeurs utiles n'excède pas le cardinal du plus grand sous-ensemble coloré.

– Cas de la relaxation approchée : identique au précédent.

– Cas du gradient : on voit aisément à partir de (1.27) que le schéma itératif associé à l'opérateur de gradient est de type Jacobi. On peut donc associer à chaque sommet $i, i \in \mathcal{N}_{\bar{d}}$ du graphe, un processeur qui effectue le calcul de $g_i(p)$ et met à jour p_i. Une synchronisation est nécessaire pour que tous les processeurs débutent une nouvelle itération avec la même représentation du vecteur p.

Ces schémas itératifs parallèles dotés de phases de synchronisation pour préserver les propriétés de convergence séquentielle, appartiennent à la classe des itérations parallèles *synchrones*.

1.4.2 Itérations parallèles asynchrones (IPA)

Les méthodes itératives parallèles que nous avons présentées dans le paragraphe précédent obéissent à des schémas de calcul bien définis. Les composantes sont réactualisées selon un certain ordre et des synchronisations sont alors requises. Cependant, les temps d'attente dûs aux synchronisations peuvent être non négligeables et peuvent pénaliser les performances des méthodes itératives parallèles. Il est alors intéressant de pouvoir considérer une procédure plus générale où les calculs s'effectuent sans ordre ni synchronisation à chaque itération : certains processeurs peuvent itérer plus que d'autres et l'information disponible sur un processeur peut être arbitrairement dépassée en regard des autres processeurs (cf. figure 1.11). Le modèle correspondant est celui des algorithmes *itératifs asynchrones* encore appelés *itérations (parallèles) asynchrones* (IPA). Les premiers travaux théoriques sur les IPA (cf. [Chazan et Miranker 69]) ainsi que les premières simulations et implantations (sur le réseau ARPANET, ancêtre de l'Internet, cf. [Rosenfeld 69]) remontent à 1969. Ces algorithmes ont été conçus aussi bien pour les calculateurs parallèles que pour les systèmes distribués. Ils ont été étudiés pour résoudre des problèmes variés : résolution de systèmes d'équations linéaires (cf. [Rosenfeld 69] et [Chazan et Miranker 69]), de systèmes d'équations non linéaires (cf. [Miellou 75a], [Miellou 75b], [Baudet 78], [Miellou et Spiteri 85a] et [Spiteri et al. 95]), de problèmes de programmation dynamique (cf. [Bertsekas 82b]), de problèmes d'optimisation (cf. [Bertsekas et El Baz 87]). Pour d'autres études, on se reportera à [Frommer et Schwandt 95] et [Miellou et al. 90].

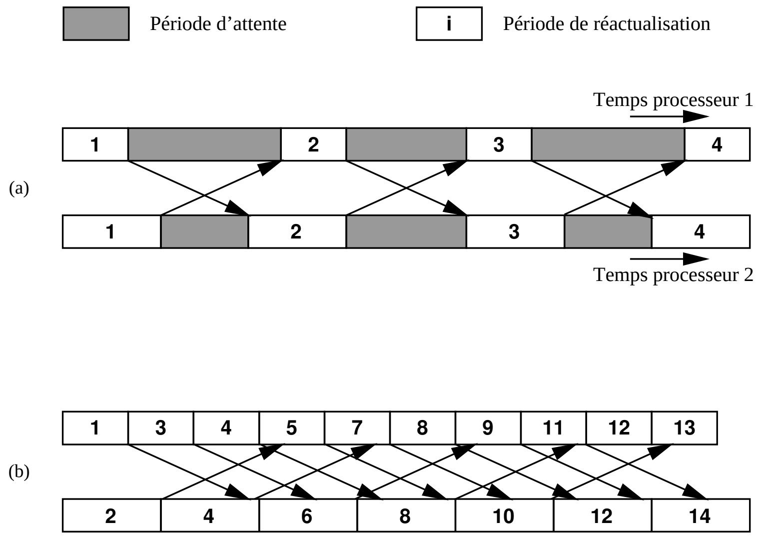

FIG. 1.11 – *(a) Algorithme synchrone : chaque processeur attend que des données précises deviennent disponibles en des points prédéterminés. (b) Algorithme asynchrone : chaque processeur va à son propre rythme.*

1.4.2.1 Formulation des IPA

Définition 1.4.1 *Un algorithme itératif asynchrone relatif à la résolution d'un problème de point fixe $p = f(p)$ où f est une application de $\mathbb{R}^n$ dans $\mathbb{R}^n$, est la suite $\{p^k\}$ des vecteurs de $\mathbb{R}^n$ définie comme suit :*

Supposons qu'il existe un ensemble $T = \{0, 1, 2, \dots\}$ composé des numéros d'itérations pour lesquels une (ou plusieurs) composante(s) p_i du vecteur p est (sont) réactualisées par un (ou plusieurs) processeurs. En notant T_i, le sous-ensemble des numéros d'itérations où la composante p_i est réactualisée et f_i, la i-ème composante de l'application f, la suite $\{p^k\}$ est définie par :

$$p_i^k = \begin{cases} f_i(p_1^{\rho_1(k)}, \dots, p_n^{\rho_n(k)}) & si\ k \in T_i \\ p_i^{k-1} & si\ k \notin T_i \end{cases} \quad avec\ i \in \{1, \dots n\} \tag{1.33}$$

Le terme $p_j^{\rho_j(k)}$ est la valeur de la j-ème composante servant à déterminer la (les) composante(s) réactualisées à l'itération k. La composante $p_j^{\rho_j(k)}$ a été réactualisée à l'itération

$\rho_j(k)$ *appelée numéro d'itération retardée.*

De plus, les conditions suivantes doivent être vérifiées pour tout $i \in \{1, \dots n\}$ *:*

(C1) l'ensemble T_i *est infini,*

(C2) $0 \leq \rho_j(k) \leq k-1$ *pour tout* $j \in \{1, \dots n\}$ *et pour tout* $k \in T_i$,

(C3) $\lim_{k \to \infty} \rho_j(k) = +\infty$ *pour tout* $j \in \{1, \dots n\}$.

La condition C1 spécifie qu'aucune composante du vecteur itéré ne doit être abandonnée définitivement. La condition C2 traduit la notion de causalité. Quant à la condition C3, elle précise que des valeurs trop anciennes des composantes du vecteur itéré doivent être abandonnées lors des calculs au fur et à mesure que ceux-ci se déroulent.

La figure 1.11b illustre le modèle en étiquetant les cases associées aux périodes de réactualisation par les numéros d'itérations.

Remarque 1.4.1

- les termes $k - \rho_j(k)$ sont appelés des retards. Ils permettent de représenter deux types de phénomènes liés au fonctionnement asynchrone : le non-déterminisme des temps de calcul et les délais dans les communications.
- la formulation des algorithmes itératifs asynchrones est très générale. Elle englobe notamment tous les algorithmes classiques. Le tableau 1.1 fournit l'expression de T_i et de $\rho_j(k)$ correspondant aux schémas séquentiels de relaxation de type Gauss-Seidel et Jacobi ainsi qu'au schéma de Jacobi parallèle synchrone sur n processeurs.

Gauss-Seidel (séquentiel)	$T_i = \{i, n+i, 2n+i, \dots\}, \quad i \in \{1, \dots n\}$ $\rho_j(k) = k-1, \quad i, j \in \{1, \dots n\}, \quad k \in T_i$
Jacobi Séquentiel	$T_i = \{i, n+i, 2n+i, \dots\}, \quad i \in \{1, \dots n\}$ $\rho_j(k) = k-i, \quad i, j \in \{1, \dots n\}, \quad k \in T_i.$
Jacobi n processeurs	$T_i = T, \quad i \in \{1, \dots n\}$ $\rho_j(k) = k-1, \quad i, j \in \{1, \dots n\}, \quad k \in T_i.$

TAB. 1.1 – *Représentation asynchrone de schémas itératifs synchrones*

1.4.2.2 Résultats de convergence

Une des difficultés des algorithmes itératifs asynchrones réside dans l'étude de leur convergence. En effet, lorsque plusieurs processeurs effectuent en parallèle des tâches de calcul non synchronisées, les ensembles T_i et les suites $\{\rho_1(k)\}, \{\rho_2(k)\}, \dots, \{\rho_n(k)\}$ sont engendrées implicitement par l'exécution de l'algorithme. L'étude de la convergence des itérations asynchrones revient à déterminer les conditions suffisantes qui garantissent la convergence vers le point fixe.

De nombreux résultats de convergence ont été établis pour des applications diverses. C'est Chazan et Miranker qui ont tout d'abord formulé un modèle d'algorithmes parallèles asynchrones (cf. [Chazan et Miranker 69]). Ils ont montré que les itérations asynchrones convergent vers la solution de systèmes linéaires $Ax = b$ si et seulement si A est une H-matrice. Dans le cas de systèmes d'équations non linéaires, la convergence a été établie lorsque A est hors diagonale monotone décroissante et diagonale monotone strictement croissante (cf. [El Baz 90]), une $\mathcal{M}$-fonction (cf. [El Baz 90]), ou H-accrétive (cf. [Miellou et Spiteri 85b]). Dans le cas de problèmes d'optimisation sans contrainte, une condition suffisante de convergence des algorithmes de gradient asynchrone est : la matrice Hessienne de la fonction coût est diagonale dominante (cf. [Bertsekas et Tsitsiklis 89] et [Bertsekas 83]). Dans le cas de problèmes de point fixe $p = F(p)$, des résultats de convergence ont été établis pour des applications F P-contractantes (cf. [Miellou 75a], [Baudet 78], [Bertsekas 83] et [Bertsekas et Tsitsiklis 89]), des applications contractantes (cf. [El Tarazi 82]), des applications continues et monotones croissantes (cf. [Bertsekas 83], [Bertsekas et Tsitsiklis 89], [Miellou 75b] et [Miellou 86]).

Nous rappelons à présent les résultats de convergence des algorithmes de relaxation et de gradient asynchrone où les calculs sont effectués concurremment sans ordre ni synchronisation suivant le modèle décrit au paragraphe 1.4.2.1.

Dans [Bertsekas et El Baz 87], Bertsekas et El Baz ont montré sous les hypothèses 1.1.1 à 1.2.1 que :

- tout algorithme de *relaxation asynchrone* associé à l'application de point fixe de composante $\underline{F}_i(p)$ et partant d'un point initial $p(0)$ sous-solution de $\underline{F}$ converge vers $\underline{p}$ qui est solution minimale de P^*.

  De même, à partir d'un point $p(0)$ sursolution de $\overline{F}$, l'algorithme converge par application de $\overline{F}$ vers $\overline{p}$ qui est la solution maximale de P^*.

- sous l'hypothèse supplémentaire que les fonctions de coût c_{ij} sont continûment différentiables, tout algorithme de relaxation asynchrone associé à l'application de point fixe de composante F_i converge vers l'unique[4] solution duale p^*.

Dans [El Baz 96a], El Baz a montré que sous les hypothèses 1.1.1 à 1.2.1 et une hypothèse supplémentaire sur le paramètre de gradient α, le gradient asynchrone (avec l'application de gradient définie par (1.36)) converge vers la solution[5] p_* si le point de départ $p(0)$ est tel que $p(0) < p_*$.

4. Si c_{ij} est continûment différentiable, c_{ij}^* est strictement convexe. Le problème dual réduit est alors strictement convexe et continûment différentiable, les applications F_i sont par conséquent des applications univoques et l'ensemble réduit des solutions duales optimales est constitué d'un unique élément p^*.

5. p_* est défini comme étant le point minimum de l'ensemble des points appartenant à l'intersection de P^* et de l'orthant positif.

1.5 Itérations asynchrones avec communication flexible (IACF)

Dans la suite, P' et P'' dénotent deux sous-ensembles de vecteurs p définis par :

$$P' = \{p \in P | p \leq \underline{p}\}, \tag{1.34}$$

$$P'' = \{p \in P | \overline{p} \leq p\}, \tag{1.35}$$

où $\underline{p}$ et $\overline{p}$ sont respectivement les solutions minimales et maximales de l'ensemble des solutions duales optimales réduit P^* (cf. équation (1.17)).

1.5.1 Sous-applications et surapplications

L'obtention du résultat exact relatif à une minimisation unidirectionnelle par les opérateurs de relaxation maximaux ou minimaux peut s'avérer difficile, en particulier lorsque la recherche de $\hat{p}_i$ tel que $g_i(\hat{p}_i; p) = 0$ ne peut s'effectuer de manière analytique exacte. Les concepts de sous-applications et surapplications nous permettent d'approcher le résultat des opérateurs décrits précédemment tout en gardant de bonnes propriétés pour la convergence des itérations asynchrones. Ils jouent notamment un rôle capital dans la définition des méthodes itératives parallèles asynchrones avec communication flexible. Les concepts de sous-application et surapplication sont généraux. Ils ont été auparavant introduits dans le contexte de résolution de systèmes d'équations non linéaires mettant en jeu des $\mathcal{M}$-fonctions (cf. [Miellou 86]). Nous allons voir qu'ils peuvent aussi apparaître dans le cadre de problèmes de minimisation à critère séparable strictement convexe et à contraintes linéaires.

1.5.1.1 Définitions

Définition 1.5.1 *Une application $\check{F}$, de composantes $\check{F}_i$, $i \in \mathcal{N}_{\bar{d}}$, est une sous-application associée à l'application de relaxation minimale $\underline{F}$ sur P', ou plus simplement que $\check{F}$ est une sous-application sur P', si pour tout $p \in P'$, sous-solution de $\underline{F}$, on a $\check{F}_i(p) \in [p_i, \underline{F}_i(p)]$ et $\check{F}_i(p) \neq p_i$ si $\underline{F}_i(p) \neq p_i$.*

Définition 1.5.2 *Une application $\hat{F}$, de composantes $\hat{F}_i, i \in \mathcal{N}_{\bar{d}}$, est une surapplication associée à l'application de relaxation maximale $\overline{F}$ sur P'', ou plus simplement que $\hat{F}$ est une surapplication sur P'', si pour tout $p \in P''$, sursolution de $\overline{F}$, on a $\hat{F}_i(p) \in [\overline{F}_i(p), p_i]$ et $\hat{F}_i(p) \neq p_i$ si $\overline{F}_i(p) \neq p_i$.*

Nous introduisons maintenant quelques propriétés se rapportant à la continuité de $\check{F}$ et $\hat{F}$. Ces propriétés sont indispensables pour l'analyse de convergence des algorithmes asynchrones avec communication flexible que nous développons au paragraphe suivant.

Définition 1.5.3 *Une sous-application $\check{F}$ est continue pour l'ordre sur P' si*

$$p(k) \uparrow p' \in P', k \to \infty \Rightarrow \check{F}_i(p(k)) \uparrow \check{F}_i(p'), k \to \infty, \text{ pour tout } i \in \mathcal{N}_{\bar{d}} ,$$

où la notation $p(k) \uparrow p', k \to \infty$, *signifie que* $p(0) \leq p(1) \leq ... \leq p(k) \leq p(k+1) \leq ... \leq p'$ *et* $\lim_{k\to\infty} p(k) = p'$.

Définition 1.5.4 *Une surapplication* $\hat{F}$ *est continue pour l'ordre sur* P'' *si*

$$p(k) \downarrow p' \in P'', k \to \infty \Rightarrow \hat{F}_i(p(k)) \downarrow \hat{F}_i(p'), k \to \infty, \text{ pour tout } i \in \mathcal{N}_{\bar{d}} \ ,$$

où la notation $p(k) \downarrow p', k \to \infty$, *signifie que* $p(0) \geq p(1) \geq ... \geq p(k) \geq p(k+1) \geq ... \geq p'$ *et* $\lim_{k\to\infty} p(k) = p'$.

Définition 1.5.5 *Une sous-application* $\check{F}$ *est minorée par une sous-application* $\check{F}'$ *continue pour l'ordre sur* P', *ou* $\check{F}$ *est dite* m-continue *sur* P' *si*

$$\check{F}'_i(p) \in [p_i, \check{F}_i(p)], \qquad \text{pour tout } p \in P' \text{ sous-solution de } \underline{F}.$$

Définition 1.5.6 *Une surapplication* $\hat{F}$ *est majorée par une surapplication* $\hat{F}'$ *continue pour l'ordre sur* P'', *ou* $\hat{F}$ *est dite* $\mathcal{M}$-continue *sur* P'' *si*

$$\hat{F}'_i(p) \in [\hat{F}_i(p), p_i], \qquad \text{pour tout } p \in P'' \text{ sursolution de } \overline{F}.$$

Nous présentons maintenant quelques sous-applications et surapplications qui seront mises en œuvre dans le chapitre suivant. Dans un premier temps, nous considèrons des applications basées sur une recherche unidirectionnelle inexacte. Nous nous intéressons ensuite à des applications construites à partir de l'application de gradient.

1.5.1.2 Exemple 1 : Méthodes de relaxation avec recherche unidirectionnelle inexacte

Considérons les applications de relaxation inexacte $\tilde{F}$ définies par les équations (1.23) au paragraphe 1.3.2.1. Nous pouvons alors montrer la proposition suivante :

Proposition 1.5.1 *Sous les hypothèses 1.1.1–1.1.3 et 1.2.1, toute application* $\tilde{F}$ *définie par* (1.23) *est une sous-application sur* P'.

Preuve : pour tout $i \in \mathcal{N}_{\bar{d}}$ et $p \in P'$ tel que $p_i \leq \underline{F}_i(p)$, nous avons $g_i(p) \leq g_i(\underline{F}_i(p);) = 0$ car g_i est une fonction non décroissante suivant la i-ème composante. De plus, on déduit de (1.22) que $g_i(p) < 0$ si $\underline{F}_i(p) \neq p_i$. Par conséquent, on obtient $p_i \leq \tilde{F}_i(p) \leq \underline{F}_i(p)$ à partir de (1.23a). Si $p_i = \underline{F}_i(p)$ alors $g_i(p) = 0$ entrainant $\tilde{F}_i(p) = p_i$ à partir de (1.23b).
□

Il est facile de démontrer par le même raisonnement que toute application $\tilde{F}$ définie par (1.23) est une surapplication sur P''.

1.5.1.3 Exemple 2 : Applications basées sur l'opérateur de gradient

Les composantes F'_i de l'application de gradient F' sont définies par :

$$F'_i(p) = p_i - \frac{1}{\alpha} g_i(p), \text{ pour tout } i \in \mathcal{N}_{\bar{d}} \text{ et } p \in P, \tag{1.36}$$

où α est une constante positive. F' est continue puisque le gradient g de la fonction duale est continu. Supposons à présent que :

Hypothèse 1.5.1 *c_{ij} est fortement convexe de module $\frac{1}{\beta}$. C'est à dire qu'il existe une constante $\beta > 0$ telle que pour tous points $(\xi, \eta), (\xi', \eta')$ éléments de la courbe caractéristique Γ_{ij} associée à c_{ij} et vérifiant $\xi' < \xi$, on ait $\eta - \eta' \geq \frac{1}{\beta}(\xi - \xi')$.*

Il a été montré (cf. [El Baz 96a], Théorème 2.2) que sous les hypothèses 1.1.1–1.1.3, 1.2.1 et 1.5.1, il existe une constante $\alpha = \beta. \max_{i \in \mathcal{N}} a_i$ (où a_i est le degré du sommet $i \in \mathcal{N}$), telle que pour tout $p, p' \in P$ satisfaisant $p' \leq p$, on a : $g(p) - g(p') \leq \alpha.(p - p')$. Il en résulte que l'application de gradient F' est monotone croissante sur P. On montre également que F' est monotone croissante sur un sous-ensemble de P si c_{ij} est fortement convexe sur un sous-domaine associé (cf. [El Baz 96a]).

Proposition 1.5.2 *Supposons vérifiées les hypothèses 1.1.1–1.1.3, 1.2.1 et 1.5.1. En prenant $\alpha = \beta. \max_{i \in \mathcal{N}} a_i$, l'application de gradient F' définie par* (1.36) *est une sous-application sur P' continue pour l'ordre.*

Preuve : Pour tout $i \in \mathcal{N}_{\bar{d}}$ et $p \in P'$ tel que $p_i \leq \underline{F}_i(p)$, on a $g_i(p) \leq g_i(\underline{F}_i(p); p) = 0$ car g_i est une fonction non décroissante suivant la i-ème composante. Par conséquent, on déduit de (1.36) que $p_i \leq F'_i(p)$. Il découle de la monotonie de F' sur P', de (1.36) et (1.22) que pour tout $i \in \mathcal{N}_{\bar{d}}$ et $p \in P'$ tel que $p_i \leq \underline{F}_i(p)$, on a :

$$F'_i(p) \leq F'_i(\underline{F}_i(p); p) = \underline{F}_i(p) - \frac{1}{\alpha} g_i(\underline{F}_i(p); p) = \underline{F}_i(p).$$

De plus, on déduit de (1.22) et (1.36) que $F'_i(p) \neq p_i$ si $\underline{F}_i(p) \neq p_i$. Enfin, l'application de gradient F' est continue pour l'ordre sur P' puisque F' est continue sur P'.

□

Par la même démarche, on prouve que l'application de gradient F' est une surapplication continue pour l'ordre sur P''.

Il est possible d'étendre la classe des sous-applications et surapplications basées sur l'application de gradient :

Proposition 1.5.3 *Supposons vérifiées les hypothèses de la proposition 1.5.2. Toute application $\check{F}$ définie de manière récursive par :*

$$\begin{aligned} &\check{F}_i(p) = p_i^{q'}, \text{ pour tout } i \in \mathcal{N}_{\bar{d}} \text{ et } p \in P \text{ où } q' \text{ est un entier positif} \\ &p_i^q = F'_i(p_i^{q-1}; p) \quad q = 1, ..., q' \text{ et } p_i^0 = p_i, \end{aligned} \tag{1.37}$$

est une sous-application m-continue sur P'.

Preuve : Nous allons montrer que $\check{F}$ est une sous-application minorée par la sous-application de gradient F' continue pour l'ordre sur P'. Nous raisonnons par récurrence. Pour $q' = 1$, l'application $\check{F}$ n'est autre que l'application de gradient et la preuve de la Proposition 1.5.2 donne le premier pas de la récurrence. Supposons à présent qu'il existe q, $1 \leq q < q'$, tel que pour tout p sous-solution de $\underline{F}$,

$$p_i^0 \leq \cdots \leq p_i^{q-1} \leq p_i^q \leq \underline{F}_i(p) \text{ et } p_i^q \neq p_i \text{ si } \underline{F}_i(p) \neq p_i.$$

Alors, en raison de la monotonie de F' sur P', on déduit de (1.22) que

$$\begin{aligned} p_i^{q+1} &= F_i'(p_i^q; p) \geq F_i'(p_i^{q-1}; p) = p_i^q \quad \text{ et} \\ p_i^{q+1} &= F_i'(p_i^q; p) \leq F_i'(\underline{F}_i(p); p) = \underline{F}_i(p). \end{aligned}$$

Enfin, $p_i^{q+1} \neq p_i$ si $\underline{F}_i(p) \neq p_i$ car $p_i^{q+1} \geq p_i^q \geq p_i$ et $p_i^q \neq p_i$ si $\underline{F}_i(p) \neq p_i$.

□

Remarquons que l'entier q' donné, détermine le nombre de pas de gradient qui est effectué au cours d'une recherche unidirectionnelle. Nous pouvons mettre en évidence deux méthodes de type gradient qui diffèrent dans la valeur du nombre de pas :

– q' **est fixé à 1**. On voit aisément que l'application $\check{F}$ n'est autre que l'application de gradient F'. En utilisant cette opérateur de relaxation suivant le schéma Jacobi, nous obtenons la *méthode de gradient* décrite dans [El Baz 96a].

– q' **est le plus petit entier positif tel que** $|g_i(p_i^{q'}; p)| \leq \epsilon'$, avec ϵ' constante positive donnée. L'application correspondante est encore une sous-application m-continue sur P' (cf. [El Baz et al. 96b]). Dans ce cas, la recherche unidirectionnelle s'achève lorsque le déficit atteint, en valeur absolue, la précision ϵ'. Le nombre de pas q' n'est pas connu à priori (non fixé par l'utilisateur) : il dépend de la précision unidirectionnelle ϵ', du point où s'effectue la recherche suivant la direction donnée et du paramètre de gradient $1/\alpha$. Conduite suivant le schéma Gauss-Seidel, cette méthode est appelée par la suite *méthode de type gradient*.

1.5.2 Formulation des IACF

Nous présentons maintenant une nouvelle classe de méthodes itératives asynchrones : les itérations asynchrones avec communication flexible. Cette nouvelle classe a été tout d'abord proposée dans [Miellou et al. 94] et est définie de la manière suivante :

Définition 1.5.7 *Soit T_i le sous-ensemble infini des itérations auxquelles la i-ème composante du vecteur p est réactualisée, tel que $T_i \cap T_j = \emptyset$, pour tout $i, j \in \mathcal{N}_{\bar{d}}, j \neq i$. Soit $\{\rho_i(k)\}$ la suite de numéros d'itération retardés satisfaisant pour tout $i \in \mathcal{N}_{\bar{d}}$, $0 \leq \rho_i(k) \leq k-1$ avec $\rho_i(k) = k-1$ si $k \in T_i$ et $\lim_{k\to\infty} \rho_i(k) = +\infty$. Une itération asynchrone avec communication flexible associée à une sous-application $\check{F}$ m-continue sur P' et au point initial*

$p^0 \in P'$ sous-solution de l'application de relaxation minimale $\underline{F}$ est une suite $\{p^k\}$ de vecteurs de P telle que pour tout $i \in \mathcal{N}_{\bar{d}}$:

$$\begin{cases} p_i^k = \check{F}_i(\tilde{p}^k), & si\ k \in T_i, \\ p_i^k = p_i^{k-1}, & si\ k \notin T_i, \end{cases}$$

où $\tilde{p}^k$ est implicitement défini par

$$\tilde{p}^k \in \langle \max\{p^{\rho(k)}, \tilde{p}^q\}, p^{k-1} \rangle,$$

$\langle , \rangle$ étant un segment d'ordre dans P, le vecteur $p^{\rho(k)}$ ayant pour composantes $p_i^{\rho_i(k)}, i \in \mathcal{N}_{\bar{d}}$ et $q = \max\{l \in T_i | l < k\}$ (on pose $q = 0$ et $\tilde{p}^0 = p^0$, si $\{l \in T_i | l < k\} = \emptyset$).

Définition 1.5.8 *On définit de manière analogue les méthodes itératives asynchrones avec communication flexible associées à une surapplication $\hat{F}$ $\mathcal{M}$-continue sur P'' et au point initial $p^0 \in P''$ sursolution de l'application de relaxation maximale $\overline{F}$.*

Afin de simplifier la présentation nous avons supposé que $T_i \cap T_j = \emptyset$ pour tout $i, j \in \mathcal{N}_{\bar{d}}, j \neq i$, cette hypothèse ne diminue en rien la portée de cette étude (cf. [El Baz et al. 96b]).

Les itérations asynchrones avec communication flexible sont des méthodes générales pour lesquelles les itérations sont conduites en parallèle sans ordre ni synchronisation sur un nombre de processeurs pouvant aller jusqu'à $(n-1)$. L'originalité de cette nouvelle classe de méthodes provient des communications flexibles entre processeurs. En effet dans ce nouveau modèle $\tilde{p}_i^k$ n'est pas issu nécessairement d'une nouvelle réactualisation de p_i, il peut aussi correspondre à la valeur courante du prix p_i produite par quelques pas du processus construisant la sous-application $\check{F}$. Ainsi, $\tilde{p}_i^k$ peut provenir d'une réactualisation ou d'une réactualisation partielle. Il en résulte une meilleure intéraction entre communication et calcul. Ceci se traduit dans le modèle par le fait que le vecteur $\tilde{p}^k$ utilisé lors de la réactualisation de p_i^k est inclus dans un segment d'ordre borné par en dessus par p^{k-1}.

1.5.3 Convergence des IACF

Les itérations asynchrones avec communication flexible sont fortement liées aux concepts de sous-application et de surapplication qui permettent de générer une séquence monotone de vecteurs.

La proposition suivante établit la convergence monotone des itérations asynchrones avec communication flexible.

Proposition 1.5.4 *Si $\check{F}$ est une sous-application m-continue sur P', $p^0 \in P'$ un point de départ sous-solution de $\underline{F}$ et $\underline{p}$ la solution optimale minimale du problème, alors la suite $\{p^k\}$ des IACF associées à $\check{F}$ et p^0 est bien définie et satisfait $p^k \uparrow \underline{p}, k \to \infty$.*

De manière analogue, les IACF associées à une surapplication $\mathcal{M}$-continue $\hat{F}$ sur P'' convergent vers la solution optimale maximale $\overline{p}$ du problème à partir d'un point p^0 sursolution de $\overline{F}$.
Preuve : cf. [El Baz et al. 96b].

Conclusion

Dans un premier temps nous avons formulé le problème de flot de coût minimum à critère convexe dans un réseau $\mathcal{G} = (\mathcal{N}, \mathcal{A})$. Il s'agit d'un problème d'optimisation avec contraintes linéaires et à critère séparable convexe : les variables primales sont les composantes $x_{ij}, (i,j) \in \mathcal{A}$ du vecteur flot x et le critère est la sommation des fonctions de coût $c_{ij}(x_{ij})$ associées à chaque arc du graphe.

La théorie de la dualité et de la convexité nous a permis d'écrire le dual du problème en faisant apparaître dans le critère dual $q(p)$, les fonctions convexes conjuguées $c^*_{ij}(p_i - p_j)$ de c_{ij} fonction des variables duales p_i. Ces variables associées à chaque sommet du graphe sont les composantes du vecteur prix ou potentiel $p \in \mathbb{R}^n$ (n étant le nombre de sommets du graphe $\mathcal{G}$). Sous certaines hypothèses sur les fonctions de coût c_{ij}, il s'avère que le problème dual est sans contrainte et différentiable. Sa résolution peut être entreprise par une méthode de descente.

En fixant l'une des composantes du vecteur p à 0, il est possible de travailler sur un problème dual réduit dont l'ensemble des solutions duales optimales est compact (alors qu'il est non borné sur $\mathbb{R}^n$). La structure de cet ensemble de solutions facilite la construction d'opérateurs de point fixe (relaxation) qui ont de bonnes propriétés (monotonie et continuité) pour la mise en œuvre d'itérations asynchrones. La méthode de relaxation (séquentielle) consiste à sélectionner de manière cyclique chaque axe de l'espace dual réduit ($\mathbb{R}^{n-1}$) et à minimiser le critère $q(p)$ suivant cette direction.

Dans les méthodes classiques, les composantes du vecteur sont évaluées dans un ordre prédéfini. Cet ordre est naturellement préservé sur une architecture séquentielle mais il nécessite le recours de synchronisations sur une architecture parallèle. Pour éviter de pénaliser les processeurs qui attendent sur les synchronisations, on envisage une résolution par méthode itérative parallèle asynchrone. La mise en œuvre des méthodes asynchrones fait disparaître cette notion d'ordre puisque chaque processeur réévalue à son propre rythme les composantes qui lui sont affectées. Le modèle des itérations asynchrones tient compte des retards dans l'accès aux itérés précédemment calculés. La convergence a été prouvée dans le cadre des itérations asynchrones classiques par Bertsekas et El Baz concernant les methodes de relaxation (cf. [Bertsekas et El Baz 87]) et de gradient asynchrones (cf. [El Baz 96a]).

Le modèle standard des itérations asynchrones permet l'échange de données qui correspondent uniquement au résultat des réactualisations. Il peut être intéressant de communiquer également des itérés partiels correspondant à la valeur courante du vecteur itéré prise à certains instants, ceci afin d'accélérer la convergence. Nous avons appliqué ces nouveaux schémas au problème de flot non linéaire et montré leur convergence dans [El Baz et al. 96b].

Le chapitre suivant est consacré à la mise en œuvre des algorithmes synchrones, asynchrones et asynchrones avec communication flexible sur plusieurs architectures parallèles.

Deuxième partie

Optimisation linéaire dans les graphes : résolution du problème de Flot Maximum

Conclusion générale

Les travaux présentés dans ce mémoire sont à la frontière entre les Mathématiques Appliquées et l'Informatique. De manière plus précise, nous nous sommes intéressés à la résolution de divers problèmes de flot dans les réseaux à l'aide des techniques du parallélisme. Notre contribution a porté plus exactement sur la mise en œuvre de méthodes parallèles sur diverses architectures de type mémoire distribuée (TNode à 16 processeurs) et mémoire partagée (Multiprocesseur symétrique ou SMP à 4 processeurs) et sur l'analyse de leurs performances.

Dans un premier temps, nous avons considéré des problèmes de flot de coût minimum à critère convexe et nous avons proposé diverses mises en œuvre sur TNode et SMP d'algorithmes itératifs parallèles synchrones ou asynchrones. Plus précisément, nous avons étudié des méthodes de relaxation ou de relaxation approchée, des méthodes de gradient, et des méthodes basées sur des sous-applications. Ces dernières se sont avérées être les plus rapides dans nos expérimentations. Ces applications sont essentiellement régulières et présentent des activités parallèles à gros grain. Les résultats expérimentaux ont montré que, pour ce type d'application, les versions asynchrones de ces méthodes sont plus performantes que les versions synchrones. Les gains d'efficacité sont particulièrement importants lorsque les charges de calcul sont déséquilibrées de manière indéterministe comme c'est le cas pour les méthodes basées sur des sous-applications. Dans ce cas, le nombre de pas des activités de recherche d'un minimum suivant une direction donnée est variable suivant la proximité de ce minimum.

Les résultats numériques ont mis en évidence les très bonnes performances des schémas itératifs asynchrones avec communication flexible pour lesquels des itérés partiels peuvent être échangés entre processeurs : les efficacités sont proches de 90%. Notons de plus que le concept de communication flexible est très naturel dans le cas des architectures SMP qui sont des machines associées à un modèle asynchrone (processus interagissant avec une mémoire commune). En effet, la lecture de la valeur courante de chaque composante du vecteur itéré rangé dans la mémoire commune correspond à l'accès à un itéré qui peut être partiel et donc, non référencé par un numéro d'itération particulier.

Nous avons également traité des problèmes de flot maximum/coupe minimale. Nous avons considéré quelques-uns des différents algorithmes appartenant aux deux grandes classes de méthodes de résolution : les méthodes de type chaîne augmentante (ou améliorante) et les méthodes basées sur la notion de préflot. Cette application présente essentiellement un aspect irrégulier et les activités de calcul sont à grain fin.

Dans cette partie, nous nous sommes attachés à l'étude des performances des différentes méthodes. Les expérimentations numériques ont mis en évidence les bonnes performances des méthodes de préflot et l'importance des heuristiques permettant de détecter efficacement une coupe minimale. Nous avons enfin proposé des éléments pour la parallélisation d'une méthode préflot sur machine SMP faisant usage de librairies de threads. Les threads sont en effet bien adaptés à la granularité fine et permettent de maintenir un bon taux d'activité des processeurs.

Les perspectives de recherche de ce travail sont multiples. En ce qui concerne les problèmes de flot de coût minimum à critère convexe, de nombreux points restent en suspens notamment en ce qui concerne la mise en œuvre de méthodes asynchrones avec communication flexible. L'étude des performances de ces schémas itératifs sur divers types d'architectures tels que supercalculateur (Cray T3E, Fujitsu VPP300) et réseau de stations, est actuellement en cours au laboratoire. Notons aussi que d'autres applications pour les itérations asynchrones avec communication flexible sont aussi considérées. Citons notamment les problèmes de commande optimale et la résolution de systèmes linéaires.

Enfin, pour ce qui est des problèmes de flot maximum/coupe minimale, il serait intéressant de mettre en œuvre l'approche que nous avons proposée et d'étudier ses performances sur un multiprocesseur symétrique.

Bibliographie

[3L Ltd. 88] 3L Ltd. *Parallel C User Guide*, 1988.

[Ahuja et al. 93] R. K. Ahuja, T. L. Magnanti and J. B. Orlin. Network flows : theory, algorithms and applications. Prentice Hall, 1993.

[Ahuja et Orlin 91] R. K. Ahuja and J. B. Orlin. *Distance-directed augmenting path algorithms for maximum flow and parametric maximum flow problems.* volume 38 of *Naval Research Logistics Quaterly*, pages 413–430. 1991.

[Alizadeh et Goldberg 93] F. Alizadeh and A. V. Goldberg. *Implementing the Push-Relabel method for the maximum flow problem on a Connection Machine.* In The First DIMACS Implementation Challenge (refereed proceedings), pages 65–96. AMS, September 1993.

[Authié 87] G. Authié. *Contribution à l'optimisation de flots dans les réseaux. Un multiprocesseur expérimental pour l'étude des itérations asynchrones.* Thèse d'état, Université Paul Sabatier, Toulouse, Septembre 1987.

[Badics et Boros 92] T. Badics and E. Boros. *Implementing a maximum flow algorithm : experiments with dynamic trees.* Rapport technique, DIMACS, 1992.

[Bahi et Miellou 93] M. Bahi and J. C. Miellou. *Contractive mappings with maximum norms : comparison of constants of contraction and application to asynchronous iterations.* Parallel Computing, vol. 19, pages 511–523, 1993.

[Baudet 78] G. M. Baudet. *Asynchronous iterative methods for multiprocessors.* Journal Assoc. Comput. Mach., vol. 2, pages 226–244, 1978.

[Bertsekas et al. 87] D. P. Bertsekas, P. Hossein and P. Tseng. *Relaxation methods for network flow problems with convex arc costs.* SIAM Journal on Control and Optimization, vol. 25, pages 1219–1243, 1987.

[Bertsekas et al. 94] D. P. Bertsekas, D. A. Castañon, J. Eckstein and S. A. Zenios. *Parallel computing in network optimization.* report lids-p-2236, Department of Electrical Engineering and Computer Science, M.I.T., Cambridge, Mass., 1994.

[Bertsekas et El Baz 87] D. P. Bertsekas and D. El Baz. *Distributed asynchronous relaxation methods for convex network flow problems.* SIAM Journal on Control and Optimization, vol. 25, no. 1, pages 74–85, 1987.

[Bertsekas et Tsitsiklis 89] D. P. Bertsekas and J. N. Tsitsiklis. Parallel and distributed computation, numerical methods. Prentice Hall, Englewood Cliffs, N.J., 1989.

[Bertsekas 82a] D. P. Bertsekas. Constrained optimization and lagrange multiplier methods. Academic Press, New York, 1982.

[Bertsekas 82b] D. P. Bertsekas. *Distributed dynamic programming.* IEEE Transactions on Automatic Control, vol. AC-27, pages 610–616, 1982.

[Bertsekas 83] D. P. Bertsekas. *Distributed asynchronous computation of fixed points.* Mathematical Programming, vol. 27, pages 107–120, 1983.

[Bertsekas 95a] D. P. Bertsekas. *An auction algorithm for the max-flow problem.* Journal of Optimization Theory and Applications, vol. 87, no. 1, pages 69–101, October 1995.

[Bertsekas 95b] D. P. Bertsekas. Nonlinear programming. Athena Scientific, Belmont, Mass., 1995.

[Birkhoff et Diaz 65] G. Birkhoff and J. B. Diaz. *Nonlinear network problems.* Quart. Appl. Math., vol. 13, pages 431–443, 1965.

[Chajakis et Zenios 91] E. Chajakis and S. A. Zenios. *Synchronous and asynchronous implementations of relaxation algorithms for nonlinear network optimization.* Parallel Computing, vol. 17, pages 873–894, 1991.

[Chazan et Miranker 69] D. Chazan and W. Miranker. *Chaotic relaxation.* Linear Algebra Appl., vol. 2, pages 199–222, 1969.

[Cheriyan et al. 96] J. Cheriyan, T. Hagerup and K. Mehlhorn. *An $O(n^3)$-time maximum-flow algorithm.* SIAM Journal on Computing, vol. 25, no. 6, pages 1144–1170, December 1996.

[Cheriyan et Maheshwari 89] J. Cheriyan and S. N. Maheshwari. *Analysis of preflow push algorithms for maximum network flow.* SIAM Journal on Computing, vol. 18, no. 6, pages 1057–1086, December 1989.

[Cherkassky et Goldberg 97] B. V. Cherkassky and A. V. Goldberg. *On implementing Push-Relabel method for the maximum flow problem.* Algorithmica, vol. 19, pages 390–410, 1997.

[Derigs et Meier 89] U. Derigs and W. Meier. *Implementing Goldberg's maxflow algorithm : a computational investigation.* ZOR–Methods and models of operations research, vol. 33, pages 383–403, 1989.

[Dinic 70] E. A. Dinic. *Algorithm for solution of a problem of maximum flow in networks with power estimation.* Soviet Mathematics Doklady, vol. 11, pages 1277–1280, 1970.

[Edmonds et Karp 72] J. Edmonds and R. M. Karp. *Theoretical improvements in algorithmic efficiency for network flow problems.* Journal of the ACM, vol. 19, pages 248–264, 1972.

[El Baz et al. 96a] D. El Baz, D. Gazen, J. C. Miellou and P. Spiteri. *Mise en œuvre de méthodes itératives asynchrones avec communication flexible.* Calculateurs Parallèles, vol. 8, no. 4, pages 393–410, 1996.

[El Baz et al. 96b] D. El Baz, P. Spiteri, J. C. Miellou and D. Gazen. *Asynchronous iterative algorithms with flexible communication for nonlinear network flow problems.* J. Parallel Distrib. Comput., vol. 38, 1996.

[El Baz 90] D. El Baz. *M-functions and parallel asynchronous algorithms.* SIAM Journal on Numerical Analysis, vol. 27, no. 1, pages 136–140, February 1990.

[El Baz 93] D. El Baz. *Asynchronous implementation of relaxation and gradient algorithm for convex network flow problems.* Parallel Computing, vol. 19, pages 1019–1028, 1993.

[El Baz 96a] D. El Baz. *Asynchronous gradient algorithms for a class of convex separable network flow problems.* Computational Optimization and Applications, vol. 5, pages 187–205, 1996.

[El Baz 96b] D. El Baz. *An efficient termination protocol for asynchronous iterative algorithms.* In Parallel and distributed computing systems, volume 1 of *ISCA*, pages 1–6, Dijon, France, September 25–27 1996.

[El Tarazi 82] M. N. El Tarazi. *Some convergence results for asynchronous algorithms.* Numerisch Mathematik, vol. 39, pages 325–340, 1982.

[Ford et Fulkerson 56] Jr. Ford L. R. and D. R. Fulkerson. *Maximal flow through a network.* Canadian Journal of Mathematics, vol. 8, pages 339–404, 1956.

[Frommer et Schwandt 95] A. Frommer and H. Schwandt. *Asynchronous parallel methods for enclosing solutions of nonlinear equations.* J. Comput. Appl. Math., vol. 60, pages 47–62, 1995.

[Frommer 91] A. Frommer. *On asynchronous iterations in partially ordered spaces.* Numer. Funct. Anal. and Optimiz., vol. 12, pages 315–325, 1991.

[Gazen et El Baz 95] D. Gazen and D. El Baz. *Efficient implementation of parallel algorithms for non linear network problems.* In B. Hertzberger and G. Serazzi, editeurs, High Performance Computing and Networking, volume 919 of *Lecture Notes in Computer Science*, pages 945–946, 1995.

[Giraud et Spiteri 91] L. Giraud and P. Spiteri. *Résolution parallèle de problèmes aux limites non linéaires.* RAIRO Modél. Math. Anal. Numér., vol. 25, pages 73–100, 1991.

[Goldberg et Tarjan 88] A. V. Goldberg and R. E. Tarjan. *A new approach to the maximum flow problem.* Journal of the ACM, vol. 35, pages 921–940, 1988.

[Goldberg 85] A. V. Goldberg. *A new maximum flow algorithm.* Rapport technique, MIT/LCS/TM-291, Laboratory for Computer Science, Mass. Inst. of Technology, Cambridge, MA, 1985.

[Goldfarb et Grigoriadis 88] D. Goldfarb and M. D. Grigoriadis. *A computational comparison of the Dinic and network simplex methods for maximum flow.* Annals of Oper. Res., vol. 13, pages 83–123, 1988.

[Goldschlager et al. 82] L. M. Goldschlager, R. A. Shaw and J. Staples. *The maximum flow problem is Log Space Complete for P.* Theoretical Computer Sci., vol. 21, pages 105–111, 1982.

[Hoare 85] C.A.R. Hoare. Communicating sequential processes. Prentice Hall, Englewood Cliffs, N.J., 1985.

[INMOS 87] Ltd. INMOS. *Occam 2 reference manual*, prentice hall edition, 1987.

[Karypis et Kumar 96] G. Karypis and V. Kumar. *Fast and high quality techniques for partitioning irregular graphs.* In Parallel Optimization Colloquium, pages 119–133. PRiSM-Versailles & CRT-Montréal, March 1996.

[Karzanov 74] A. V. Karzanov. *Determining the maximal flow in a network with the method of preflows.* Soviet Mathematics Doklady, vol. 15, pages 434–437, 1974.

[Kleiman et al. 96] S. Kleiman, D. Shah and B. Smaalders. Programming with threads. Prentice Hall, 1996.

[Lewis et Berg 96] B. Lewis and D. J. Berg. Threads primer : a guide to multithreaded programming. Prentice Hall, 1996.

[Luenberger 73] D. G. Luenberger. Introduction to linear and nonlinear programming. Addison-Wesley, 1973.

[Miellou et al. 90] J. C. Miellou, Ph. Cortey-Dumond and M. Boulbrachêne. *Perturbation of fixed-point iterative methods.* Advances in Parallel Computing, vol. 1, pages 81–122, 1990.

[Miellou et al. 94] J. C. Miellou, D. El Baz and P. Spiteri. *A new class of asynchronous iterative algorithms with order intervals.* Rapport technique 1994-16, Rapport LCS, 1994. to appear in *Mathematics of Computation* on January 1998.

[Miellou et Spiteri 85a] J. C. Miellou and P. Spiteri. *Two criteria for the convergence of asynchronous iterations.* In Computers and Computing, pages 91–95. Wiley-Masson,, Paris, 1985.

[Miellou et Spiteri 85b] J. C. Miellou and P. Spiteri. *Un critère de convergence pour des méthodes générales de point fixe.* Modél. Math. Anal. Numér., vol. 19, pages 645–669, 1985.

[Miellou 74] J. C. Miellou. *Itérations chaotiques à retards.* C.R. Acad. Sci., vol. 278, pages 957–960, 1974.

[Miellou 75a] J. C. Miellou. *Algorithmes de relaxation chaotique à retards.* RAIRO, vol. R1, pages 55–82, 1975.

[Miellou 75b] J. C. Miellou. *Itérations chaotiques à retards, étude de la convergence dans le cas d'espaces partiellement ordonnés.* In C.R.A.S., volume 280, pages 233–236, Paris, 1975.

[Miellou 86] J. C. Miellou. *Asynchronous iterations and order intervals.* In Parallel algorithms and architectures, pages 85–96. M. Cosnard Ed., North-Holland, Amsterdam, 1986.

[Ostrowski 61] A. M. Ostrowski. *Iterative solution of linear systems of functional equations.* Journal of Mathematical Analysis and Applications, vol. 2, pages 351–369, 1961.

[Pellegrini 95] F. Pellegrini. *SCOTCH 3.0 User'sGuide.* LaBRI, Université Bordeaux I, October 1995.

[Porshing 69] A. Porshing. *Jacobi and Gauss-Seidel methods for nonlinear network problems.* SIAM Journal on Numerical Analysis, vol. 6, pages 437–449, 1969.

[Powell 73] M. J. D. Powell. *On search directions for minimization algorithms.* Mathematical Programming, vol. 4, pages 193–201, 1973.

[Rheinboldt 70] W. C. Rheinboldt. *On M-functions and their application to nonlinear Gauss-Seidel iterations and to network flows.* Journal of Mathematical Analysis and Applications, vol. 32, pages 274–307, 1970.

[Roch 93] J. L. Roch. Ecole d'automne capa (conception et analyse d'algorithmes parallèles), chapitre Complexité parallèle et algorithmique PRAM, pages 93–114. Port d'Albret (Landes), Septembre 1993.

[Rockafellar 70] R. T. Rockafellar. Convex analysis. Princeton University Press, Princeton New Jersey, 1970.

[Rockafellar 84] R. T. Rockafellar. Network flows and monotropic optimization. John Wiley & Sons, New York, 1984.

[Rosenfeld 69] J. L. Rosenfeld. *A case study in programming for parallel processors.* Communications of the ACM, vol. 12, pages 645–655, 1969.

[Shiloach et Vishkin 82] Y. Shiloach and U. Vishkin. *An $O(n^2 \log n)$ parallel Max-Flow algorithm.* Journal of algorithms, vol. 3, pages 128–146, 1982.

[Spiteri et al. 95] P. Spiteri, J. C. Miellou and D. El Baz. *Asynchronous alternating Schwarz method for the solution of nonlinear partial differential equations.* Rapport technique 95-17-R, Rapport IRIT, 1995.

[Sun Solaris 93] Sun Solaris. *AnswerBook Solaris 2.4 : Multithreaded programming Guide, Threads Solaris.*, 1993.

[Sun Solaris 94] Sun Solaris. *AnswerBook Solaris 2.5 : Multithreaded programming Guide, Threads POSIX (/Solaris)*, 1994.

[Tseng et Bertsekas 87] P. Tseng and D. P. Bertsekas. *Relaxation methods for problems with strictly convex separable costs and linear constraints.* Math. Programming, vol. 38, pages 303–321, 1987.

[Tseng 90] P. Tseng. *Dual coordinate ascent for problem with strictly convex cost and linear constraints, a unified approach.* SIAM Journal on Control and Optimization, vol. 28, pages 214–242, 1990.

[Tsitsiklis et Bertsekas 86] J. N. Tsitsiklis and D. P. Bertsekas. *Distributed asynchronous optimal routing in data networks.* IEEE Trans. Auto. Contr., vol. AC-31, pages 325–332, 1986.

[Waissi] G. Waissi. Program ac-max.c for creating maximum flow problem instances. Available through DIMACS.

[Zenios et Lasken 88] S. A. Zenios and R. Lasken. *The Connection Machines CM-1 and CM-2: solving nonlinear network problems.* pages 648–658, St Malo, France, 1988. International Conference on Supercomputing.

[Zenios et Mulvey 88] S. A. Zenios and J. Mulvey. *A distributed algorithm for convex network optimization problems.* Parallel Computing, vol. 6, pages 45–56, 1988.