REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOHAMED BOUDIAF - M'SILA
FACULTE DES MATHEMATIQUES ET DE L'INFORMATIQUE
DEPARTEMENT D'INFORMATIQUE

MEMOIRE de fin d'étude

Présenté pour l'obtention du diplôme de MASTER

Domaine : Mathématiques et Informatique

Filière : Informatique

Spécialité : Informatique Décisionnel et Optimisation

Par : BELOUADAH Allaoua Taieb

SUJET

Accélération de la résolution des systèmes d'équations non linéaires par décomposition et parallélisation.

Soutenu publiquement le : / /2019 devant le jury composé de :

OULDMOHAMEDI Najib	**Université de M'sila**	**Président**
MOUSSAOUI Adel	**Université de M'sila**	**Rapporteur**
BARKAT Abdelbasset	**Université de M'sila**	**Examinateur**

Promotion : 2018 /2019

Remerciements

Je remercie Dieu le tout puissant et miséricordieux, qui M'a donné la force et la patience d'accomplir ce modeste travail.

Mes vifs remerciements vont à mes parents et ma famille pour leurs sacrifices, leurs encouragements.

Je tiens à remercier mon encadreur Mr : MOUSSAOUI Adel pour son précieux conseil et son aide durant toute la période de travail.

Mes vifs remerciements vont également aux membres du jury pour l'intérêt qu'ils ont porté à ma recherche en acceptant d'examiner mon travail et de l'enrichir par leurs propositions.

Enfin, je tiens également à remercier tous mes amis et toutes les personnes qui ont participé de près ou de loin à la réalisation de ce travail.

Chapitre 1 : Modélisation d'un problème par un système d'équation

Chapitre 2 : Accélération par décomposition

Chapitre 3 : Accélération par parallélisation

Chapitre 4 : Réalisation, Teste et Résultat

List des figures

List des tableaux

INTRODUCTION GÉNÉRALE

Galilée a déclaré que « les mathématiques sont le langage de la nature ». Il voulait dire que les phénomènes naturels peuvent se modéliser par des équations et des systèmes d'équations. Mais la plupart des phénomènes sont des systèmes complexe c.-à-d. ils sont composés d'un grand nombre de parties composés, donc c'est très difficile à étudier, traitée et résoudre ces problèmes dans cette forme.

Les mathématiques allaient fournir une gamme d'outils pour décrire une multitude de phénomènes différents et les représenter sous forme d'un modèle mathématique. Un modèle mathématique est un ensemble d'équations décrivant le mieux possible le phénomène qu'il représente.

Pour faciliter et simplifier la résolution des problèmes dans tous les domaines soit dans la physique, mécanique, synthèse d'image, informatique, finance ou bien dans la biologie... etc., généralement, nous les modélisons par des systèmes d'équations.

Un système d'équation peut être modélisé à partir de l'ensemble de relations et contraintes que vérifient les différentes parties d'un problème. C.-à-d. un systéme d'équation est la composition de n équations, et chaque équation représente les relations avec ces contraintes. Donnons quelques exemples de contraintes : la température, la vitesse, incidence entre un point et une droite (un plan ou toute courbe ou surface), tangences entre cercles ...

Pour résoudre les systèmes d'équations, plusieurs méthodes existent pour les systèmes d'équations linéaires, parmi les méthode les plus connues (Méthode de Newton-Raphson, La méthode de Broyden, Méthode de dichotomie (ou de la bissection)) mais contrairement à ce qui a été le cas pour les systèmes d'équations linéaires, la résolution des systèmes non-linéaires s'avère beaucoup plus délicate. En général il n'est pas possible de garantir la convergence vers la solution correcte et la complexité des calculs croit très vite en fonction de la dimension du système.

Il n'existe pas une méthode universelle de la résolution des systèmes des équations non linéaire, il-y-a plusieurs méthodes pour résoudre ces systèmes. Mais la résolution du système d'équation non linéaire avec les méthodes générales dans les grands systèmes sont coûteuses en temps de calcul et en espace.

Donc la question est : est-ce-que nous pouvons trouver une méthode efficace pour accélérer la résolution du système et minimiser le temps de la résolution ?

Dans ce mémoire, nous essayons de répondre à cette question via les quatre chapitres suivants:

Le premier chapitre est la modélisation d'un problème par un système d'équations. Dans ce chapitre nous allons discuter généralement les systèmes d'équations non linéaires. Nous allons présenter aussi la modélisation de quelques problèmes réels par un système d'équations non linéaires afin de les résoudre. Donc nous allons faire une présentation et une explication simple, en commençant par la définition d'un système non linéaire et un système des équations non linéaire, et la représentation de ces systèmes, puis nous allons expliquer comment nous pouvons modéliser les problèmes sous forme d'un modèle mathématique pour simplifier l'étude et faciliter la résolution de ces problèmes. Nous allons donner aussi quelques exemples sur les méthodes de résolutions des systèmes d'équations non linéaires les plus utilisés. Finalement, Nous allons montrer l'importance de la modélisation par les systèmes non linéaires via des exemples.

Le deuxième chapitre présente une méthode d'accélération du processus de résolution par décomposition. Dans ce chapitre nous allons considérer le graphe biparti associé à un système d'équations. Nous allons expliquer comment le décomposer en sous-systèmes bien contraints, sur-contraints et sous-contraints, et nous allons utiliser une méthode efficace de décomposition des systèmes bien contraints en sous-systèmes irréductibles afin d'accélérer le processus de résolution. Cette décomposition accélérer grandement la résolution dans le cas des grands systèmes réductibles, nous rappelons que la complexité de résolution des systèmes d'équations non linéaires est généralement en O (n^3).

Le troisième chapitre présente l'accélération par parallélisation. Dans ce chapitre nous allons présenter une méthode de parallélisation. Nous exploitons les résultats de la méthode de décomposition (présentée dans le chapitre précédent) avec la méthode d'ordonnancement de taches appelée PERT. Plusieurs serveurs vont résoudre le système d'équations en parallèle, selon une architecture client/serveur. Les serveurs communiquent via les Sockets.

Dans le quatrième chapitre, nous allons présenter la réalisation, les tests et les résultats. Dans ce chapitre nous allons récapituler notre travail. Nous allons expliquer notre implémentation en code JAVA, puis on va tester notre code à l'aide d'un exemple explicatif.

CHAPITER 1

MODÉLISATION D'UN PROBLÈME PAR UN SYSTÈME D'ÉQUATION

1. Introduction

Dans ce chapitre nous discutons généralement les systèmes des équations non linéaires, nous présentons aussi la modélisation de quelques problèmes réels par système d'équations non linéaire afin de les résoudre. Donc nous allons faire une présentation et une explication simple, en commençant par la définition d'un système non linéaire et un système des équations non linéaire, et la représentation de ces systèmes, puis ont expliqué comment peut-on modéliser les problèmes sous forme d'un modèle mathématique pour simplifier l'étude et faciliter la résolution des problèmes. Nous donnons aussi quelques exemples sur les méthodes de résolutions des systèmes d'équations non linéaires les plus utilisés. Finalement, Nous montrons l'importance de la modélisation par systèmes non linéaires via des exemples.

2. Systèmes non linéaires

Par définition, un système non linéaire est un système qui n'est pas linéaire, c'est-à-dire (au sens physique) qui ne peut pas être décrit par des équations différentielles linéaires à coefficients constants. Cette définition explique la complexité et la diversité des systèmes non linéaires et des méthodes qui s'y appliquent. Il n'y a pas une théorie générale pour ces systèmes, mais plusieurs méthodes adaptées à certaines classes de systèmes non linéaires. [9]

3. Système d'équation non linéaire [9]

Un système d'équation non linéaire est la composition faite de n équation non linéaire :

$$\begin{array}{c} f_1(X) = \varphi_1(X) \\ f_2(X) = \varphi_2(X) \\ \cdots \qquad \cdots \\ f_n(X) = \varphi_n(X) \end{array}$$

Ou $X = \{x_1, x_2, \dots, x_p\}$ sont les p inconnues du système alors que $\varphi_1, \varphi_2, \dots, \varphi_n$ sont les n fonctions, elles même dépendantes de X. Géométriquement les n équations représentent les n courbes$(\alpha, \beta, \gamma, \dots)$.

4. Représentation des systèmes d'équation non linéaire [9]

La forme générale pour représenter un système d'équation non linéaire est donnée comme suite :

$$\vec{x} \in \mathbb{R}^n \quad tq \quad \vec{f}(\vec{x}) = 0 \qquad \vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \; \begin{matrix} f: \mathbb{R}^n \to \mathbb{R}^n \\ \vec{x} \to \vec{f}(\vec{x}) \end{matrix} \; \vec{f}(\vec{x}) = \begin{cases} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) \end{cases}$$

4.1. Exemple

Pour représenter un système on prend comme exemple le système d'équation non linéaire suivante :

Pour N=8 et X= (x1, x2, x3, x4, x5, x6, x7, x8, x9) tel que :

$$\begin{cases} (x_2 + x_3)^2 - x_5 = 0 \\ (x_1 + x_2)(x_1 + x_2) - x_4 - x_6 = 0 \\ (x_6 - 20) = 0 \\ x_5^2 - x_6 + x_7 = 0 \\ 3x_7 - x_8 = 0 \\ x_7 x_8 - 10 = 0 \\ (x_7 + x_9)(x_7 - x_9) + 10 = 0 \\ x_9 - 20 = 0 \end{cases}$$

Donc la forme de ce système est :

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{pmatrix}, \vec{f}(\vec{x}) = \begin{Bmatrix} f_1(x_2, x_3, x_5) \\ f_2(x_1, x_2, x_4, x_6) \\ f_3(x_6) \\ f_4(x_5, x_6, x_7) \\ f_5(x_7, x_8) \\ f_6(x_7, x_8) \\ f_7(x_7, x_9) \\ f_8(x_9) \end{Bmatrix} = \begin{Bmatrix} (x_2 + x_3)^2 - x_5 = 0 \\ (x_1 + x_2)(x_1 + x_2) - x_4 - x_6 = 0 \\ (x_6 - 20) = 0 \\ x_5^2 - x_6 + x_7 = 0 \\ 3x_7 - x_8 = 0 \\ x_7 x_8 - 10 = 0 \\ (x_7 + x_9)(x_7 - x_9) + 10 = 0 \\ x_9 - 20 = 0 \end{Bmatrix}$$

5. La modélisation des Systèmes non linéaire [14]

Généralement, un problème scientifique qui peut être par exemple mécanique, électrique, biologique ou un phénomène naturel est modélisé mathématiquement afin de le résoudre. La modélisation du phénomène, c'est le traduire sous forme mathématique c'est à dire d'après l'observation du phénomène on extrait les règles et les structures sous-jacentes et le traduire à un ensemble d'équations et de fonctions. Mais avant sa traduction, Il faut d'abord reprendre les variables identifiées et déterminer parmi elles la variable indépendante, la ou les variables dépendantes (dans le cas de plusieurs fonctions) et les simples paramètres. Puis il faut également vérifier que les unités sont homogènes. Pour obtenir les fonctions ou les équations, en suite il faut reprendre les contraintes et surtout traduire l'hypothèse en équations. Cela consiste généralement à reprendre une ou plusieurs lois physiques et les mathématiser en utilisant les outils adéquats, dérivée première, dérivée seconde, dérivée d'une fonction composée, etc. En général, chacun de ces outils se traduit par une série d'équations bien définies.

Donc à partir de ce modèle mathématique, il est possible de dériver un modèle dans l'espace d'état non linéaire en choisissant les variables d'état.

5.1. Exemple

La figure 1.1 donne un exemple de modélisation d'un circuit électrique simple RLC (cas en série) par un système non linéaire. Le circuit est composé d'une résistance R, d'une bobine L, d'une capacité C en série et alimenter par un générateur de tension E.

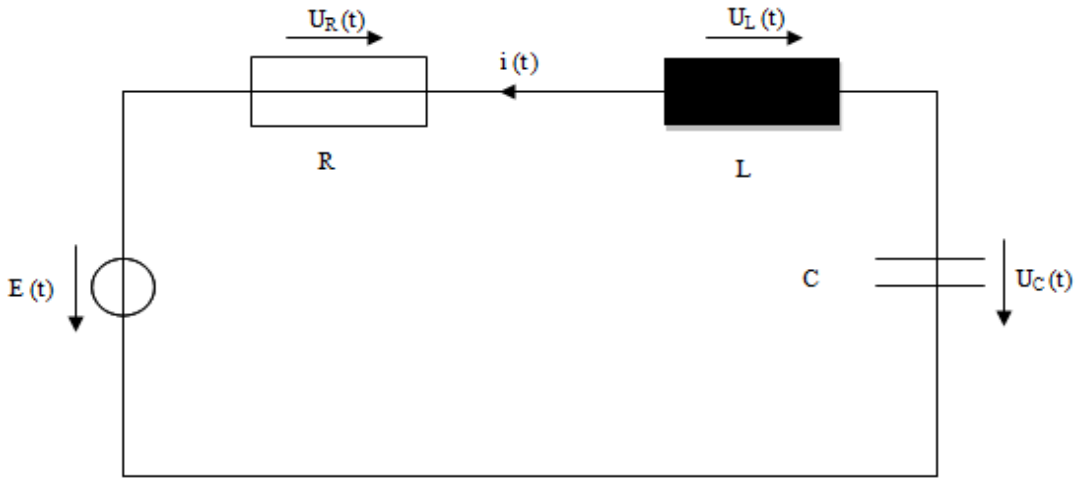


Figure 1.1: Un circuit électrique simple RLC.

Avant de développer l'équation différentielle du second ordre, il est nécessaire d'identifier les expressions des tensions, nous avons :

- Loi d'Ohm: $u_R(t) = Ri(t)$ où $u_{R(t)}$ est la tension aux bornes d'une résistance R.
- Loi $u_C(t) = \frac{q}{C}$ c.-à-d. la différence de potentiel $u_C(t)$ aux bornes d'un condensateur est proportionnelle aux charges qui sont transférées ou en d'autres termes $\frac{du_C(t)}{dt} = \frac{i}{c}$.
- Loi de Coulomb : $i(t) = \frac{dq(t)}{dt}$.
- Loi de Faraday: $u_L(t) = L\frac{di(t)}{dt}$ où $u_L(t)$ est la tension aux bornes d'une inductance L.

Nous avons appliqué la loi des mailles de Kirchhoff (la somme algébrique des tensions) nous obtenons :

$$E(t) = u_R(t) + u_L(t) + u_C(t). \qquad (1)$$

Donc après le développement de l'expression des tensions nous avons obtenu l'équation différentielle suivante :

$$E(t) = Ri(t) + L\frac{di(t)}{dt} + u_C(t) \quad \Leftrightarrow \frac{di}{dt} = -\frac{R}{L}i - \frac{1}{L}u_c(t) + \frac{1}{L}E. \qquad (2)$$

A partir de ce modèle mathématique, il est possible de dériver un modèle dans l'espace d'état non linéaire en choisissant les variables d'état :

$X_1(t) = i(t), x_2(t) = u_C(t), y(t) = x_2(t).$

$$\begin{cases} \dot{x}_1(t) = -\frac{R}{L} x_1(t) - \frac{1}{L} x_2(t) + \frac{1}{L} E(t) \\ \dot{x}_2(t) = \frac{1}{C} x_1(t) \\ y(t) = x_2(t) \end{cases} \quad (3)$$

6. Résolution des systèmes d'équation non linéaire [13]

Contrairement à ce qui a été le cas pour les systèmes d'équations linéaires, la résolution des systèmes non-linéaires s'avère beaucoup plus délicate. En général il n'est pas possible de garantir la convergence vers la solution correcte et la complexité des calculs croit très vite en fonction de la dimension du système.

Pour résoudre les systèmes d'équations, plusieurs méthodes existent, parmi les plus connues nous citons:

- Méthode de Newton-Raphson.
- La méthode de Broyden.
- Méthode de dichotomie (ou de la bissection).

6.1. Méthode de newton-Raphson [9]

- Historique

La méthode de résolution des équations numériques a été initiée par Isaac Newton vers 1669 sur des exemples numériques mais la formulation était fastidieuse. Dix ans plus tard, Joseph Raphson met en évidence une formule de récurrence. Un siècle plus tard, Mouraille et Lagrange étudient la convergence des approximations successives en fonction des conditions initiales par une approche géométrique. Cinquante ans plus tard, Fourier et Cauchy s'occupent de la rapidité de la convergence.

- La méthode

La méthode consiste à introduire une suite (x_n) d'approximation successive de l'équation$f(x) = 0$.

- On part d'un x_0 proche de la solution.

- A partir de x_0, on calcule un nouveau terme $x_1$de la manière suivante : on trace la tangente à $\mathcal{C}_f$ en x_0. Cette tangente coupe l'axe des abscisses en $x_1$comme indiqué sur la figure ci-dessous.
- On réitère ce procédé en calculant $x_2$en remplaçant $x_0$par x_1, puis $x_3$en remplaçant $x_1$par $x_2$et ainsi de suite .

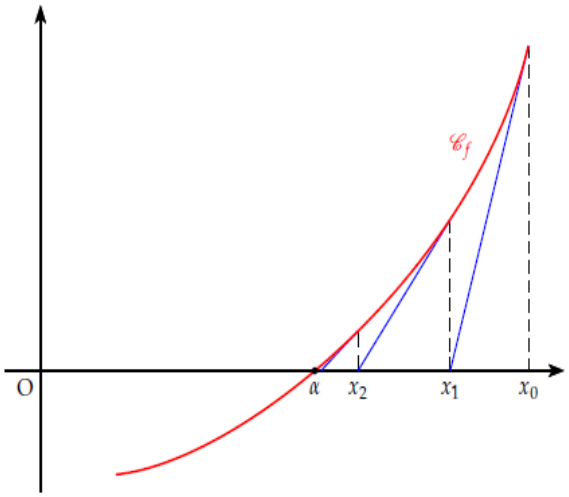


Figure 1.2 : La méthode de newton-Raphson.

- Formule de récurrence

x_{n+1} est l'abscisse du point d'intersection de la tangente à $\mathcal{C}_f$ en $x_n$avec l'axe des abscisses. L'équation de la tangente en $x_n$est : $y = f'(x_n)(x - x_n) + f(x_n)$. (1)

Cette tangente coupe l'axe des abscisses quand y = 0 :

$$f'(x_n)(x - x_n) + f(x_n) = 0 \quad \Leftrightarrow \quad f'(x_n)(x - x_n) = -f(x_n) \qquad (2)$$

$$x - x_n = -\frac{f(x_n)}{f'(x_n)} \quad \Leftrightarrow \quad x = x_n - \frac{f(x_n)}{f'(x_n)}. \qquad (3)$$

Nous avons donc la relation de récurrence suivante: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$. (4)

- Conditions d'application

Pour que la suite (x_n)existe :

- La fonction f doit être dérivable en chacun des points considérés. En pratique la fonction doit être dérivable dans un intervalle centré en α contenant x_0.
- La dérivée ne doit pas s'annuler sur cet intervalle.

Pour que la suite (x_n) soit convergente, les conditions dépassent le cours de terminale, mais en pratique, il faut prendre un x0 assez proche de la valeur a qui annule la fonction. On le détermine à l'aide du théorème des valeurs intermédiaires.

6.2. La méthode de Broyden [9]

- Introduction

Dans le cas des systèmes de grande dimension, la méthode de Newton- Raphson s'avère assez coûteuse puisqu'il faut évaluer à chaque itération $N^2 + 2$ fonctions (les N^2 dérivées partielles de la matrice Jacobienne, plus les N fonctions coordonnées). De plus, il y a un système linéaire $N \times N$ (dont la matrice est en général pleine) à résoudre. Pour pallier le premier inconvénient, il est fréquent qu'on ne recalcule pas la matrice Jacobienne à chaque itération, mais seulement de temps en temps. Bien sûr, à ce moment-là, la convergence n'est plus quadratique, mais en temps de calcul il arrive qu'on y gagne beaucoup, en particulier quand les dérivées de F sont très coûteuses à calculer.

- La méthode

La méthode de Broyden s'inspire directement de l'introduction ci-dessus en poussant même la logique un peu plus loin. Puisqu'il est coûteux (voire même dans certains cas impossible) de calculer la matrice Jacobienne de F, on va se contenter d'en déterminer une valeur approchée à chaque itération B_n. De plus la suite de matrices B_n se calculera simplement par récurrence.

On peut considérer que la méthode de la sécante, en dimension un, est basée un peu sur le même principe. En effet elle revient à approcher :

$$f'(X_n) \text{ par } \frac{f(X_n)-f(X_{n-1})}{X_n-X_{n-1}}. \qquad (1)$$

En dimension N, on va simplement imposer à la suite de matrices B_n de vérifier la même relation :

$$B_n(X_n - X_{n-1}) = F(X_n) - F(X_{n-1})(10). \qquad (2)$$

Bien sûr, la relation ci-dessus ne définit pas la matrice B_n. Elle n'impose que sa valeur dans une direction. Broyden a fait le choix simple de considérer qu'on pouvait passer de B_n à B_{n-1} en rajoutant simplement une matrice de rang 1. Ainsi, sa formule d'actualisation s'écrit :

$$B_{n+1} = B_n + \frac{(\delta F_n - B_n \delta X_n)(\delta X_n)^T}{\|\delta X_n\|^2}. \qquad (2)$$

Où on a noté $\delta X_n = X_{n+1} - X_n$ et $\delta F_n = F(X_{n+1}) - F(X_n)$. On vérifie immédiatement que la suite de matrice B_n définie par l'équation précédant satisfait bien l'équation définie en (1). La méthode de Broyden s'écrit donc :

$$\begin{cases} X_0 \text{ et } B_0 \text{ donnée , pour } n = 0,1,2,\dots, \text{test d'arret , faire} \\ \quad \text{Résolution du systéme linéaire } B_N \delta_N = -F(X_n) \\ \quad X_{n+1} = X_n + \delta_n, \ \delta F_n = F(X_{n+1}) - F(X_n) \\ \quad B_{n+1} = B_n + \dfrac{(\delta F_n - B_n \delta_n)(\delta_n)^T}{\|\delta_n\|^2} \end{cases}$$

Remarque 1 : On peut prendre comme matrice initiale $B_0 = \mathrm{id}$, il faut évidemment attendre un certain temps avant d'avoir une approximation raisonnable de la matrice Jacobienne.

Remarque 2 : On peut démontrer qu'en général, comme pour la méthode de la sécante, la convergence est superlinéaire. La suite de matrices $B_n$ne converge pas forcément vers la matrice Jacobienne de F.

6.3. Méthode de dichotomie (ou de la bissection) [9]

- Description

La méthode de dichotomie ou méthode de la bissection est, en mathématiques, un algorithme de recherche d'un zéro d'une fonction qui consiste à répéter des partages d'un intervalle en deux parties puis à sélectionner le sous-intervalle dans lequel existe un zéro de la fonction.

- La méthode

Soit $f : [a, b] \to R$ une fonction continue. Si $f(a) \cdot f(b) < 0$, la fonction f possède au moins une racine dans]a ,b[. De plus, si on définit par récurrence$[a_0, b_0] = [a, b]$:

$$x_n = \frac{1}{2}(a_n, b_n) \text{ et } [a_{n+1}, b_{n+1}] = \begin{cases} [a_n, x_n] & \text{si } f(a_n)f(x_n) < 0 \\ [x_n, x_n] = \{x_n\} & \text{si } f(x_n) = 0 \\ [x_n, b_n] & \text{si } f(x_n)f(b_n) < 0 \end{cases}$$

Les trois suites(a_n), (b_n) et (x_n) convergent linéairement vers la même limite x^* avec $f(x^*) = 0$.

- Algorithme

La méthode de la bissection est un procédé qui permet à la fois de montrer l'existence d'une racine d'une fonction $f : [a, b] \rightarrow R$ et de l'estimer numériquement. L'idée est simple : si f est continue et change de signe sur [a ,b], il faut que f s'annule en un certain point de [a,b]. Supposons que $f(a) < 0$ et $f(b) > 0$. Choisissons au hasard un point $x_0 \in]a, b[$. Si $f(x_0) = 0$, on a fini. Si $f(x0) < 0$, alors il doit y avoir une racine dans $]x_0, b[=:]a_1, b_1[$. Sinon, $f(x0) > 0$ et il doit y avoir une racine dans $]a, x_0[=:]a_1, b_1[$.

On recommence la procédure en choisissant x_1 dans $[a_1, b_1]$et ainsi de suite ce qui donne une suite décroissante d'intervalles $[a_n, b_n]$contenant chacun une racine. On espère que a_n et $b_n$sont de bonnes approximations d'une racine $x^* : a_n \leq x^* \leq b_n$et $a_n \rightarrow x^*, b_n \rightarrow x^*$.dans ce cas, il faut que la longueur de l'intervalle $[a_n, b_n]$ tende vers 0. Il faut donc choisir les points x_n.

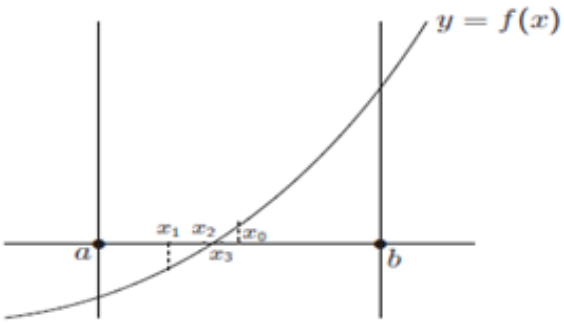


Figure 1.3 : La méthode de bissection.

7. Application

On en connaît de nombreux exemples parmi les systèmes mécaniques ou chimiques : satellites, avions, automobiles, réacteurs chimiques, procédés biotechnologiques ou agro-alimentaires, et électricité moteur à courant continu, circuit RLC, et on biologie l'exemple de proies-prédateurs. Dans la partie suivante on montre l'importance de modélisation au système d'équation via quelque exemple :

7.1. Un exemple en mécanique : Mouvement d'une planète autour du soleil

- Description

La figure 1.4 représente une planète de centre P de masse mP, qui tourne autour du Soleil, de centre S de masse MS, dans le référentiel héliocentrique considéré galiléen, La seule force extérieure qui s'applique sur la planète est la force d'attraction gravitationnelle exercée par le

Soleil sur la planète $\overrightarrow{F_{S/p}}$ d'après la deuxième loi de Newton, on note R la distance entre le centre de la planète et le centre du Soleil en m. On donne ci-dessous le modèle qui explique le mouvement.

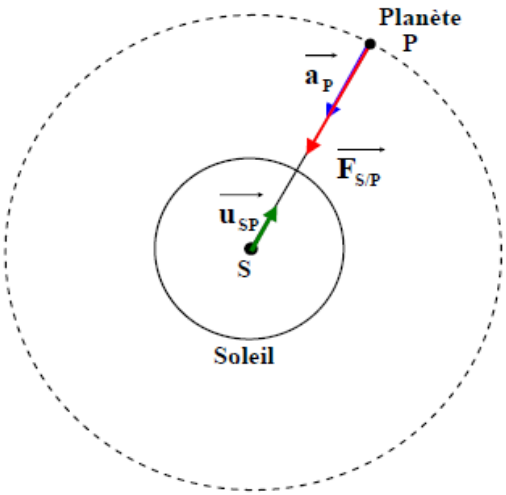


Figure 1.4 : Mouvement d'une planète autour du soleil.

- Modélisation

On a appliqué la 2ème loi de Newton à la planète considérée :

$$\overrightarrow{F_{S/p}} = m_P \cdot \overrightarrow{a_P}. \qquad (1)$$

Ainsi on trouve que l'accélération est radiale (dirigée vers le centre du Soleil) donc :

$$\overrightarrow{a_P} = -G \cdot \frac{M_S}{R^2} \overrightarrow{u_{SP}}. \qquad (2)$$

Alors on a l'équation de la force de soleil sur la planète est :

$$\overrightarrow{F_{S/p}} = -G \cdot \frac{M_S m_P}{R^2} \overrightarrow{u_{SP}}. \qquad (3)$$

Si l'accélération est centripète, c'est-à-dire, le mouvement du planète soit circulaire uniforme on peut écrire que :

$$\overrightarrow{a_P} = \frac{V^2}{R} \qquad (4), \text{ avec V est la vitesse de la planète.}$$

On peut déduire l'équation de la vitesse comme suite :

$$V = \sqrt{G \cdot \frac{M_S}{R}}. \qquad (5)$$

La période de révolution T (en s) d'une planète autour du Soleil est la durée que met la planète pour effectuer un tour complet autour du Soleil à la vitesse V, donc en mouvement circulaire uniforme on déduit que :

$$T = 2\pi \cdot \frac{R}{V} \quad \Rightarrow \quad T = \frac{2\pi \cdot R \cdot \sqrt{R}}{\sqrt{G \cdot M_S}}. \qquad (6)$$

Donc à partir de ce modèle mathématique, il est possible de dériver un modèle dans l'espace d'état non linéaire en choisissant les variables d'état $x_1 = R, x_2 = M_S, x_3 = m_P$ et $f_1 = \overrightarrow{F_{S/p}}, f_2 = V, f_3 = T$, donc :

$$\begin{cases} f_1(x_1, x_2, x_3) = \frac{G \cdot x_2 \cdot x_3}{{x_1}^2} \\ f_2(x_1, x_2) = \sqrt{\frac{G \cdot x_2}{x_1}} \\ f_3(x_1, x_2) = \frac{2\pi \cdot x_1 \cdot \sqrt{x_1}}{G \cdot x_2} \end{cases} \qquad (7)$$

7.2. Un exemple en électricité : Moteur à courant continu [5]

- Description

Un moteur à courant continu (MCC), est un dispositif électromécanique qui convertit une énergie électrique d'entrée en énergie mécanique. L'énergie électrique est apportée par un convertisseur de puissance qui alimente le bobinage disposé sur l'induit mobile (rotor). Ce bobinage est placé dans un champ magnétique. D'un point de vue électrique, le moteur courant continu peut être modélisé comme un système dont l'entrée est la tension de commande de l'induit u(t) et la sortie la vitesse de rotation de l'arbre moteur ω(t).

- Modélisation

Le MCC étant un système électromécanique, les équations dynamiques résultent de la combinaison des modélisations mécanique et électrique du moteur, On donne ci-dessous le modèle de connaissance du moteur courant continu :

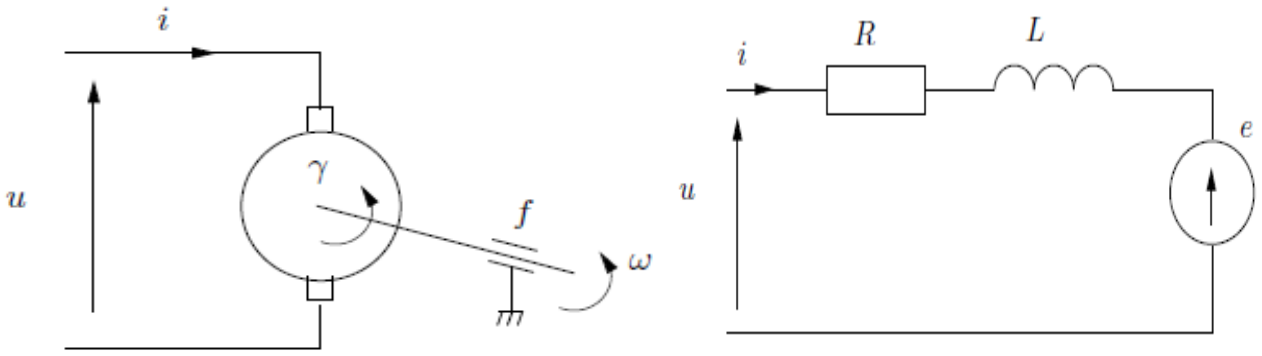


Figure 1.5 : Moteur à courant continu. [5]

- Pour la partie électrique :

On calcule la tension aux bornes de l'induit. L'équation électrique, liant la tension u aux bornes de l'induit et le courant d'induit i s'écrit :

$$Ri + L\frac{di}{dt} + e = u. \quad (1)$$

Où R est la résistance de l'induit du moteur, L son inductance et e la force électromotrice, qui est proportionnelle à la vitesse de rotation du rotor :

$$e = K_e\omega . \quad (2)$$

- Pour la partie mécanique :

On applique le principe fondamental de la dynamique autour de l'axe de rotation. L'équation mécanique rendant compte des couples agissant sur le rotor s'écrit :

$$\gamma - f\omega = J\frac{d\omega}{dt} . \quad (3)$$

Où γ est le couple moteur, f le coefficient de frottement visqueux et J le moment d'inertie du rotor. On ne tient pas compte en première approximation du frottement sec, qui introduirait des termes non linéaires. Par construction, le couple est proportionnel au courant d'induit i :

$$\gamma = K_m i . \quad (4)$$

En règle générale les coefficients Ke et Km sont si proches qu'il est raisonnable de les considérer égaux, négligeant alors les pertes durant la conversion électromécanique de puissance. En posant K = Ke = Km.

Les équations (3) et (4) donnent $K_i = f\omega + J\frac{d\omega}{dt}$. (5)

En dérivant l'équation précèdent (5) $K\frac{di}{dt} = f\frac{d\omega}{dt} + J\frac{d^2\omega}{dt^2}$. (6)

En combinant (5) et (6) avec (1) et (2) :

$$\frac{R}{K}\left(f\omega + J\frac{d\omega}{dt}\right) + \frac{L}{K}\left(f\frac{d\omega}{dt} + J\frac{d^2\omega}{dt}\right) + K_e\omega = u\,. \qquad (7)$$

Finalement, en ordonnant (7) de façon à avoir un coefficient de un devant le degré de dérivation le plus élevé, il vient :

$$\frac{d^2\omega}{dt} + \frac{RJ+Lf}{LJ}\frac{d\omega}{dt} + \frac{Rf+K^2}{LJ}\omega = \frac{K}{LJ}u\,. \qquad (8)$$

Donc à partir de ce modèle mathématique, il est possible de dériver un modèle dans l'espace d'état non linéaire en choisissant les variables d'état $x_1 = \omega$ et $x_2 = \dot{x}_1$:

$$\begin{cases} x_1 = \omega \\ x_2 = \dot{x}_1 \\ \dot{x}_2 = \frac{K}{LJ}u - \frac{RJ+Lf}{LJ}x_2 - \frac{Rf+K^2}{LJ}x_1 \end{cases} \qquad (9)$$

7.3. Un exemple en biologie : Proies-prédateurs (Lotka-Volterra)

- Description

Dans les années après la Première Guerre mondiale, la quantité de poissons prédateurs dans l'Adriatique était beaucoup plus élevée que pendant les années précédentes. En effet, les hostilités entre l'Italie et l'Autriche avaient provoqué une grande diminution de la pêche. Pour essayer de comprendre et expliquer pourquoi ce fait avait favorisé les prédateurs plutôt que les proies, le mathématicien Volterra avait proposé quelque équation exprimée ce phénomène.

Il avait supposé que le taux de croissance des populations des proies, en absence de prédateurs, était donné par une constante a. et qu'il décroissait linéairement en fonction de la densité y des prédateurs. De plus il avait supposé qu'en absence de proies, le taux de croissance des populations des prédateurs était négatif (ce qui conduit à la disparition de la population) et qu'il croissait linéairement en fonction de la densité x des proies.

- Le système de Lotka-Volterra

On suppose que la population de prédateurs se nourrit de celle des proies. Notons par x(t) la densité de la population des proies, et par y(t) celle des prédateurs dans le milieu.

Le phénomène qui se produit se traduit par le système suivant :

En posant $u\begin{pmatrix}x(t)\\y(t)\end{pmatrix}$:

$$\begin{cases}\dot{u}(t) = f\big(u(t)\big)\\ u(0) = \begin{pmatrix}x_0\\y_0\end{pmatrix}\end{cases} \qquad (1)$$

Ou encore

$$\begin{cases}\dot{x}(t) = x(a - by)\\ \dot{y}(t) = y(-c + dx)\end{cases} \qquad (2)$$

Avec f est de classe C1 donc f est localement liptshizienne, alors il existe une unique solution pour toute condition initiale, et a, b, c, d sont des constantes positives. Les termes (ax) et (-cy) traduisent le comportement de chacune des deux espèces en l'absence de l'autre.

En l'absence de prédateurs la population des proies augmente exponentiellement, par contre les prédateurs s'éteignent.

Les termes (-bxy) et (dxy) sont les termes "d'interaction".

Les méthodes générales sont coûteuses en temps de calcul et en espace pour les grands systèmes. Donc, toute méthode de décomposition du système d'équations en sous-systèmes simples à résoudre est intéressante [13].

8. La complexité de résolution des systèmes d'équation non linéaires

Généralement la complexité en temps des algorithmes de résolution des systèmes d'équations sont polynomiales. La bissection par exemple est en O (n^3). Si le système est de très grande taille, plus de 1000 équations par exemple, c'est très couteux de le résoudre et nécessite peut être des stations de travail très puissantes. Il est très nécessaire de trouver un moyen pour accélérer la résolution.

La résolution d'un système d'équation dépend du plus grand sous-système, Si un système S est décomposé en sous-système S1… S2……Sn, alors le temps de résolution dépend du plus grand sous-système Si.

Alors, résoudre un système de taille n prend plus de temps que la résolution séquentiel de deux systèmes de taille n' et n", tel que n'+n"=n. par exemple on a un système S de taille n=1000 équations, supposer que après la décomposition des systèmes en (S',S") tel que la taille de S' est n' =600 et la taille de S" est n"=400, donc on $1000^3 = 10^9$(unité de temps),et pour n' et n" on a max (n',n")= $600^3 = 216*10^6$. Si la taille du plus grand sous-système est égale à 3, nous aurons 9000 unités de temps au lieu de 10^9.

9. Conclusion

Comme on a vu dans ce chapitre, Il n'existe pas une méthode universelle de la résolution des systèmes des équations non linéaire, il-y-a plusieurs méthodes pour résoudre ces systèmes.

Mais les méthodes générales dans les grands systèmes sont coûteuses en temps de calcul et en espace, Donc toute méthode de décomposition du système d'équations en sous-systèmes simples à résoudre est intéressante [13].

Dans le chapitre suivant nous verrons une méthode de décomposition pour accélère la résolution de ces systèmes.

CHAPITER 2

ACCÈLÈRATION PAR DÈCOMPOSITION

1. Introduction

Dans ce chapitre nous allons considérer le graphe biparti associé à un système d'équations. Nous expliquons comment le décomposer en sous-systèmes bien contraints, sur-contraints et sous-contraints, et nous allons utiliser une méthode efficace de décomposition des systèmes bien contraints en sous-systèmes irréductibles afin d'accélérer le processus de résolution. Cette décomposition accéléré grandement la résolution dans le cas des grands systèmes réductibles, nous rappelons que la complexité de résolution des systèmes d'équations est généralement en O (n^3). [13]

2. Rappels

Nous donnons dans ce qui suit quelques définitions et notations de la théorie des graphes que nous utiliserons tout au long de ce chapitre (voir [10], [16]).

- **Graphe**

Un graphe fini G = (V, E) est défini par l'ensemble fini V= {v1, v2, . . ., vn} dont les éléments sont appelés sommets, et par l'ensemble fini E= {e1, e2, . . .,em} dont les éléments sont appelés arêtes (arcs cas d'un graphe orienté).

Une arête (ou arc) e de l'ensemble E est définie par une paire non ordonnée de sommets, appelés les extrémités de e. Pour tout arête e(x, y), x est appelé extrémité initiale et y est appelé extrémité terminale, on appelle ordre d'un graphe le nombre de sommets n de ce graphe, et la cardinalité d'un ensemble V sera noté | V |.

Le terme réseau et le terme Graphe l'un est utilisé pour désigner des systéme réels (réseau routier, Réseau électrique, réseau informatique, etc. …) , et le deuxième utilisé pour désigner une représentation mathématique d'un réseau. Mais, généralement: (Réseau ≡ Graphe).

- **Graphe: concepts non orientés**

Dans l'étude de certaines propriétés des graphes, il arrive que l'orientation des arcs, c'est à dire la distinction entre extrémité initiale et extrémité terminale, ne joue aucun rôle. On s'intéresse simplement à l'existence ou la non-existence d'un (ou de plusieurs) arcs entre deux sommets (sans en préciser l'ordre). A tout arc (x, y), on associe la paire (x, y) appelé arête. Un graphe est dit simple s'il est sans boucle et il n'y a jamais plus d'une arête entre deux sommets quelconques.

- **Successeur**

On dit que y est un successeur de x s'il existe un arc ayant son extrémité initiale en x et son extrémité terminale en y. L'ensemble des successeurs de x se note:

$$\Gamma_G{}^+(x).$$

- **Prédécesseur**

On dit que y est un prédécesseur de x s'il existe un arc de la forme (y, x).L'ensemble des prédécesseurs de x se note:

$$\Gamma_G{}^-(x).$$

- **Voisin**

L'ensemble des sommets voisins de x se note : $\Gamma_G(x) = \Gamma_G{}^+(x) \cup \Gamma_G{}^-(x)$.

- **Source et puits**

On appelle source tout sommet de G n'ayant pas de prédécesseur. On appelle puits tout sommet de G n'ayant pas de successeur.

- **Représentation graphique**

Les graphes tirent leur nom du fait qu'on peut les représenter par des dessins. À chaque sommet de G, on fait correspondre un point distinct du plan et on relie les points correspondant aux extrémités de chaque arête. Il existe donc une infinité de représentations d'un graphe. Les arêtes ne sont pas forcément rectilignes.

Exemple: On a défini le graphe G =(V,E) par ces ensemble des sommets :V = { 1, 2, 3, 4, 5}, et ensemble des arêtes :E = { (1, 3), (1, 4), (1, 5), (2, 3), (3, 4), (3, 5), (4, 5) }

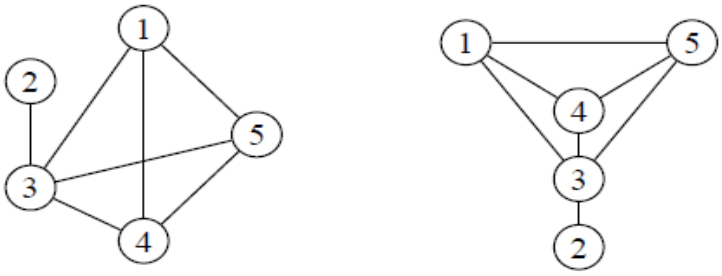


Figure 2.1: Un exemple pour représenter un graphe. [6]

- **Graphe biparti**

Un graphe G =(V, E) est biparti si ses sommets V peuvent être divisés en deux sous-ensembles X et Y, de sorte que toutes les arêtes du graphe relient un sommet dans X à un sommet dans Y .On le note G = (X, Y, E),où E est l'ensemble des arêtes.(dans l'exemple ci-dessous, on a V= {1,2,3,4,5} et E = { (1, 2), (1, 4), (2, 5), (3, 4), (4, 5) } tel que X = {1,3,5} et Y = {2,4}, ou vice versa).

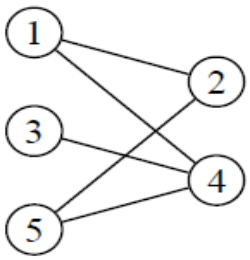


Figure2.2: Un graphe biparti. [6]

- **Graphe partiel et sous-graphe**

Soit G = (V, E) un graphe. Le graphe G' = (V, E') est un graphe partiel de G, si E' est inclus dans E. Autrement dit, on obtient G' en enlevant une ou plusieurs arêtes au graphe G.

Un sous-graphe est un graphe contenu dans un autre graphe, formellement , un graphe $H = (V_H, E_H)$ est un sous-graphe de$G = (V_G, E_G)$ si $V_H \subseteq V_G$ et $E_H \subseteq \{(x, y) \in E_G | x \in V_H \wedge y \in V_H\}$, l'ensemble des sommets du sous-graphe H est un sous-ensemble de l'ensemble des sommets de G et l'ensemble des arcs de H est un sous-ensemble de l'ensemble des arcs de G ayant leur origine et leur extrémité parmi les sommets de H.

- **Couplage**

Soit G(X, U) un graphe simple non orienté. Un couplage C du graphe G est un sous graphe partiel de G, ou un sous ensemble d'arêtes deux à deux non adjacentes (sans extrémités communes).

Un sommet x est dit saturé par le couplage C si x est l'extrémité d'une arête de C. dans le cas contraire, x est dit insaturé. Un couplage qui sature tous les sommets du graphe est appelé couplage parfait.

- **Couplage maximum**

Un couplage maximum est un couplage contenant le plus grand nombre possible d'arêtes (de cardinalité maximale |C| = max).Un graphe peut posséder plusieurs couplages maximum.

- **Graphe connexe**

Un graphe est dit connexe si pour tout couple x et y de sommets, il existe une chaîne reliant ces deux points.

- **Composante connexe d'un graphe G**

La relation R telle que : x R y s'il existe dans G une chaîne reliant x et y est une relation d'équivalence. Les classes de cette relation d'équivalence constituent une partition de V en sous-graphes connexes de G, appelés les composantes connexes de G.

- **Composante fortement connexe d'un graphe G**

Considérons un graphe G= (V, E) connexe quelconque et soit la relation R' dans G telle que : x R' y s'il existe un chemin allant de x à y et un chemin allant de y à x. Cette relation est une relation d'équivalence. Ses classes constituent une partition de V, et s'appellent les composantes fortement connexes du graphe G.

3. Méthode de décompositions [13][14]

Dans cette section, nous allons considérer le graphe biparti associé à un système d'équations. Ce graphe biparti possède un sommet par équation, un sommet par inconnue, et une arête entre une inconnue x et une équation y si, et seulement si, x apparaît dans l'équation y. les sommets "équations" sont les éléments de l'ensemble Y, les sommets "inconnues" sont les éléments de l'ensemble X. Ce type de graphe a été déjà utilisé, entre autres, par Serrano dans [6].

On donne quelques définitions intuitives. [13]

• Un système sur-contraint possède plus d'équations que d'inconnues, ses équations peuvent être redondantes, ou incompatibles et donc ne conduire à aucune solution.

• Un système sous-contraint possède plus d'inconnues que d'équations, il possède, en général, une infinité non dénombrable de solutions.

• Un système bien contraint possède autant d'équations que d'inconnues et ne contient aucun sous-système sur-contraint. Il possède donc un nombre fini de solutions

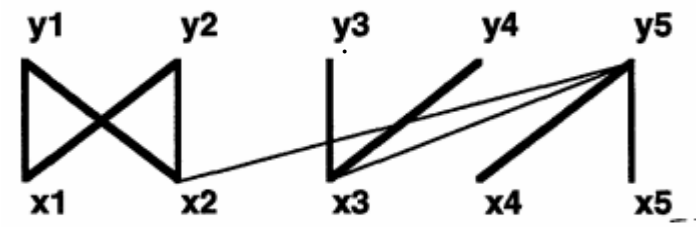


Figure2.3 : Les sous-systèmes (bien, sous, sur contrainte). [13]

Dans la figure 2.3, le sous-système {y1, y2. x1, x2} est bien contraint. Le sous-système {y3, y4, x3} est sur-contraint. Le sous-système {y5, x4, x5} est sous-contraint.

Dans le cas général, ce graphe est suffisant pour décomposer le système d'équations en trois sous-systèmes: bien contraint, sur-contraint, et sous-contraint. Chacun de ces sous-systèmes peut être· éventuellement vide. Cette décomposition existe toujours et elle est unique, elle est due- à Dulmage et Mendelsohn [1], [2], [3], [4].

En termes mathématiques, un système est bien contraint si, et seulement si, le graphe biparti associé satisfait à la relation de Konig-Hall: $|\Gamma(Y)| = |Y|$et pour tout sous-ensemble Y' de l'ensemble Y des équations, on a $|\Gamma(Y')| \geq |Y'|$.Autrement dit, le nombre d'inconnues apparaissant dans Y' est au moins égal au nombre d'équations dans Y'. Cette condition est équivalente à l'existence d'un couplage parfait. Tout graphe bien-contraint est soit irréductible, soit contient un sous-graphe irréductible I. [13]

• Si pour chaque sous-graphe G' on a : $|\Gamma(Y')| > |Y'|$alors G est irréductible d'après la définition.

• S'il existe un sous-graphe G' tel que $|\Gamma(Y)| = |Y|$ alors G' est bien contraint et soit irréductible soit contient un sous-graphe bien-contraint. On décompose ainsi les graphes bien-contraints jusqu'à aboutir à un irréductible. Cet irréductible est aussi contenu dans G.

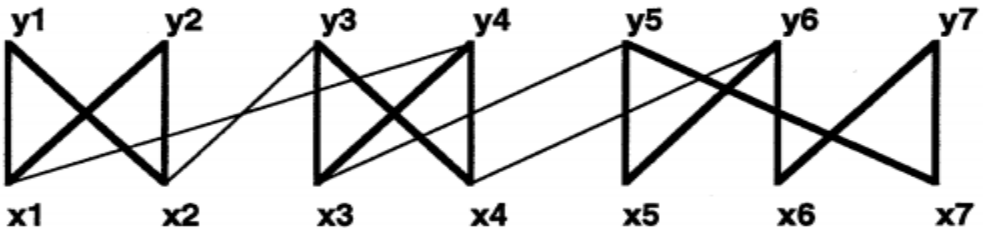


Figure 2.4 : décomposition du graphe G. [13]

Dans la figure 2.4, G est bien contraint. I1 = {y1, y2, x1, x2} est l'unique sous-graphe irréductible de G. I2 = {y3, y4, x3, x4} est l'unique sous-graphe irréductible de G-I1. Enfin, G- I1- I2 = {y5, y6, y7, x5, x6, x7} est irréductible.

Les graphes bipartis associés aux systèmes sur-contraints admettent aussi une telle décomposition, mais elle n'est pas unique.

Dans ce chapitre, nous présentons une méthode efficace pour obtenir une telle décomposition. Son coût est borné par le temps de recherche d'un couplage maximum.

Pour les systèmes sur-contraints, le graphe associé n'est pas suffisant pour décider si les équations sont redondantes ou incompatibles. Une méthode possible pour résoudre un tel système est de trouver un sous-système bien-contraint quelconque, de le résoudre et de vérifier si le reste des équations sont satisfaites ou non.

Pour les systèmes sous-contraints, nous n'avons pas assez d'informations pour calculer toutes les inconnues. Une méthode possible est de considérer certaines de ces inconnues comme des paramètres. On pourra les choisir de façon à obtenir un système bien-contraint, ensuite on pourra leur appliquer la méthode précédente.

4. Propriétés Structurelles des graphes bipartis

Dans cette partie nous avons présenté maintenant les résultats fondamentaux de la décomposition DM des graphes bipartis (voir [1], [2], [3] et [4]). L'algorithme de décomposition est décrit par ce qui suit :

4.1. Décomposition de Dulmage-Mendelsohn [13]

- Théorème1

Soit G = (V, E) un graphe biparti quelconque. V peut être partitionné en trois ensembles D, A, et C définis par :

• D est l'ensemble des sommets de Gnon saturés par au moins un couplage maximum.

• A est l'ensemble des sommets de V- D, adjacents à au moins un sommet de l'ensemble D.

• C est l'ensemble V- A - D.

Ces trois sous-ensembles sont uniques et conduisent à une décomposition unique de G en trois sous-graphes G1, G2, et G3, définit par:

• G1= (C1, C2, E1), où C1= C1 ∩ Y, C2 = C∩ X, et E1 est l'ensemble des arêtes induites par G1

• G2= (D1, A2, E2), où D1= D1∩Y, A2 =A ∩ X, et E2 est l'ensembledes arêtes induites par G2

• G3= (A1, D2, E3), où A1 = A ∩ Y, D2 = D ∩ X, et E3 est l'ensemble des arêtes induites par G3.

- Exemple

Soit G = (Vx, Vy, E) un graphe biparti associé à un système d'équations, on Vx={x1, x2, x3, x4, x5} et Vy={y1,y2,y3,y4,y5}, on a appliqué l'algorithme de couplage maximum M comme il est représenté dans la figure ci–dessous, et on prend le graphe suivant :

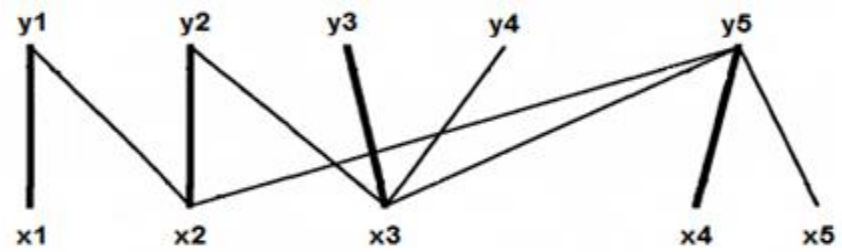


Figure 2.5: un couplage maximum [13]

Maintenant, et après la sélection du couplage maximum du graphe on a appliqué l'algorithme de Dulmage-Mendelsohn pour la décomposition du graphe au sous-graphe.

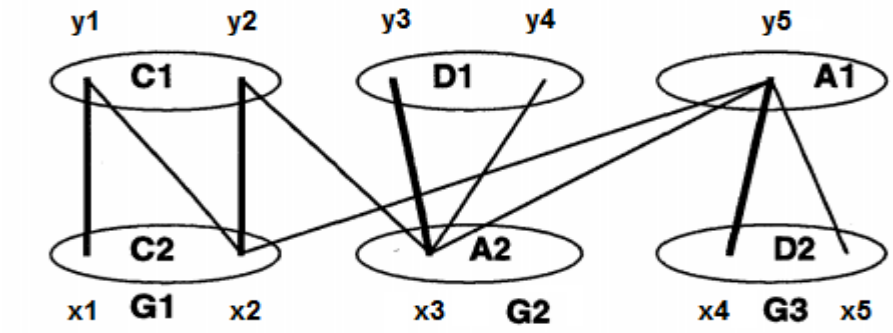


Figure 2.6 : décomposition du sous-graphe (G1, G2, G3). [13]

Dans la figure 2.6, G1 correspond à la composante bien-contrainte du système, G2 à la composante sur-contrainte, et G3 à la composante sous-contrainte.

Cette décomposition a les propriétés suivantes :

• Il n'existe aucune arête entre D1 et C2, entre D2 et C1, ou entre D1 et D2.

• G1 a un couplage parfait, donc |C1l = |C2l.

• Tout couplage maximum M de G consiste en un couplage parfait de G1, un couplage de A1 dans D2 (tous les sommets de A1 sont saturés, et au moins un sommet de D2 ne l'est pas), et un couplage de A2 dans D1 (tous les sommets de A2 sont saturés, et au moins un sommet de D1 ne l'est pas).Donc, IMI = IC1I + IA1I + IA2I, ID1I > IA2I,et IA1I< ID2I.

• Les arêtes entre C1 et A2, entre C2 et A1, ou entre A1 et A2, n'appartiennent jamais à un couplage maximum.

4.2. Graphes bipartis à couplage parfait [13]

Soit un graphe G= (V, E) ayant un couplage parfait: le système associé est structurellement bien contraint. G est irréductible si, et seulement si, pour tout sous-graphe propre Z, on a : |Γ(Z)|>|Z|. De manière équivalente, G est irréductible si, et seulement si, pour toute arête de G, il existe toujours un couplage parfait contenant cette arête (voir [13]). En d'autres termes, avec notre modèle, un graphe irréductible correspond à un système bien contraint dont tous les sous-systèmes propres sont sous-contraints.

- Théorème2

Soit H le graphe résultant de G après avoir supprimé de G toutes les arêtes qui n'appartiennent pas à un couplage parfait de G. Alors, les composantes connexes H1, H2, ..., Hk de H sont irréductibles.

- Preuve

Soit Hi une composante connexe. Supposons que Hi ne soit pas irréductible. Hi contient alors un irréductible I, et Hi - I est bien-contraint. II existe une arête A entre I et Hi - I.

Cette arête n'appartient à aucun couplage parfait car autrement I ne serait pas irréductible. D'où une contradiction avec la condition sur Hi dont toutes les arêtes appartiennent à un couplage parfait. Hi est donc irréductible.

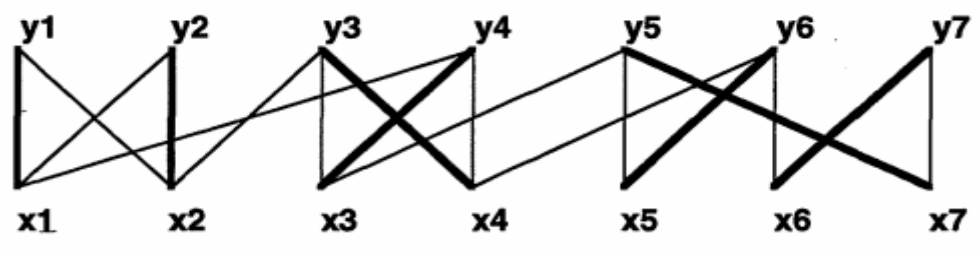


Figure 2.7: Un couplage parfait d'un graphe biparti G. [13]

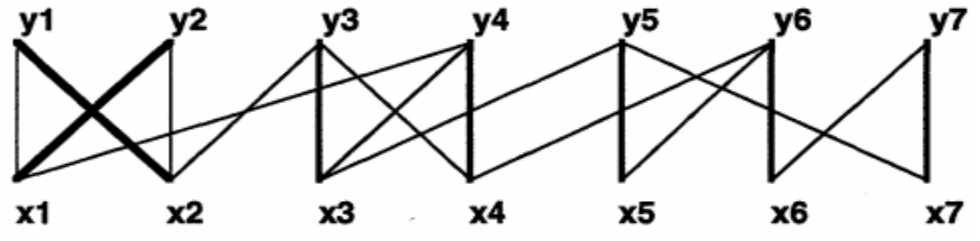


Figure 2.8: Un autre couplage parfait de G. [13]

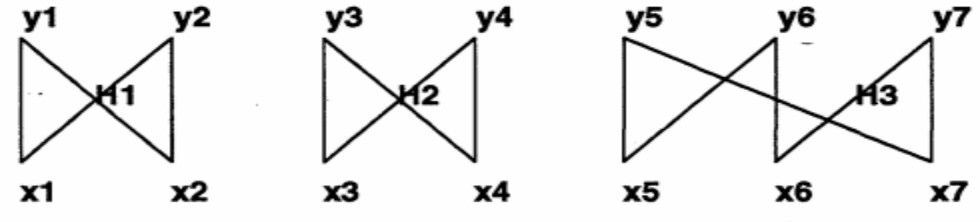


Figure 2.9: Le graphe H. [13]

Maintenant, la question est de trouver une méthode efficace pour déterminer cette décomposition. Les propriétés suivantes donnent une réponse à cette question.

- Propriété 1

Considérons un couplage parfait M de G. Soit G' le graphe orienté obtenu à partir de G en remplaçant chaque arête xy de M par deux arcs xy et yx, et en orientant toutes les autres arêtes de Y à X. Alors les composantes fortement connexes de G' sont exactement les composantes connexes de H: H1, H2, ... Hk.

- Preuve de la propriété 1

Il est suffisant de noter que nous avons : G est irréductible ⇔ G' est fortement connexe. Donc les composantes fortement connexes de G' sont irréductibles, elles correspondent nécessairement aux composantes H1, H2, ... Hk du théorème 2.

En fait cette construction apporte plus d'informations. Soit R le graphe orienté obtenu à partir de G en contractant chaque composante fortement connexe en un sommet de G' (ou H). Par exemple pour le graphe précédent, nous avons le graphe R donné par la figure 2.10.

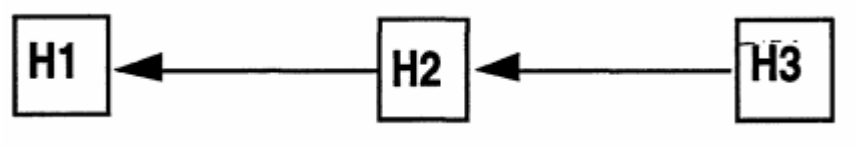


Figure 2.10 : graphe de résolution R [13]

Le graphe R est acyclique, donc R induit un ordre partiel sur H1, H2, ... Hk. Pour notre propos, si R a un arc de Hi vers Hj, alors le sous-système Hi utilise une (plusieurs) variable(s) du sous-système Hj, donc Hj doit être résolu avant Hi (voir chapitre 3, 2.3).

4.3. **Cas général** [13]

Considérons maintenant un graphe biparti G et soit M un couplage maximum de G. Nous définissons G' comme le graphe orienté obtenu à partir de G en remplaçant chaque arête xy de M par deux arcs xy et yx, et en orientant toutes les autres arêtes de Y vers X. Les composantes fortement connexes de G' sont incluses dans G1, dans G2 ou dans G3. De plus, si Y contient des sommets non saturés, alors ces sommets sont les sources de G2, G2 est donc non vide. Symétriquement, si X contient des sommets non saturés, alors ces sommets sont des puits de G3 qui est donc non vide. Ces propriétés résultent directement de la définition de G', du théorème 1 et de ses conséquences.

- Preuve

D'une façon similaire au cas des graphes ayant un couplage parfait, le graphe orienté R, obtenu à partir de G' en contractant chaque composante fortement connexe en un sommet, est acyclique. Ceci induit un ordre partiel entre les composantes fortement connexes. Donc tout ordre total compatible donne un ordre de résolution des sous-systèmes associés aux composantes fortement connexes de G1 et G2. D'autre part, G3 ne peut contenir de sous-graphe irréductible, on ne peut le résoudre que si nous affectons des valeurs à certaines inconnues correspondant à des sommets non saturés.

Il est important de mentionner que l'ordre de résolution de G2 est fondamentalement dépendant du couplage maximum choisi. En fait, G2 contient plusieurs sous-graphes

irréductibles, et la meilleure méthode est de résoudre le plus petit en cardinalité. Cependant, la recherche efficace du plus petit sous-graphe irréductible de G2 est un problème ouvert.

Appliquons la décomposition mentionnée ci-dessus au graphe G de la figure 2.11 avec le couplage maximum {y1x1, y2x2, y4x3, y6x4, y7x6} :

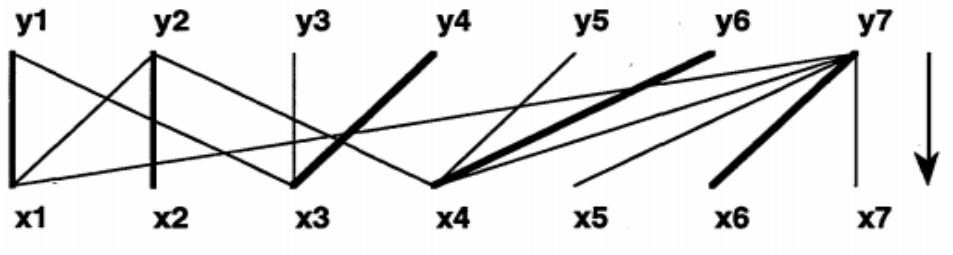


Figure 2.11 : un couplage maximum du graphe G. [13]

y3 et y5 sont les sommets non saturés de Y, et donc sont les sources de G 2· D'après la propriété 2, nous avons G2 = {y3, x3, y4, y5, x4, y6}, illustré par la figure 2.12. Notons que G2 a deux composantes connexes {y3, x3, y4} et {y5, x4, y6}.

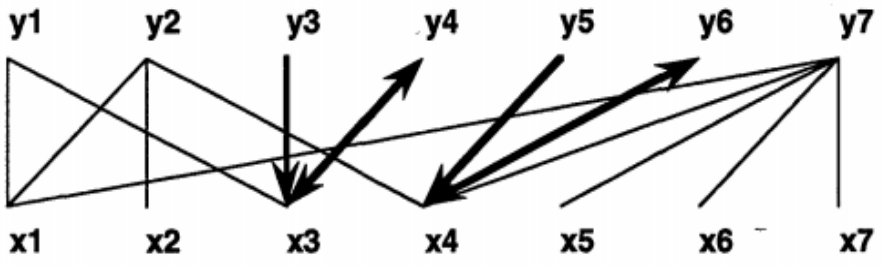


Figure 2.12 : calcule du graphe G2. [13]

x5 et x7 sont les sommets non saturés de X, ce sont donc les puits de G3. D'après la propriété 2, nous avons G3= {y7, x5, x6, x7}, illustré par la figure 2.13.

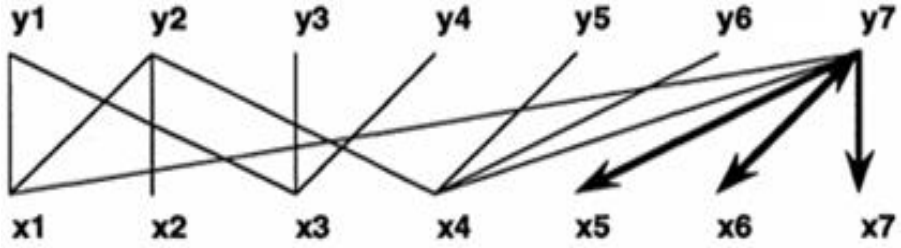


Figure 2.13 : calculer le graphe G3. [13]

Finalement, nous avons G1= G- G2- G3= {x1, x2, y1, y2}.

5. Algorithmes [13]

Soit G = (V, E) un graphe biparti associé à un système d'équations. Notons n= | V | et m = | E |. Nous donnons dans la section suivante les algorithmes pour obtenir la décomposition présentée précédemment (voir théorème 1).

5.1. DM-Décomposition

Les sous-graphes G1, G2 et G3 de G sont obtenus à l'aide de l'algorithme suivant:

1. Trouver un couplage maximum M de G.

2. Construire le graphe orienté G' à partir de G en remplaçant chaque arête xy de M par deux arcs xy et yx, et en orientant toutes les autres arêtes de Y vers X.

3. G2 est l'ensemble des descendants des sources de G'.

4. Symétriquement, G3 est l'ensemble des ancêtres des puits de G'.

5. Finalement, G1 est G- G2 - G3.

Pour l'étape 1 nous utilisons l'algorithme Ford-Fulkerson pour rechercher le couplage maximum, Le problème de couplage maximum peut être résolu en le convertissant en un réseau de flux, l'algorithme s'exécute en O ((m + n)*(n+m)).Les étapes 2, 3, 4 et 5 sont calculées en O (n2+ m).

Le calcul des composantes connexes de G1, G2 et G3 peut être fait par une recherche en profondeur ou en largeur en un temps linéaire.

5.2. Décomposition en parties irréductibles [13]

5.2.1. Systèmes bien contraints

Supposons que G soit bien contraint, G = G1 et G2 = G3 = 0. L'algorithme suivant donne l'unique décomposition de G en composantes irréductibles et donne un ordre de résolution entre ces différentes composantes (voir théorème 2).

- Algorithme

1. Trouver un couplage maximum M de G (dans ce cas M est un couplage parfait).

2. Construire le graphe orienté G' de G en remplaçant chaque arête xy de M par deux arcs xy et yx, et en orientant toutes les autres arêtes de Y vers X.

3. Calculer les composantes fortement connexes de G'. Chaque composante fortement connexe correspond à un irréductible.

4. Pour trouver les dépendances entre les sous-graphes irréductibles, construire le graphe R à partir de G' en contractant chaque composante fortement connexe en un sommet. Chaque arc de R allant d'un sommet Si vers un sommet Sj a la signification suivante : résoudre le sous-système Si avant le sous-système Sj, Un ordre total compatible entre ces sous-systèmes peut être obtenu par un tri topologique sur les sommets de R.

5.2.2. Systèmes bien contraints et systèmes sur-contraints

La méthode précédente peut être appliquée à G1 ∪ G2. Cependant, le couplage maximum M n'est pas parfait, et nous avons vu que la décomposition de G2 dépend du couplage maximum M, la méthode consiste à écarter les sommets non saturés de G2. Nous obtiendrons à ce moment-là un système bien contraint qui peut être complètement résolu. A la fin, nous avons vérifié que les solutions trouvées satisfont les équations rejetées.

6. Conclusion

La décomposition des systèmes en sous-système nous permet un premier niveau de l'accélération dans le temps de la résolution dans le cas des grands systèmes , mais la résolution des systèmes dans ce cas est séquentiel, c.-à-d. on décompose le système S en sous-système (S1,S2,…,Sn), puis on résoud le sous-système S1 puis le sous-système S2 jusqu'à Sn, donc le temps de la résolution dans le cas de l'accélération par décomposition est dépend de plus grand sous-système, dans le chapitre suivant on présente un deuxième niveau de l'accélération et l'accélération par parallélisation.

CHAPITER 3

ACCÈLÈRATION PAR PARALLÈLISATION

1. Introduction

Nous rappelons que la décomposition des systèmes en sous-système nous permet un premier niveau de l'accélération dans le temps de la résolution, mais le processus est séquentiel, alors dans ce chapitre nous allons proposer une méthode de parallélisassions pour plus d'accélérer de la résolution.

Dans ce chapitre nous présentons la parallélisassions de la résolution via une architecture Client/serveur. Nous présentons une méthode d'ordonnancement de taches appelée PERT (Program Evaluation and Review Technique), qui utilise le résultat de la décomposition comme entrée et comme sortie elle produit le schéma de parallélisassions, c'est-à-dire : l'ordre de résolution des différents sous système, ainsi que le nombre de serveurs nécessaires.

2. La parallélisation

2.1. Introduction aux parallélisation

Le concept de parallélisme est apparu à peu près en même temps que la science informatique elle-même. Les ordinateurs parallèles sont devenus de plus en plus abordables et certains efforts ont été faits dans le but de les rendre encore plus accessibles.

Le premier but du parallélisme est d'utiliser plusieurs processeurs de façon concurrente pour effectuer des calculs plus rapidement qu'avec un seul processeur.

2.2. La programmation parallèle [15]

La programmation parallèle consiste à résoudre un problème particulier en le divisant en plusieurs sous-problèmes indépendants, en créant plusieurs entités d'exécution, en les résolvant ensuite indépendamment à travers différents processeurs et en établissant une communication entre eux pour toute coordination, le cas échéant, afin de résoudre le problème indiqué. Donc, c'est le mécanisme de création de programmes qui permet d'utiliser efficacement les ressources de calcul disponibles en exécutant le code simultanément sur plusieurs nœuds de calcul.

Une des primitives de base du calcul parallèle sur systèmes multi-cœurs ou multiprocesseurs est le fil d'exécution (thread). Chaque processus du système peut créer un certain nombre de fils d'exécution qui partageront toutes les ressources du programme, à

l'exception des registres et de la pile, qui sont uniques à chaque fil. Ces fils peuvent ensuite s'exécuter sur une unité de calcul (processeur ou cœur) de manière simultanée. Puisqu'ils partagent le même espace mémoire, il est facile et relativement peu coûteux de créer de nouveaux fils.

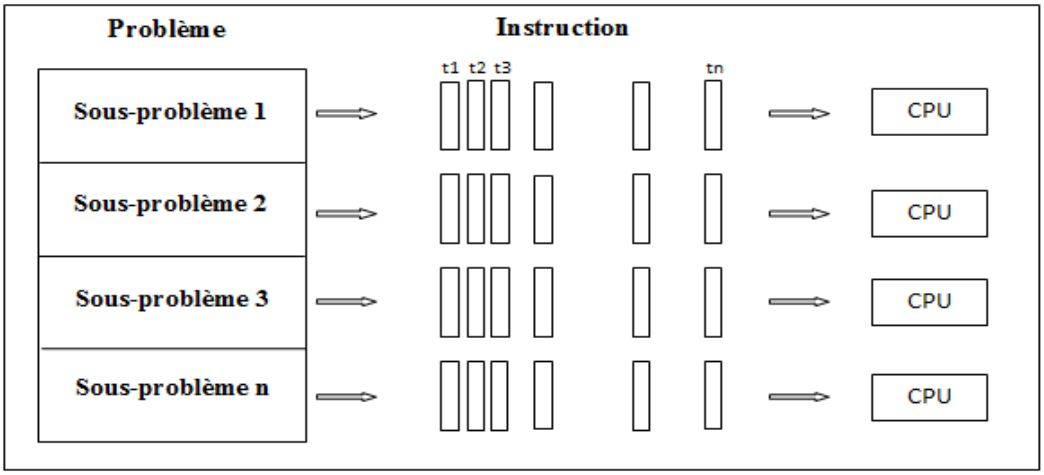


Figure 3.1 : un exemple de la programmation parallèle [15]

2.3. L'ordonnancement des tâches (sous-système)

Dans notre étude les tâches représentent les sous-systèmes. Donc après la décomposition d'un système au sous-système il faut l'organiser et ordonner les tâches selon le principe de l'antériorité pour déduire un graphe de résolution R en parallèle du ces sous-système, et pour obtenir le graphe R en utilisant la méthode de PERT.

2.3.1. La méthode PERT [7]

La méthode PERT a pour but de planifier la durée de résolution des sous-systèmes en parallèle, et pour obtenir la durée totale de la résolution on prend le chemin dont la succession des tâches donne la durée d'exécution la plus longue de la résolution et fournit le délai d'achèvement le plus court. Si l'on prend du retard sur la réalisation de ces tâches, la durée globale de la résolution est allongée. Dans ce cas le chemin est le chemin critique.[11]

2.3.2. Méthodologie de construction d'un réseau PERT [7]

1. Établir la liste des tâches (faire la décomposition du Dulmage-Mendelsohn).
2. Déterminer des antériorités : si Hi a un arc vers Hj, donc Hi doit être avant Hj.

3. Déterminer les niveaux d'exécution des tâches: Pour déterminer les niveaux d'exécution nous pouvons utiliser une matrice (ou grille) de dépouillement des données, on met une croix lorsqu'il y a une antériorité entre une tâche et une autre. On cherche s'il existe des croix dans l'une des colonnes. Si nous ne trouvons pas de croix dans certaines, cela signifie que les tâches repérées en haut des colonnes n'ont pas d'antériorité. Elles sont alors de rang 1. On note ces tâches, puis on barre les lignes horizontales correspondant à ces tâches et on réitère l'opération précédente. On détermine les tâches de rang 2 et ainsi de suite.

4. Construire le réseau PERT.

2.3.3. Exemple

Soit les sous-systèmes suivants qui constituent un graphe R : S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, et les antériorités sont les suivantes : S1 enclenche S3 et S4, S2 enclenche S4 et S5, S3 enclenche S6, S4 enclenche S7, S5 enclenche S7 et S10, S6 enclenche S8, S7 enclenche S9, la détermination des niveaux comme on a expliqué dans (2.3.1).

Nous en déduisons le réseau PERT correspondant à l'application proposée :

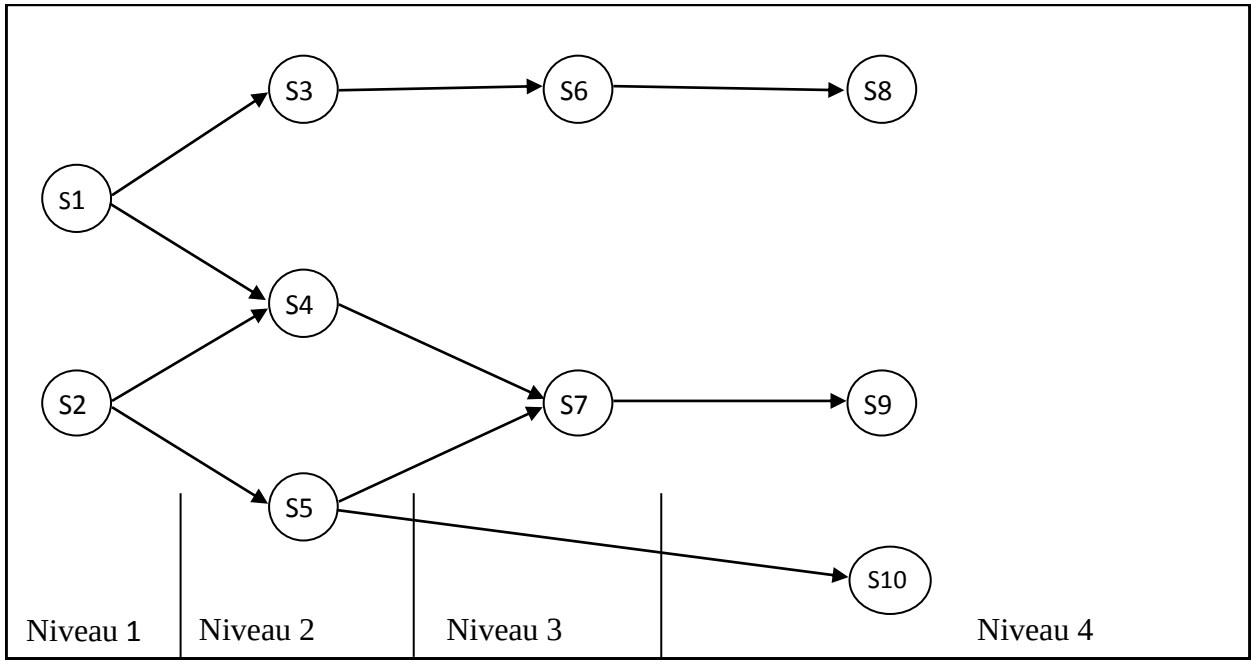


Figure 3.2 : le réseau PERT(le graphe de résolution R ordonner selon les niveaux d'exécution).

2.4. Parallélisation de la résolution

La parallélisation des systèmes d'équations nécessite plusieurs étapes :

1. Crée un serveur multithread.
2. Décomposé le système en sous-systèmes avec la méthode de décomposition de Dulmage-Mendelsohn.
3. Déduire le graphe de la résolution R.
4. Ordonner le graphe R selon le niveau d'exécution avec la méthode PERT.
5. Crée les clients selon le nombre de sous-systèmes dans chaque niveau.
6. Communication entre serveur et les clients par des sockets.

Le schéma suivant présente les étapes :

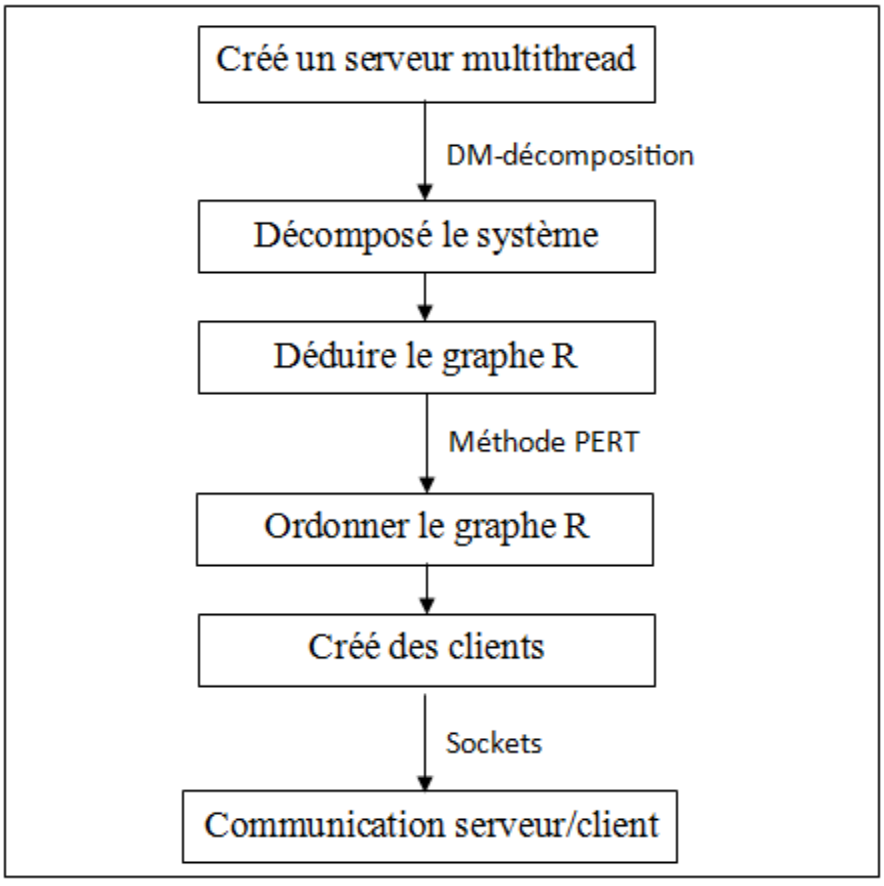


Figure 3.3 : Schéma de parallélisation de la résolution.

3. Conception de base des sockets [12]

3.1. Introduction aux sockets

La notion de sockets a été introduite dans les distributions de Berkeley (un fameux système de type UNIX, dont beaucoup de distributions actuelles utilisent des morceaux de code), c'est la raison pour laquelle on parle parfois de sockets BSD (Berkeley Software Distribution).

La communication par socket est souvent comparée aux communications humaines. On distingue ainsi deux modes de communication:

• Le mode connecté, utilisant le protocole TCP (Transmission Control Protocol). Dans ce mode de communication, une connexion durable est établie entre les deux processus, de telle façon que le socket de destination n'est pas nécessaire à chaque envoi de données.

• Le mode non connecté, utilisant le protocole UDP (User Datagram Protocol). Ce mode nécessite l'adresse de destination à chaque envoi, et aucun accusé de réception n'est donné.

- Les Sockets

Les sockets sont une interface de programmation de bas niveau pour la communication réseau, et est un mécanisme de communication entre des processus s'exécutant sur différents machines éloignées physiquement. Il se base sur la couche 4 (couche transport) de la pile protocolaire TCP/IP.

Une Socket est une liaison point à point entre un serveur et un client (le code des programmes est légèrement différent). La communication est full-duplex c.-à-d. Un socket définit un canal entre deux programmes : client et serveur. Le premier est l'initiateur de la communication à travers l'envoi d'un message (plus exactement d'une requête) vers le serveur. Le serveur, quand lui, est bloqué en attente d'une requête. Le protocole d'échange d'informations est laissé à la charge du programmeur.

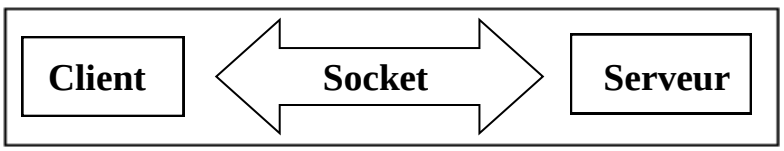


Figure 3.4 : Schéma de communication entre client et serveur

Le package java.net fournit deux classes nécessaires permettant de développer des applications Client/serveur. Ce sont les classes Socket et SocketServer.

- Le client

Le client est un logiciel installé sur l'ordinateur local qui permet d'une communication avec un serveur du réseau. C'est lui qui fait le premier pas lors d'une connexion avec un serveur en établissant une Socket.

- Le Serveur

Un serveur est un logiciel exécuté sur un certain ordinateur du réseau, il accepte commandes, questions, demandes et envoie une réponse automatiquement.il attend la connexion d'un client sur un port de l'ordinateur sur lequel il est installé et renvoie des données demandées. Dans un programme Java, le serveur doit créer une ServerSocket en indiquant son numéro de port puis attendre qu'un client demande une connexion qui peut alors être acceptée. Des flux d'entrées sont ensuite définis et les échanges peuvent commencer selon le protocole choisi. En fin de programme, tous les sockets doivent être libérés.

3.2. L'architecture Client/Serveur

Dans l'architecture Client/Serveur, le serveur fournit un service tel que le traitement des requêtes de bases de données et le client utilise ce service fourni pour une fin donnée : Traiter les résultats des requêtes. La communication qui se produit entre le client et le serveur doit être fiable (pas de perte de données) et les données doivent être reçues par le client dans le même ordre dans lequel elles ont été transmises. Le protocole TCP fournit une telle communication à travers laquelle les applications Client/Serveur sur Internet se communiquent.

- Client crée un socket avec (adrIP, numPort) et envoyer une requête.
- Un socket (en Java) définit deux flux : un flux de sortie (output Stream) et un flux d'entrée (input Stream).
  -Client utilise OS pour envoyer la requête et le IS pour recevoir la réponse
  -Serveur utilise le IS pour recevoir la requête et OS pour la réponse.
- Serveur écoute un port : numPort. Et s'exécute le service sur une machine dont l'adrIP, puis envoyer la réponse.

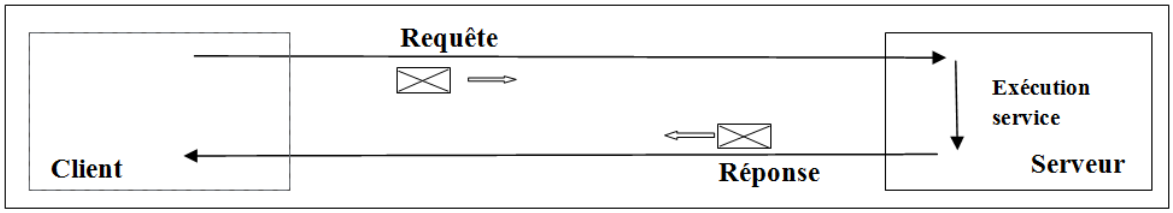


Figure 3.5 : L'architecture Client/Serveur. [12]

3.3. Les structures de communication Client/Serveur [12]

Dans cette partie nous allons présenter les algorithmes du client et serveur dans les deux types en mode connecté et non connecté, puis Schéma de la communication entre client et serveur, et les fonctions des sockets utilisés.

3.3.1. Algorithme d'un serveur en mode connecté

1. Création du socket serveur
2. Récupération de l'adresse IP et du numéro de port du serveur
3. Mise en mode passif du socket : elle est prête à accepter les requêtes des clients
4. (opération bloquante) : acceptation d'une connexion d'un client et création d'un socket service client, dont l'identité est rendue en retour
5. et 6. Lecture, traitement et écriture (selon algorithme du service)
7. Fermeture et remise en attente

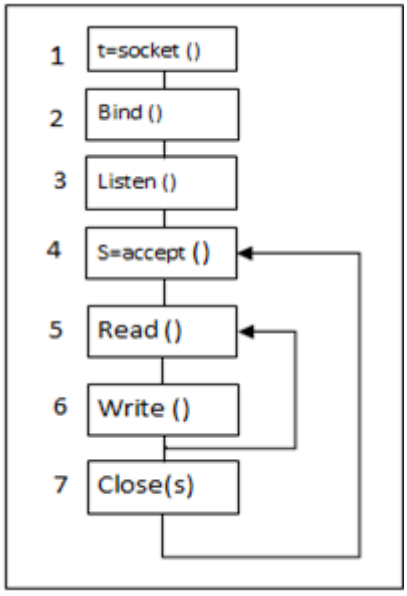


Figure 3.6 : Algorithme d'un serveur en mode connecté [12]

3.3.2. Algorithme d'un client en mode connecté

1. Création du socket
2. Connexion du socket au serveur : choix d'un port libre pour le socket par la couche TCP et attacher automatique à l'adresse (IP machine locale + n° de port), connexion du socket au serveur par l'adresse IP et le n° de port du serveur
3. et 4. Dialogue avec le serveur (selon algorithme du service)
5. Fermeture de la connexion avec le serveur.

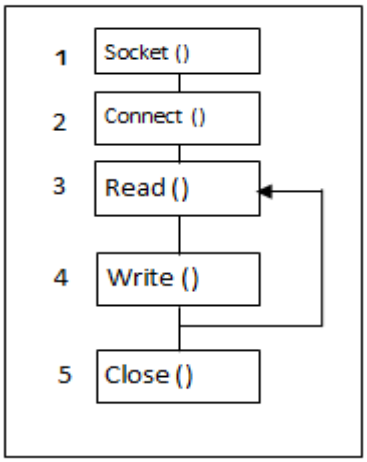


Figure 3.7 : Algorithme d'un serveur en mode connecté [12]

3.3.3. Algorithme d'un serveur en mode non connecté

1. Création du socket
2. Récupération de l'adresse IP et du numéro de port du serveur
3. Réception d'une requête de client
4. Traitement de la requête, préparation de la réponse
5. Réponse à la requête en utilisant le socket et l'adresse du client obtenu par recvfrom, retour pour attente d'une nouvelle requête.

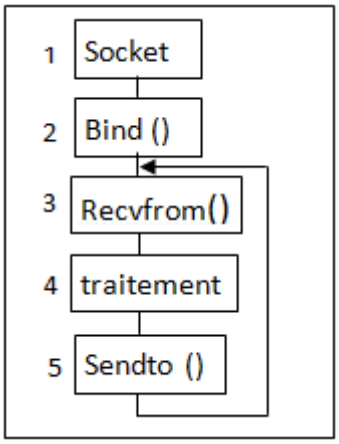


Figure 3.8 : Algorithme d'un serveur en mode non connecté [12]

3.3.4. Algorithme d'un client en mode non connecté

1. Création du socket (l'association à une adresse locale [adresse IP + n° port] est faite automatiquement lors de l'envoi de la requête)
2. Envoi d'une requête au serveur en spécifiant son adresse dans l'appel
3. Réception de la réponse à la requête
4. Fermeture du socket

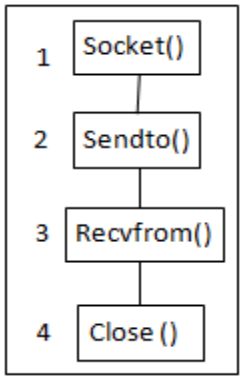


Figure 3.9 : Algorithme d'un client en mode non connecté [12]

3.4. Schéma de la communication Client/serveur [12]

3.4.1. Schéma d'une communication en mode non connecté

Nous allons tout d'abord voir, comment mettre en œuvre un dialogue entre un client et un serveur en mode datagramme. Le schéma ci-dessous illustre les différentes étapes que doivent accomplir un serveur (à droite) et un client (à gauche) pour pouvoir échanger des données en utilisant le protocole UDP.

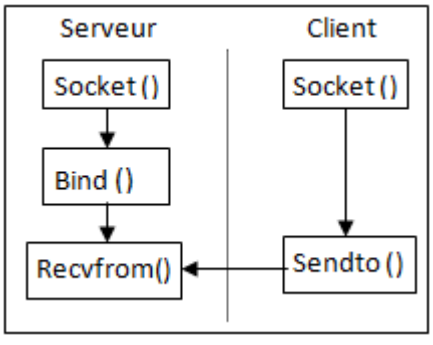


Figure 3.10 : Schéma d'une communication en mode non connecté [12]

3.4.2. Schéma d'une communication en mode non connecté

Nous allons maintenant voir comment mettre en œuvre un dialogue entre un client et un serveur en mode connecté. Le schéma ci-dessous illustre les différentes étapes à accomplir pour un serveur (à droite) et pour un client (à gauche).

On peut immédiatement remarquer que cet échange nécessite plus d'étapes qu'en mode datagramme. En effet, le mode connecté diffère du mode datagramme par deux aspects essentiels :

1. Il est nécessaire d'ouvrir une connexion, et de la fermer. On ne peut donc pas commencer directement par envoyer un segment.

2. Un serveur peut vouloir limiter le nombre de clients avec qui il interagit simultanément pour contrôler l'utilisation des ressources (mémoire, CPU, ...). En mode datagramme, la notion de connexion n'existant pas, la couche transport ne sait pas si une communication est terminée ou non. Elle ne peut pas gérer elle-même le nombre de clients connectés.

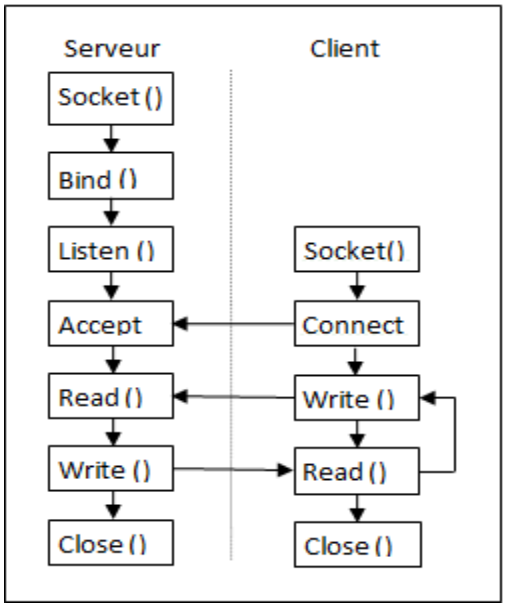


Figure 3.11 : Schéma d'une communication en mode non connecté [12]

3.5. Les fonctions des sockets [8]

1. La fonction socket () : création d'un socket.
2. La fonction bind () : le rôle de la fonction est après la création du socket, il s'agit de le lier à un point de communication local défini par une adresse et un port.
3. La fonction listen () : permet de mettre un socket en attente de connexion en mode connecté (TCP).

4. La fonction accept () : permet la connexion en acceptant un appel.
5. La fonction connect () : permet d'établir une connexion avec un serveur.
6. La fonction Read () : permet de lire dans un socket en mode connecté (TCP)
7. La fonction write () : permet d'écrire dans un socket (envoyer des données) en mode connecté (TCP)
8. La fonction recvfrom () : permet de lire dans un socket en mode non connecté (UDP)
9. La fonction sendto () : permet d'écrire dans un socket (envoyer des données) en mode non connecté (UDP)
10. Les fonctions close () : permet la fermeture d'un socket en permettant au système d'envoyer les données restantes (pour TCP).

4. Parallélisassions et communication par sockets

Comme on a déduit dans le chapitre précédant. Les traitements des clients multiples se fait séquentiellement, car les requêtes sont conservées dans une file d'attente associée au socket serveur.la complexité dans ce cas est égale la somme du temps de la résolution de requête.

Nous allons maintenant proposer un protocole de partage de sous-système et des données en plusieurs clients.

L'objectif de notre proposition est de donnée une méthode de traitement les clients multiples en temps parallèle.

4.1 Le principe

Le principe de la méthode est d'utiliser un serveur multithread. Le serveur est créé un nouveau thread exécutant à chaque nouvelle connexion d'un client (donc à l'exécution de accept()). Le nombre des clients dépend au nombre des sous-systèmes de niveau d'exécution dans le graphe de résolution R, la communication entre le serveur et les clients est fait selon le graphe de résolution R.

4.2 Les procédures

4.2.1 Procédure de serveur

1. Décomposer le graphe en sous-graphe.
2. Ordonner le graphe de résolution R.
3. Lire le graphe de résolution R.

4. Créer des clients selon le nombre de sous-système au niveau d'exécution.
5. Coder les sous-graphes dans un message socket.
6. Envoyer à chaque client un sous-système avec les variables.
7. Recevoir les solutions de chaque client.
8. Décoder les messages du client.
9. Ajouter les solutions à la table de solutions.

Procédure de serveur

```
Début
    Input
        Graphe ( G ) ;
        Int sol [ ] ;
        Nbr = size ( G ) ;
        Decompose ( G ) ;
        Order ( G ) ;
    Tanque ( Nbr != 0 ) {
        Si ( Nbr = size ( G ) )
            Crée_client ( size ( size ( G ( Nbr ) ) ) );
            Codage ( G ( Nbr ) ) ;
            Envoyer_client ( Client , G ( Nbr ) );
            Recevoir ( solution ) ;
            Decodage ( solution ) ;
            Ajouter ( solution , sol );
        Fin Si
        Si ( Nbr != size ( G ) )
            Crée_client ( size ( size ( G ( Nbr ) ) ) );
            Codage ( G ( Nbr ) , sol ) ;
            Envoyer_client ( Client , G ( Nbr ) , sol );
            Recevoir ( solution ) ;
            Decodage ( solution ) ;
            Ajouter ( solution , sol );
        Fin Si
    Nbr - - ;
    Fin tanque
Fin
```

Figure 3.12: Procédure de serveur.

4.2.2 Procédure de client

1. Le client recevoir le sous-système et les variables.
2. Décoder le message de socket.
3. résoudre le sous-système.
4. coder le message socket.
5. envoyer au serveur les solutions.

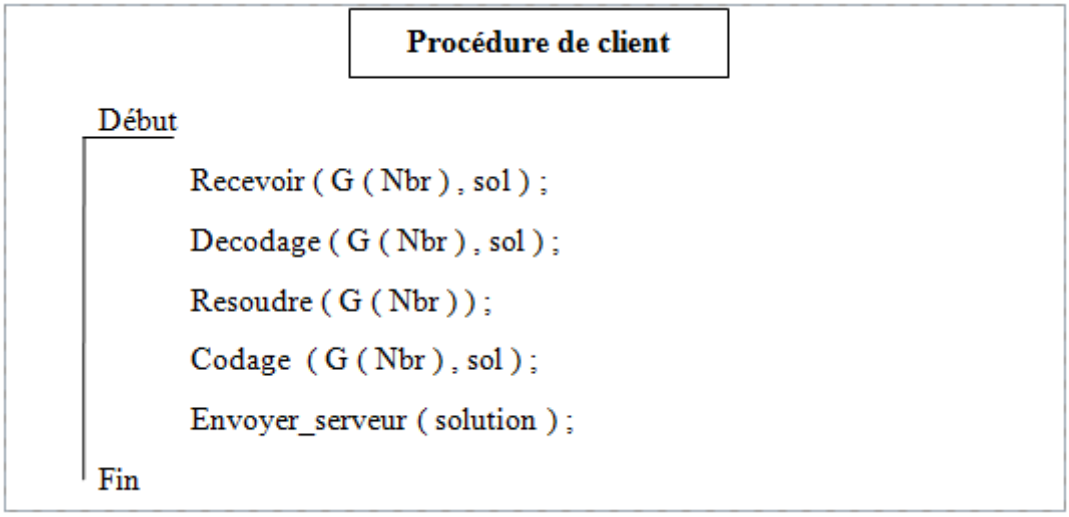


Figure 3.13: Procédure de client.

5. Conclusion

La parallélisation de la résolution des sous-systèmes nous permet un deuxième niveau d'accélération dans le temps de la résolution, et la complexité de la résolution des sous-systèmes en parallèle est égale la somme de temps de résolution des sous-systèmes de chemin critique dans le graphe de résolution R.

CHAPITER 4

RÉALISATION, TEST ET RÉSULTAT

1. Introduction

Dans ce chapitre nous présentons notre travail, celui-ci est composé en deux parties la première partie concerne la réalisation du travail et dans la deuxième partie nous donnons quelques exemples explicatifs.

2. Réalisation

2.1 Objectif

Notre objectif est de faire une accélération à la résolution des systèmes d'équations non linéaires par décomposition et parallélisation. L'approche du travail commence en première étape par une décomposition du système en sous-systèmes, et une parallélisation de la résolution de sous-systèmes, pour ensuite passer à une comparaison du temps de la résolution des sous-systèmes en parallèle et en séquentielle.

2.2 Principe du travail

Nous avons utilisé l'algorithme de décomposition de Dulmage-Mendelsohn proposé par A. L. Dulmage and Nathan Mendelsohn qui calcule la partition des sommets d'un graphe bipartite en sous-ensembles, avec la propriété que deux sommets adjacents appartiennent au même sous-ensemble si et seulement si, ils sont appariés dans un couplage parfait du graphe. L'algorithme permet de calculer le graphe de l'ordre de la résolution des sous-ensembles, la complexité de l'algorithme dépend du temps de calcul de couplage maximum du graphe.

Pour chercher sur le couplage maximum nous avons basé notre travail sur la méthode de Ford-Fulkerson et l'algorithme de parcours en profondeur DFS (Depth First Search).

Pour déterminer le graphe R et l'ordre de la résolution des sous-systèmes nous utilisons la méthode de PERT.

Pour la parallélisation nous avons proposé une méthode qui a utilisé l'architecture client /serveur avec les sockets.

2.3 Les outils utilisés

2.3.1 Les éléments matériels

Pour agrandir cette méthode, nous avons utilisé comme élément matériel un ordinateur hp qui possède comme particularités, un processeur Intel (R) Core (TM) i3-3110M CPU@ 2.40Hz 2.40GHz, une mémoire vive de 4 Go, et un disque dur 500 Go.

2.3.2 Les éléments logiciels

Nous avons implémenté cette méthode dans un environnement intégré NetBeans qui utilise le langage de développement JAVA, sur une plateforme Windows 7 professionnel édition intégrale service pack1.

Pour la résolution des systèmes d'équations non linéaires nous utilisons MATLAB sur un deuxième ordinateur qui possède un processeur Intel (R) Core (TM) i7 Q720 CPU@ 1.60Hz 1.60GHz, une mémoire vive de 16 Go.

2.4 La structure des données

Le système d'équation est présenté par une matrice, les lignes représentent les équations et les colonnes représentent les inconnus, si l'équation y contient l'inconnu x on a marqué 1 si non on a marqué 0. Mais dans le code java nous avons utilisé les listes des successeurs.

2.5 L'implémentation

La structure de travail et les fonctions utilisées sont organisées comme suit :

1. La fonction read () : permet de lire un fichier texte (.txt) et remplir une matrice.

```
52
53      public static void read(String s) throws Exception {
54        BufferedReader bf= new BufferedReader(new FileReader(s));
55          String line=bf.readLine();
56          while(line!=null){
57              countline++;
58              String[] words=line.split(" ");
59              countcolumns=words.length;
60              line=bf.readLine(); }
61          Scanner sc = new Scanner(new BufferedReader(new FileReader(s)));
62        rows = countline;
63        columns = countcolumns;
64         myArray = new int[rows][columns];
65          while(sc.hasNextLine()) {
            for (int i=0; i<myArray.length; i++) {
67              String[] lines = sc.nextLine().trim().split(" ");
68              for (int j=0; j<lines.length; j++) {
69                 myArray[i][j] = Integer.parseInt(lines[j]);}} }
70          System.out.println("le graphe est : ");
71          System.out.println(Arrays.deepToString(myArray));
72
73      }
```

Figure 4.1 : la méthode read ().

2. La fonction bpm () : nous permet de parcourir et d'essayer tous les inconnus X un par un, Si l'équation y s'intéresse à un inconnu x et que x n'est pas visité, marqué x comme visité.

```
75
76      static boolean bpm(int myArray[][], int u, boolean seen[], int matchR[]){
77                  for (int v = 0; v < countcolumns; v++) {
78                          if (myArray[u][v]==1 && !seen[v]) {
79                                  seen[v] = true;
80                          if (matchR[v] < 0 || bpm(myArray, matchR[v],seen, matchR)) {
81                                          matchR[v] = u;
82                                          return true;
83                                  }
84                          }
85                  }
86                  return false;
87          }
```

Figure 4.2 : la méthode bpm ().

3. La fonction maxBPM () : Créer un tableau pour garder la trace de l'équation assignée à un inconnu. La valeur de match R[i] est le numéro d'équation attribué à un inconnu i, et la valeur -1 indique que l'équation n'est pas assignée.

Au départ tous les inconnus sont disponibles, puis calculer le nombre d'inconnu attribué à l'équation, ensuite marquer tous les inconnus comme non vus pour la prochaine équation et tester si l'équation suivante peut avoir un inconnu.

Les résultats de cette méthode sont cinq listes (Liste des inconnus non saturés, Liste des inconnus saturés, Liste des équations non saturées, Liste des équations saturées et Liste du couplage maximum).

```
88
    static int tst[] = new int[countcolumns];
90      static int maxBPM(int bpGraph[][]) {
91              int matchR[] = new int[countcolumns];
92              for(int i = 0; i < countcolumns; ++i)
93                      matchR[i] = -1;
94              int result = 0;
95              for (int u = 0; u < countline; u++) {
96                      boolean seen[] =new boolean[countcolumns] ;
97                      for(int i = 0; i < countcolumns; ++i)
98                              seen[i] = false;
99                      if (bpm(myArray, u, seen, matchR))
100                             result++;
101             }
102             System.out.println("le couplage est : ");
103     for(int i=0;i<countcolumns;i++) {
104         if(matchR[i]>=0){
105             System.out.println(("("+(matchR[i])+","+(i)+")"));
106              equa_sa.add(matchR[i]);
107              G7.add("y"+matchR[i]);
108              inco_sa.add(i);
109         }
110         else{
111              inco_no_sa.add(i);}}
112         for (int i = 0; i <countline; i++) {
113             if(!equa_sa.contains(i)){
114                 equa_no_sa.add(i);
115             }
116         }
117                     return result;
118     }
119
```

Figure 4.3 : la méthode bpmMAX ().

Les deux fonctions bpm () et MAXbpm () sont basées sur la méthode de Ford-Fulkerson et l'algorithme de parcours en profondeur (DFS).

4. La fonction sous_graphe () : cette fonction est basée sur la méthode de Dulmage-Mendelsohn pour déterminer les sous-graphes (bien-contrainte, sous-contrainte et sur-contrainte).

Cette fonction teste, si la liste des équations non saturées est vide alors, elle retourne G2= ∅, et si la liste des inconnus non saturés est vide alors, retourne G3= ∅ , si non est retourne G1, G2, G3.

```
public static void sous_graphe(){
    if (!equa_no_sa.isEmpty()){
       for (int i = 0; i <equa_no_sa.size(); i++) {
          getVertexNeighbor(equa_no_sa.get(i));
            for (int j = 0; j < l.size(); j++) {
               if(inco_sa.contains(l.get((j)))){
                  G2.add(equa_sa.get(inco_sa.indexOf(l.get(j))));
                  G2.add(l.get(j));G2.add(equa_no_sa.get(i));
                  G5.add("y");G41.add("y");G5.add("x"); G41.add("x");G5.add("y");G41.add("y"); }}}
          for (int k = 0; k < G2.size(); k++) {
              G5.set(k,G5.get(k)+G2.get(k)); } }
    if (!inco_no_sa.isEmpty()){
       for (int i = 0; i <inco_no_sa.size(); i++) {
          getG3(inco_no_sa.get(i));
            for (int j = 0; j < l.size(); j++) {
               if(G7.contains("y"+l.get((j)))){
                   if(G8.contains("y"+l.get(j))){
                      G8.add("x"+inco_no_sa.get(i));G3.add(inco_no_sa.get(i)); G6.add("x"); }
                     else{
                     G3.add(inco_sa.get(equa_sa.indexOf(l.get(j))));
                     G8.add("x"+inco_sa.get(equa_sa.indexOf(l.get(j))));
                     G3.add(l.get(j));G8.add("y"+l.get(j));G3.add(inco_no_sa.get(i));
                     G8.add("x"+inco_no_sa.get(i));G6.add("x");G6.add("y");G6.add("x"); } } }}
          for (int k = 0; k < G3.size(); k++) {
                                G6.set(k,G6.get(k)+G3.get(k)); } }
          G1.addAll(equa_sa);yx(equa_sa.size());G4.addAll(o);
          G1.addAll(inco_sa);xy(inco_sa.size());G4.addAll(o);
          G1.addAll(inco_no_sa);xy(inco_no_sa.size());G4.addAll(o);
          G1.addAll(equa_no_sa);yx(equa_no_sa.size());G4.addAll(o);
           for (int k = 0; k < G1.size(); k++) {
              G42.add(G4.get(k));G4.set(k,G4.get(k)+G1.get(k));
          int index =0;
       for (int i = 0; i < G2.size(); i++) {
         if(G4.contains(G5.get(i))){
             index=G4.indexOf(G5.get(i));
              G4.remove(G5.get(i));G42.remove(index);G1.remove(index);} }
       index =0;
        for (int i = 0; i < G3.size(); i++) {
         if(G4.contains(G6.get(i))){
              index=G4.indexOf(G6.get(i));G4.remove(G6.get(i));G42.remove(index);
             G1.remove(index);  }}
System.out.println("G1 : "+G4+"  G2 : "+G5+ "  G3 : "+G8); }
```

Figure 4.4 : la méthode sous_graphe ().

5. La fonction fortconx () : nous permet de rechercher les fortement connexes du graphe G1.

Les entrées : La liste de graphe G1.

Les sorties : Liste des fortement connexe.

Étape 01: prend le premier élément de la liste puis l'ajouter à deux nouvelles listes avec "+" dans la première liste et "-" dans la deuxième.

Étape 02: parcourir sur la première liste et ajouter tous les successeurs des éléments de la liste, et à cheque ajoute un nouvel élément et ajouter "+".même travail pour la deuxième liste mais avec les prédécesseurs et ajouter "-".

Étape 03: tester sur les deux listes, si l'élément [i] appartient à la première liste et à la deuxième liste, on l'ajoute à la liste d'affichage du fortement connexe.

```
545  static void fortconx (){
546       String S;
          frtc      = new ArrayList<List<String>>();   frtc_alph = new ArrayList<List<String>>();
          frtc_num  = new ArrayList<List<Integer>>();  tst1      = new ArrayList<List<Integer>>();
549       String[] arc = c.split("[\\/\\-]");          int t=arc.length;
550       ArrayList<Arc> arcs = new ArrayList<>();
551    for (int i = 0,j=1; i <t; i=i+2,j=j+2){Arc a = new Arc(arc[i],arc[j]);  arcs.add(a); }
552    for (int i = 0; i <G1.size(); i++) { FRG1.add(i); }
553    for (int K = 0; K <G1.size(); K++) {
          tst1.add(new ArrayList<Integer>());       frtc.add(new ArrayList<String>());
          frtc_alph.add(new ArrayList<String>());   frtc_num.add(new ArrayList<Integer>());
556     if(!FRG1.isEmpty()){
557       ArrayList<Fortp> fconexp = new ArrayList<>(); ArrayList<Fortm> fconexm = new ArrayList<>();
558       S=""+FRG1.get(0);
559       Fortp a = new Fortp(S,"+");  fconexp.add(a);  Fortm z = new Fortm(S,"-");  fconexm.add(z);
560      for(int i = 0; i < arcs.size() ; i++) {
561       for(int j = 0; j < arcs.size() ; j++){
562         if(fconexp.size()>i)
563         if(fconexp.get(i).getSp().equals(arcs.get(j).getA())){
            Fortp s = new Fortp(arcs.get(j).getB(),"+");  fconexp.add(s);}
          if(fconexm.size()>i)
566       if( fconexm.get(i).getSm().equals(arcs.get(j).getB()) ){
            Fortm s = new Fortm(arcs.get(j).getA(),"-");  fconexm.add(s);}} }
568      for(int i = 0; i < fconexp.size() ; i++) {
569       for(int j = i+1 ; j < fconexp.size() ; j++){
570        if( fconexp.get(i).getSp().equals(fconexp.get(j).getSp())){ fconexp.remove(j);  j--; }}}
571      for(int i = 0; i < fconexm.size() ; i++)  {
572        for(int j = i+1 ; j < fconexm.size() ; j++){
573        if( fconexm.get(i).getSm().equals(fconexm.get(j).getSm())){fconexm.remove(j);  j--; }}}
       StringBuilder txt = new StringBuilder(); int m=0;
607      for (int i = 0; i <FR.size(); i++) {
608       if(FRG1.contains(FR.get(i))){FRG1.remove(FR.get(i));}  }
609      for (int i = 0; i <FR.size(); i++) {
610       if(FR.get(i)<G1.size()/2){frtc_alph.get(K).add("y"); frtc_num.get(K).add(G1.get(FR.get(i)));
611          frtc.get(K).add("y"+G1.get(FR.get(i)));}
612       else {frtc.get(K).add("x"+G1.get(FR.get(i))); frtc_alph.get(K).add("x");
613             frtc_num.get(K).add(G1.get(FR.get(i)));} }
614             tst1.get(K).addAll(FR);FR.clear(); }   }
616  int b=frtc.size();int c=0;
617      for (int i = 0; i <b; i++) {
618       if(frtc.get(i).isEmpty()){ c=i;break; } }
619      for (int i = 0; i <G1.size()-c; i++) {
620        frtc.remove(c);frtc_alph.remove(c);frtc_num.remove(c);tst1.remove(c); } }
```

Figure 4.5 : la méthode fortconx ().

6. La fonction graphe_R () : retourne le graphe de la résolution R.

```
600
601  static void graph_R(){
       graphe_R = new ArrayList<List<Integer>>();
603      for (int i = 0; i <frtc.size(); i++) {
           graphe_R.add(new ArrayList<Integer>());
605        for (int j = 0; j <frtc.get(i).size(); j++) {
606          if(frtc_alph.get(i).get(j).equals("y")){
607            int d=G4.indexOf("y"+frtc_num.get(i).get(j));
608            for (int m = 0; m <fortcon.get(d).size(); m++) {
609              if(!(tst1.get(i).contains(fortcon.get(d).get(m)))){
610                graphe_R.get(i).add(return_s(fortcon.get(d).get(m)));}}}}
611      getTranspose(graphe_R);
612  }
```

Figure 4.6 : la méthode graphe_R ().

7. La fonction graphe_S () : retourne l'ordre de la résolution des sous-systèmes solen les niveaux d'exécution.

```
623
   static List <List<Integer>> s= new ArrayList<List<Integer>>();
625 static boolean x=true;
626 static List<Integer> b;
627 static int dix=0;
628 static void graph_S(List<List<Integer>> a){
629     if(x==true){
            s.add(new ArrayList<Integer>());
            b=new ArrayList<Integer>();
632             for (int i = 0; i <a.size(); i++) {
633             if(a.get(i).isEmpty()) {
634                 dix++;
635                a.get(i).add(-1);
636                 b.add(i);  } }
637             graph_remove(b);
638             s.add(b);
639             if(dix!=0){
640                 dix=0;
641             graph_S(a);}
642             else x=false;
643             for (int i = 0; i <s.size(); i++) {
644             if(s.get(i).isEmpty()) {
645                s.remove(i); } }    }
646     System.out.println(s);
647 }
648
```

Figure 4.7 : la méthode graphe_S ().

3. Test et résultats

La décomposition des systèmes d'équations aux sous-systèmes est implémentées avec langage JAVA, mais nous avons résolu ces sous-systèmes et avons calculé le temps de résolution avec MATLAB. Nous présentons dans cette partie un exemple explicatif, d'un système décomposable.

3.1. Exemple explicatif

3.1.1. Exemple d'un système décomposable

Dans cet exemple nous avons un systéme d'équations décomposable de taille (40 équations), c.à.d. la matrice qui représente le graphe est une matrice creuse.

$$(x_1 + x_2)(x_1 + x_2) - x_4 - x_6 = 0$$
$$(x_1 - 1) = 0$$
$$(x_2 + x_3)^2 + x_1 = 0$$
$$2x_3^2 + 4x_3 + 2 = 0$$
$$x_6 - 20 = 0$$
$$\left(x_5^2 - x_6 + x_7\right) = 0$$
$$x_7 x_8 - 10 = 0$$
$$(x_7 + x_9)(x_7 - x_9) + 10 = 0$$
$$x_9 - 20 = 0$$
$$19x_{10} + 5x_{11} - 15x_{12} - 5 = 0$$
$$-4x_{10} - 12x_{11} + 8x_{12} + 3 = 0$$
$$4x_{10} + 10x_{11} + 3x_{12} - 4 = 0$$
$$2x_{13} - 3x_{31} + x_{14} - 5 = 0$$
$$-3x_{13} + x_{31} + 2x_{15} - 5 = 0$$
$$5x_{31} - 2x_{14} + 3x_{16} - 6 + 0$$
$$4x_{14} - 5x_{16} + x_{15} - 6 = 0$$
$$-4x_{13} + 2x_{16} - 3x_{15} + 17 = 0$$
$$x_{32} + 2x_{17} + 3x_{18} + 4x_{19} + 5x_{20} - 6 = 0$$
$$x_{32} + 3x_{17} + 5x_{18} + 7x_{19} + 4x_{20} - 13 = 0$$
$$x_{32} + 4x_{17} + 7x_{18} + 10x_{19} + 3x_{20} - 20 = 0$$
$$x_{32} + 2x_{17} + 4x_{18} + 6x_{19} + 8x_{20} - 13 = 0$$
$$x_{32} + 2x_{17} + 3x_{18} + 4x_{19} + 5x_{20} - 6 = 0$$
$$x_{21} + 5x_{22} + 3x_{23} - 2.5 = 0$$
$$6x_{21} + 2x_{22} + 3x_{24} - 1 = 0$$
$$4x_{21} + 2x_{23} + 5x_{24} + 1 = 0$$
$$4x_{22} + 6x_{23} + 9x_{24} - 3 = 0$$
$$3x_{25} - 2x_{26} + 4x_{27} - 17 = 0$$
$$2x_{25} - 3x_{26} + 2x_{27} - 14 = 0$$
$$5x_{25} + 4x_{26} - 6x_{27} - 1 = 0$$
$$x_{28} + 2x_{29} - x_{30} + x_{33} - 1 = 0$$
$$x_{28} + 3x_{29} + x_{30} - x_{33} - 2 = 0$$
$$-x_{28} + x_{29} + 7x_{30} + 2x_{33} - 3 = 0$$
$$2x_{28} + x_{29} - 8x_{30} + x_{33} - 4 = 0$$
$$-x_{34} - 4x_{35} - 4x_{36} - 27x_{37} = 0$$
$$x_{34} + 2x_{35} + 2x_{36} + 17x_{37} = 0$$
$$x_{34} + 2x_{35} + x_{36} + 12x_{37} = 0$$
$$\mathrm{x}_{37} - 10 = 0$$
$$x_{38} + x_{39} + x_{40} - 6 = 0$$
$$x_{38} + 2x_{39} + 2x_{40} - 11 = 0$$
$$x_{38} + 3x_{39} + x_{40} - 10 = 0$$

Le graphe biparti G associé à un systéme d'équation, possède un sommet par équation, un sommet par inconnu, et une arête entre une inconnue x et une équation y si, et seulement si, x apparaît dans l'équation y, et la matrice de graphe possède un 1 si l'équation y contient l'inconnu x, la matrice donnée par la figure 4.9 (voir chapitre 2 (2.3)).

Figure 4.8 : matrice de graphe du système d'équation.

La décomposition des systèmes d'équations se fait selon les étapes suivantes:

1- Lire le graphe à partir de la matrice par la méthode read ().

2- Trouver un couplage maximum du graphe G en utilisant les deux fonctions bpm () et bpmMAX ().

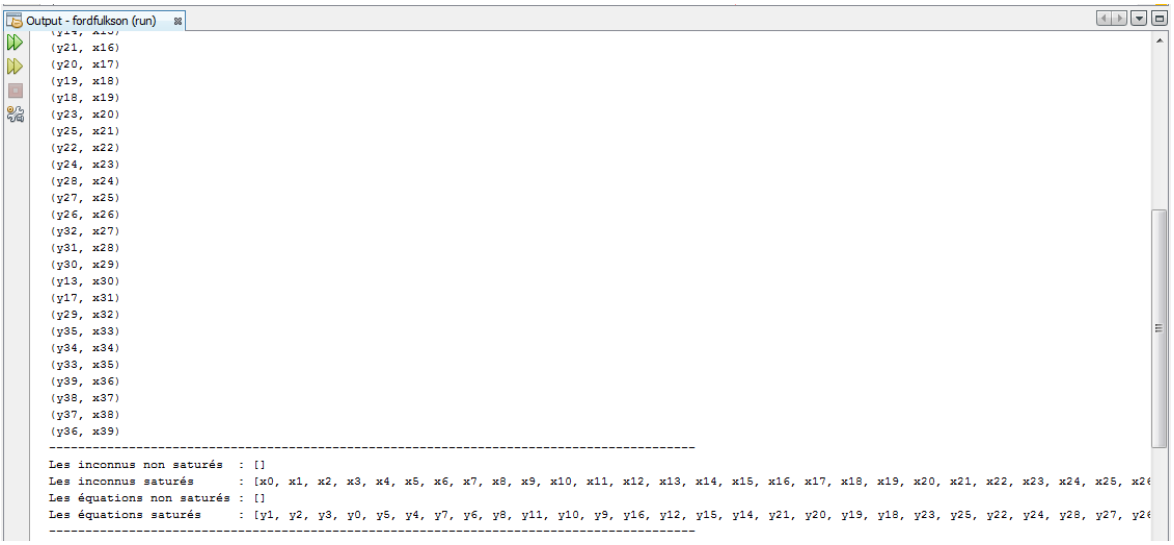


Figure 4.9 : Le résultat dans la console de NetBeans du couplage maximum

La figure 4.9 représente le résultat dans la console de NetBeans du couplage maximum, avec les équations et les inconnus saturés et non saturés dans le graphe G.

3- chercher du sous-système G1, G2 et G3 avec la méthode sous_graphe ().

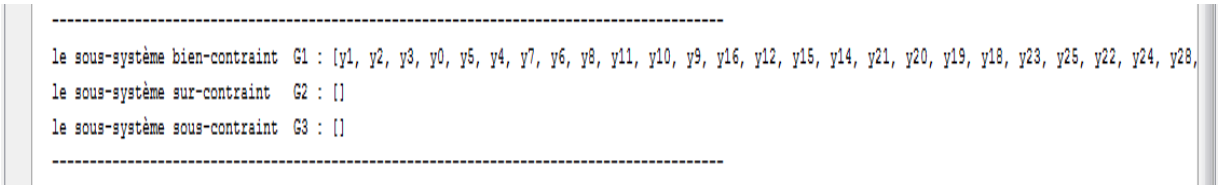


Figure 4.10 : Les sous-systèmes G1, G2 et G3.

4- Dans cette étape, nous supposons que G soit G1, puis nous retrouvé le couplage maximum M de graphe G1, et nous construire le graphe orienté G' de G en remplaçant chaque arête xy de M par deux arcs xy et yx, et en orientant toutes les autres arêtes de Y vers X. (voir chapitre 2 (5.2)). Après la construction du graphe G' nous calculé les composantes fortement connexes de G'. Chaque composante fortement connexe correspond à un sous-système irréductible.

```
--------------------------------------------------------------------------------
Les fortements connexe du sous-système Bien-contraint G1 :
[[y1, x0], [y2, x1], [y3, x2], [y0, x3], [y5, x4], [y4, x5], [y7, x6], [y6, x7], [y8, x8], [y11, x9, y10, y9, x10, x11], [y16, x12, y12, y13, x13, x30,
--------------------------------------------------------------------------------
```

Figure 4.11 : Les fortement connexe du sous-système bien-contraint G1.

5- Pour trouver l'ordre de résolution du graphe R, on construit le graphe R à partir de G' en contractant chaque composante fortement connexe en un sommet. Chaque arc de R allant d'un sommet Si vers un sommet Sj a la signification suivante : résoudre le sous-système Si avant le sous-système Sj.

```
--------------------------------------------------------------------------------
Le graphe R de résolution :
Les successeurs  : [[], [0, 2], [], [0, 1, 5], [5, 6], [], [8], [6], [], [], [], [], [], [], [], [16], [], []]
Les prédisseurs : [[1, 3], [3], [1], [], [], [3, 4], [4, 7], [], [6], [], [], [], [], [], [], [], [15], []]
Le graphe de résolution : [[3, 4, 7, 9, 10, 11, 12, 13, 14, 15, 17], [1, 5, 6, 16], [0, 2, 8]]
BUILD SUCCESSFUL (total time: 2 seconds)
```

Figure 4.12 : Le graphe de résolution R.

La résolution des sous-systèmes est faite avec MATLAB, et le temps de résolution obtenu est donné dans le tableau ci-dessous :

Le temps de la résolution du systèmes générale est : 71.6719s.

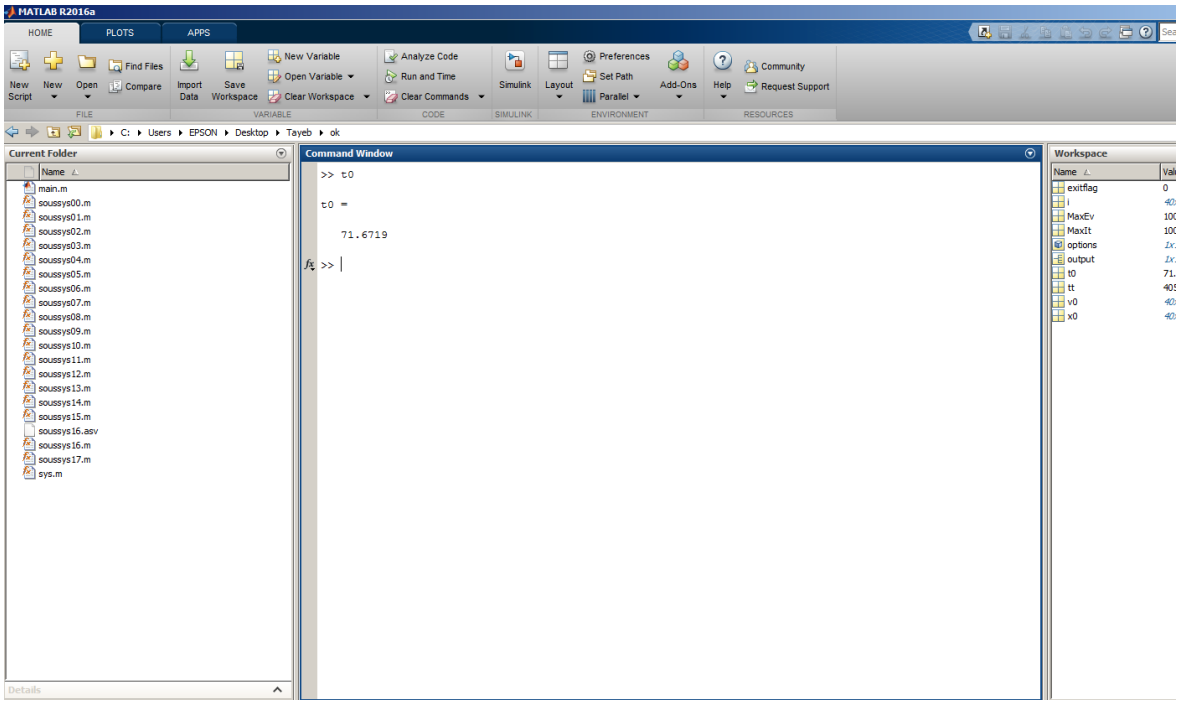
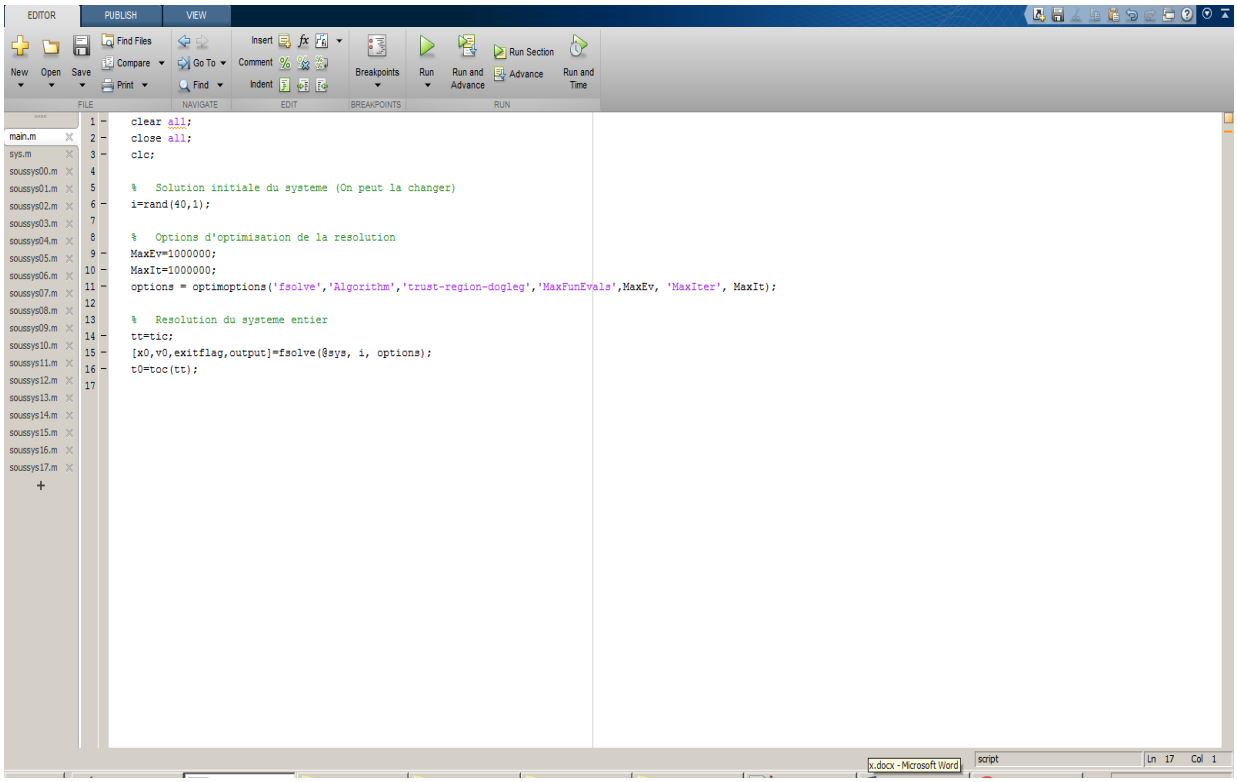


Figure 4.13 : Le temps de résolution du système en générale.



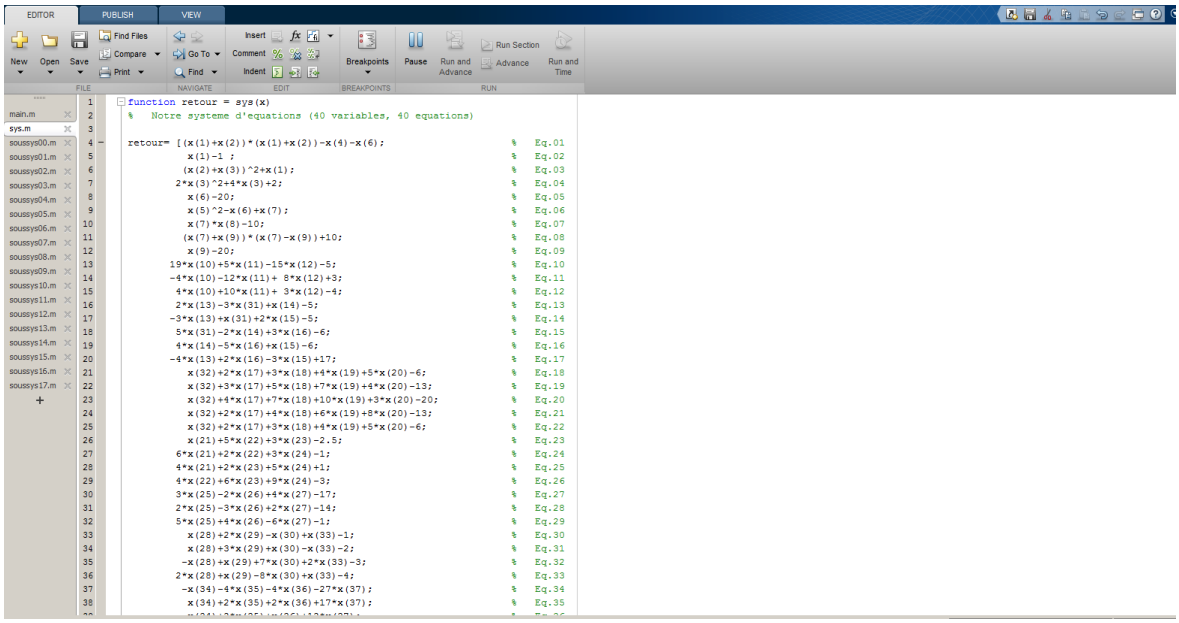
```
clear all;
close all;
clc;

%   Solution initiale du systeme (On peut la changer)
i=rand(40,1);

%   Options d'optimisation de la resolution
MaxEv=1000000;
MaxIt=1000000;
options = optimoptions('fsolve','Algorithm','trust-region-dogleg','MaxFunEvals',MaxEv, 'MaxIter', MaxIt);

%   Resolution du systeme entier
tt=tic;
[x0,v0,exitflag,output]=fsolve(@sys, i, options);
t0=toc(tt);
```

Figure 4.14 : Programmation de la résolution dans MATLAB .



```
function retour = sys(x)
%   Notre systeme d'equations (40 variables, 40 equations)

retour= [(x(1)+x(2))*(x(1)+x(2))-x(4)-x(6);                    %  Eq.01
          x(1)-1 ;                                             %  Eq.02
         (x(2)+x(3))^2+x(1);                                   %  Eq.03
        2*x(3)^2+4*x(3)+2;                                     %  Eq.04
          x(6)-20;                                             %  Eq.05
          x(5)^2-x(6)+x(7);                                    %  Eq.06
          x(7)*x(8)-10;                                        %  Eq.07
         (x(7)+x(9))*(x(7)-x(9))+10;                           %  Eq.08
          x(9)-20;                                             %  Eq.09
       19*x(10)+5*x(11)-15*x(12)-5;                            %  Eq.10
       -4*x(10)-12*x(11)+ 8*x(12)+3;                           %  Eq.11
        4*x(10)+10*x(11)+ 3*x(12)-4;                           %  Eq.12
        2*x(13)-3*x(31)+x(14)-5;                               %  Eq.13
       -3*x(13)+x(31)+2*x(15)-5;                               %  Eq.14
        5*x(31)-2*x(14)+3*x(16)-6;                             %  Eq.15
        4*x(14)-5*x(16)+x(15)-6;                               %  Eq.16
       -4*x(13)+2*x(16)-3*x(15)+17;                            %  Eq.17
          x(32)+2*x(17)+3*x(18)+4*x(19)+5*x(20)-6;             %  Eq.18
          x(32)+3*x(17)+5*x(18)+7*x(19)+4*x(20)-13;            %  Eq.19
          x(32)+4*x(17)+7*x(18)+10*x(19)+3*x(20)-20;           %  Eq.20
          x(32)+2*x(17)+4*x(18)+6*x(19)+8*x(20)-13;            %  Eq.21
          x(32)+2*x(17)+3*x(18)+4*x(19)+5*x(20)-6;             %  Eq.22
          x(21)+5*x(22)+3*x(23)-2.5;                           %  Eq.23
        6*x(21)+2*x(22)+3*x(24)-1;                             %  Eq.24
        4*x(21)+2*x(23)+5*x(24)+1;                             %  Eq.25
        4*x(22)+6*x(23)+9*x(24)-3;                             %  Eq.26
        3*x(25)-2*x(26)+4*x(27)-17;                            %  Eq.27
        2*x(25)-3*x(26)+2*x(27)-14;                            %  Eq.28
        5*x(25)+4*x(26)-6*x(27)-1;                             %  Eq.29
          x(28)+2*x(29)-x(30)+x(33)-1;                         %  Eq.30
          x(28)+3*x(29)+x(30)-x(33)-2;                         %  Eq.31
         -x(28)+x(29)+7*x(30)+2*x(33)-3;                       %  Eq.32
        2*x(28)+x(29)-8*x(30)+x(33)-4;                         %  Eq.33
         -x(34)-4*x(35)-4*x(36)-27*x(37);                      %  Eq.34
          x(34)+2*x(35)+2*x(36)+17*x(37);                      %  Eq.35
```

Figure 4.15 : Représentation des sous-systèmes dans MATLAB.

Sous-système	Equation		Tps séquentiel	Niveau	Tps parallèle
00	02	$(x_1 - 1) = 0$	0.2099	01	0.3543
02	04	$2x_3^2 + 4x_3 + 2 = 0$	0.2326		0.3693
08	09	$x_9 - 20 = 0$	0.2141		0.4041
01	03	$(x_2 + x_3)^2 + x_1 = 0$	0.2270	02	0.4199
05	05	$x_6 - 20 = 0$	0.2135		0.4190
06	08	$(x_7 + x_9)(x_7 - x_9) + 10 = 0$	0.2400		0.4395
16	37	$x_{37} - 10 = 0$	0.2160		0.5179
03	01	$(x_1 + x_2)(x_1 + x_2) - x_4 - x_6 = 0$	0.2022	03	0.3599
04	06	$(x_5^2 - x_6 + x_7) = 0$	0.2166		0.0916
07	07	$x_7 x_8 - 10 = 0$	0.2258		0.3909
09	10	$19x_{10} + 5x_{11} - 15x_{12} - 5 = 0$	0.2041		0.1188
	11	$-4x_{10} - 12x_{11} + 8x_{12} + 3 = 0$			
	12	$4x_{10} + 10x_{11} + 3x_{12} - 4 = 0$			
10	13	$2x_{13} - 3x_{31} + x_{14} - 5 = 0$	0.2062		0.3887
	14	$-3x_{13} + x_{31} + 2x_{15} - 5 = 0$			
	15	$5x_{31} - 2x_{14} + 3x_{16} - 6 + 0$			
	16	$4x_{14} - 5x_{16} + x_{15} - 6 = 0$			
	17	$-4x_{13} + 2x_{16} - 3x_{15} + 17 = 0$			
11	18	$x_{32} + 2x_{17} + 3x_{18} + 4x_{19} + 5x_{20} - 6 = 0$	0.2211		0.1081
	19	$x_{32} + 3x_{17} + 5x_{18} + 7x_{19} + 4x_{20} - 13 = 0$			
	20	$x_{32} + 4x_{17} + 7x_{18} + 10x_{19} + 3x_{20} - 20 = 0$			
	21	$x_{32} + 2x_{17} + 4x_{18} + 6x_{19} + 8x_{20} - 13 = 0$			
	22	$x_{32} + 2x_{17} + 3x_{18} + 4x_{19} + 5x_{20} - 6 = 0$			
12	23	$x_{21} + 5x_{22} + 3x_{23} - 2.5 = 0$	0.2118		0.4233
	24	$6x_{21} + 2x_{22} + 3x_{24} - 1 = 0$			
	25	$4x_{21} + 2x_{23} + 5x_{24} + 1 = 0$			
	26	$4x_{22} + 6x_{23} + 9x_{24} - 3 = 0$			
13	27	$3x_{25} - 2x_{26} + 4x_{27} - 17 = 0$	0.2094		0.1245
	28	$2x_{25} - 3x_{26} + 2x_{27} - 14 = 0$			
	29	$5x_{25} + 4x_{26} - 6x_{27} - 1 = 0$			
14	30	$x_{28} + 2x_{29} - x_{30} + x_{33} - 1 = 0$	0.2220		0.1476
	31	$x_{28} + 3x_{29} + x_{30} - x_{33} - 2 = 0$			
	32	$-x_{28} + x_{29} + 7x_{30} + 2x_{33} - 3 = 0$			
	33	$2x_{28} + x_{29} - 8x_{30} + x_{33} - 4 = 0$			
15	34	$-x_{34} - 4x_{35} - 4x_{36} - 27x_{37} = 0$	0.2122		0.1314
	35	$x_{34} + 2x_{35} + 2x_{36} + 17x_{37} = 0$			
	36	$x_{34} + 2x_{35} + x_{36} + 12x_{37} = 0$			
17	38	$x_{38} + x_{39} + x_{40} - 6 = 0$	0.2137		0.1096
	39	$x_{38} + 2x_{39} + 2x_{40} - 11 = 0$			
	40	$x_{38} + 3x_{39} + x_{40} - 10 = 0$			

Table 4.1 : comparaison de temps de résolution.

Donc d’après les méthodes précédant le temps de la résolution séquentiel est 3.8982s, et le temps de la résolution parallèle est 1.1478s.

4. Conclusion

Les méthodes de décomposition nous permettent d'accélérer le processus de résolution de systèmes d'équations, cette méthode est réussie et très prometteuse dans les cas des matrices creuses et les systèmes largement décomposables.

CONCLUSION GÉNÉRALE

La complexité de la résolution des systèmes d'équations non linéaire de grande taille (par exemple un système qui contient 1000 équations) avec les méthodes classiques est $O(n^3)$, alors notre objectif était d'accélérer la résolution de ces systèmes par la décomposition des systèmes en sous-systèmes, ensuite faire une parallélisation a la résolution de sous-systèmes, puis nous avons fait une comparaison du temps entre la résolution en parallèle et en séquentiel.

Dans ce travail nous avons réalisé l'accélération de la résolution des systèmes par deux niveaux:

Dans le premier niveau, nous avons utilisé l'algorithme de la décomposition du Dulmage-Mendelsohn pour décomposer les systèmes en sous-systèmes et avons extrait la partition des systèmes que nous avons pue résoudre, puis nous avons calculé le graphe de la résolution des sous-systèmes.Ce pour nous permet de gagner un certain temps même si la résolution est faite séquentiellement.

Nous avons aussi ajouté un deuxième niveau d'accélération qui est la parallélisation. Nous avons utilisé les sockets et l'architecture client/serveur pour faire une structure de résolution des sous-systèmes selon l'ordre de graphe de résolution. La méthode d'ordonnancement PERT nous a fourni le schéma de résolution. Ce niveau d'accélération nous permet de gagner beaucoup de temps.

Nous avons testé notre méthode sur des systèmes décomposables de taille (40 équations maximum), cette accélération permet de gagner beaucoup de temps, surtout si nous avons des systèmes largement décomposables (les matrices des systèmes sont creuses), mais si nous avons des systèmes très reliés (non décomposable) l'accélération ne permet pas gagner beaucoup de temps.

Cette méthode est réussie et très prometteuse dans les cas des matrices creuses et les systèmes largement décomposables, c.-à-d. les systèmes dans ce cas seront décomposés en un grand nombre des sous-systèmes, par exemple si nous avons un systéme d'équation de taille (500 équations), et la taille de plus grand sous-système est (5 équations), donc le nombre des sous-systèmes est au moins égale (100 sous-systèmes) et la complexité de la résolution est

fortement accélérée, mais malgré que les systèmes qui ne sont pas largement décomposables n'accélèrent pas la résolution, nous pouvons gagner un 50% de temps.

comme perspective on va tester cette méthode de décomposition sur des grand systèmes d'équations réel ou bien naturel ou bien industrielle, les données des systèmes ce sont des données physiques réel, par exemple un systéme solaire, un systéme dans la physique nucléaire ou bien un systéme dans l'industrie ... etc. Le teste de cette méthode ce fait est selon les types des systèmes d'équations donnée, c.-à-d. dans les systèmes mécanique nous pouvons utiliser cette méthode dans les logicielles mécanique, pour les systèmes biologique on va tester cette méthode dans les logicielles de biologie.

BIBLIOGRAPHIE

[1] A.L. Dulmage, N.S. Mendelsohn, Coverings of bipartite graphs, Canad. J. Math, 10, 1958, 517-534.

[2] A.L. Dulmage, N.S. Mendelsohn, A structure theorie of bipartite graphs of finite exterior dimension. Trans. Of Royal Soc, Canada, Section III, 53, 1959, 1-13.

[3] A.L. Dulmage, N.S. Mendelsohn, On the .inversion of sparsed matrices, Math. Comput, 16, 1962 494-496.

[4] A.L. Dulmage, N.S. Mendelsohn, Two algorithms for bipartite graphs, SIAM J., 11, 1963, 183-194.

[5] Bernard BAYLE, Cours Automatique Continue chapitre 2 page 6-7, Université de Strasbourg -Télécom Physique Strasbourg.

[6] D. Serrano, Automatic dimensioning in design for manufacturing. Proceeding Symposium on Solid Modeling Foundations and CAD/CAM Applications, Austin Texas. June 91. pp. 379-386.

[7] Edmond Maurel, Daniel Roux et Daniel Dupont, Techniques opérationnelles d'ordonnancement, EYROLLES, mars 1977.

[8] Ellen Siever, Stephen Spainhour et Nathan Patwardhan, Perl in a Nutshell, Linda Mui, 1999.

[9] Jacques Rappaz et Marco Picasso, Introduction à l'analyse numérique, Presses polytechniques et universitaires romandes, Lyon France, 2017.

[10] M. Gondran et M. Minoux. Graphes et algorithmes. Editions Eyrolles 1979.

[11] Philippe Etchecopar et Céline Saint-Pierre, DÉMARCHE DE MODÉLISATION EN MATHÉMATIQUES, Saut quantique.

[12] Sacha Krakowiak, Programmation client-serveur sockets – RPC, Université Joseph Fourier.

[13] Samy Ait-Aoudia, Modélisation géométrique par contraintes : quelques méthodes de résolution, thèse doctorat, université de Saint-Etienne et de l'école national supérieure des mines de Saint-Etienne, 24 Juin 1994.

[14] Samy Ait-Aoudia, Roland Jegou, Dominique Michelucci, reduction of constraint systems, Ecole des Mines de Saint Etienne 1993.

[15] Experiences math cnrs, https://experiences.math.cnrs.fr/Le-modele-proie-predateur-de-Lotka.html, consulté le 07/04/2019.

[16] Jacques Labelle, Théorie des graphes, Modulo, 1981.

ملخص

يعد حل أنظمة المعادلات غير الخطية بالطرق العامة الكلاسيكية في حالة الأنظمة الكبيرة مكلفًا من حيث زمن حساب المعادلات و مساحة التخزين، ولهذا السبب فإن أي طريقة لتسريع حل أنظمة المعادلات مثير للاهتمام.

في هذه المذكرة قدمنا طريقة لتسريع عملية حل هذه الأنظمة باستخدام التحليل و الموازاة.

في هذا العمل، اعتمدنا على مبدأ خوارزمية Dulmage-Mendelsohn لتحليل الأنظمة الكبيرة إلى أنظمة فرعية غير قابلة للاختزال، و كذلك قمنا باستخدام بنية (العميل و الخادم) ومآخذ التوصيل للموازنة.

الكلمات المفتاحية: أنظمة المعادلات غير الخطية، تحليل Dulmage-Mendelsohn، التوازي ، المقابس.

Abstract

The resolution of systems of nonlinear equations with general methods is expensive in computation time and in space in the case of large systems, for this reason any method of acceleration is interesting.

In this thesis we present a method of acceleration of the resolution by decomposition and parallelization, we use the Dulmage-Mendelsohn algorithm to break down systems into irreducible subsystems. We use client / server architecture and sockets for parallelization.

Key words: nonlinear equation systems, Dulmage-Mendelsohn decomposition, parallelization, sockets.

Résume

La résolution des systèmes d’équations non linéaires avec les méthodes générales sont coûteuses en temps de calcul et en espace dans le cas de grands systèmes, pour cette raison toute méthode d’accélération est intéressante.

Dans ce mémoire nous présentons une méthode d’accélération de la résolution par décomposition et parallélisation, nous utilisons l’algorithme de Dulmage-Mendelsohn pour décomposer les systèmes en sous-systèmes irréductibles. Nous utilisons l’architecture client /serveur et les sockets pour la parallélisation.

Mots clés : systèmes d'équation non linéaire, décomposition de Dulmage-Mendelsohn, parallélisation, les sockets.