Estudo de Algoritmos Combinados Paralelos Assíncronos

Benjamín Barán Cegla

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Eugenius Kaszkurewicz, D.Sc.
(Presidente)

Prof. Djalma Mosqueira Falcão, Ph.D.

Prof. Amit Bhaya, Ph.D.

Prof. José Mario Martinez, D.Sc.

Prof. Sérgio Granville, D.Sc.

RIO DE JANEIRO, RJ — BRASIL
OUTUBRO DE 1993

BARÁN, BENJAMÍN
Estudo de Algoritmos Combinados Paralelos Assíncronos
[Rio de Janeiro] 1993
xv, 149 p. 29,7 cm (COPPE/UFRJ, D.Sc.,
Engenharia de Sistemas e Computação, 1993)
Tese — Universidade Federal do Rio de Janeiro, COPPE
1. Computação Paralela
2. Algoritmos Combinados Assíncronos
3. Algoritmos bloco iterativos
I. COPPE/UFRJ II. Título (série).

À memória de meu pai, Manfredo Barán, a pouco de sua partida sem ter visto cristalizado este sonho; em homenagem a minha adorada mãe Betty, e em gratidão a minha esposa Rosana e meus filhos Taly, Anahi e Michel, por me acompanhar neste longo caminhar por terras distantes...

Agradecimentos

Minha mais profunda e sincera gratidão ao Prof. Eugenius Kaszkurewicz, artífice fundamental deste trabalho, pela ajuda constante, os conselhos que guiaram estas pesquisas, a confiança, mesmo nos dias em que o assincronismo parecia-me um mistério indecifrável, e a compreensão nos momentos difíceis.

Quero agradecer também aos professores Djalma M. Falcão e Amit Bhaya pelo soporte que me brindaram nestes anos de trabalho. Ao Prof. Falcão, especialmente, por ter-me dado a primeira oportunidade de trabalhar em computação paralela; ao Prof. Bhaya pelas discussões e conselhos que muito ajudaram à formulação deste trabalho.

Uma menção especial a todos os companheiros de trabalho do Laboratório de Computação Paralela que proporcionaram o ambiente de estudo adequado, na pessoa de Francisco Mota, por sua incansável predisposição a dar mais um 'boot' no sistema nas primeiras experiências com assincronismo.

Minha gratidão também a todas as pessoas e instituções que acreditaram em mim nestes anos de estudo, na perssoa do Ing. Hugo Cataldo; assim como a todos os amigos que nos acompanharam nestes anos, longe de nossa casa, na pessoa da querida Mami.

Foram tantas as pessoas que ajudaram nestes anos que seria impossível mencionar a todas elas; portanto, me limito a agradecer, em forma muito especial, a toda minha familia por ter proporcionado o ambiente de amor e felicidade que fizeram destes anos de trabalho tempos maravilhosos e inesquecíveis; mesmo aqueles fisicamente distantes, mas próximos de nós nesta maravilhosa terra brasileira,

Obrigado Brasil !

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para obtenção do grau de Doutor em Ciências (D.Sc.).

Estudo de Algoritmos Combinados Paralelos Assíncronos

Benjamín Barán Cegla

Outubro de 1993

Orientadores: Eugenius Kaszkurewicz e Djalma Mosqueira Falcão

Programa: Engenharia de Sistemas e Computação

Neste trabalho formaliza-se uma combinação paralela assíncrona de algoritmos bloco iterativos, conhecida como *'Team Algorithm'*, estudando-se a convergência num contexto parcialmente assíncrono e implementando-se várias versões num hipercubo de 8 nós, o iPSC/860 da Intel.

A utilização dos *'Team Algorithms'* para resolver sistemas de equações algébricas de grande porte viabiliza a resolução das mesmas com um máximo de eficiência, em sistemas computacionais de memória distribuída, caracterizados pelo baixo custo por instrução quando comparados com os tradicionais 'mainframes'. Também foi possível verficar que um *'Team Algorithm'* resolve problemas em que os algoritmos que o compõem não conseguem resolver quando aplicados individualmente, ou o fazem com uma eficiência menor.

A partir do conceito de continuidade *bloco-Lipschitz* formula-se uma metodologia que permite derivar condições suficientes de convergência no contexto assíncrono para vários métodos bloco iterativos como: o bloco Jacobi, o método geral das cordas paralelas, o método da solução por componente e os métodos contrativos. Esta formulação permite também a derivação de condições suficientes de convergência para combinações de diferentes métodos (ou *'Team Algorithms'*). Um *indicador de mérito* para comparações 'a priori' é postulado.

Tomando por base o problema do fluxo de potência elétrica, dois problemas exemplos foram analisados e resolvidos experimentalmente utilizando o métodos Desacoplado Rápido e a versão bloco Jacobi do método da matriz Y. Estos algoritmos foram combinados formando um *'Team Algorithm'* cujas características foram analisadas. Resultados experimentais mostram que o *'Team Algorithm'* pode apresentar desempenhos bem superiores aos métodos considerados trabalhando individualmente, encontrando inclusive a solução de problemas que os métodos considerados não resolvem isoladamente. Também foi verificado experimentalmente a vantagem das versões combinadas assíncronas em relação às versões síncronas e seqüenciais.

Abstract of the Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.).

Study of Parallel Asynchronous Team Algorithms

Benjamín Barán Cegla

1993, October

Thesis Supervisor: Eugenius Kaszkurewicz and Djalma Mosqueira Falcão

Department: System and Computer Engineering

This thesis gives a mathematical formulation of parallel asynchronous combinations of block iterative algorithms, known as *Team Algorithms*, and studies convergence in a partially asynchronous environment. Several versions of the above algorithms are implemented in an 8 nodes Intel iPSC/860 hypercube.

Using a *Team Algorithm* it is possible to solve a large system of algebraic equations, with maximum of efficiency, in a distribuited memory system characterized by its low price per instruction when compared to mainframes. These *Team Algorithms* are able to solve several problems that each individual algorithm (team member) either solves inefficiently or is unable to solve.

The concept of *block-Lipschitz continuity* is introduced and used to give a unified derivation of sufficient convergence conditions, in the asynchronous case, for several block iterative methods as: the block Jacobi, the general parallel chord method, the component solution method and for general contraction iterations. The above metodology is used as well, to derive sufficient convergence conditions for several possible combinations of different methods (or *Team Algorithms*). A figure of merit is proposed for 'a priori' comparisons of those algorithms.

Two examples based on the Load Flow problem are analysed and experimentally solved using a Fast Decoupled method and a Jacobi version of the Y matrix method. A *Team Algorithm* combining both methods is also analysed. Experimental results show that for some cases a *Team Algorithm* has a better performance than any of the individual members of the *team*; being even able to find a solution for problems that are not solved by the individual methods. Experimental data show that the asynchronous versions of the combined methods are more efficient than the synchronous and sequential ones.

Índice

Lista de Figuras

Lista de Tabelas

Símbolos e Abreviaturas

SÍMBOLO OU ABREVIATURA	SIGNIFICADO	DEFINIÇÃO ou PRIMEIRA REFERÊNCIA
$\lvert\cdot\rvert$	valor absoluto	seção 2.3
$\lVert\cdot\rVert$	norma p de uma matriz (quando p não é especificado, trata-se de uma norma genérica)	seção 2.1
$Ax = F(x)$	sistema de n equações quase lineares	seção 2.2
$A = (a_{ij})$	matriz de dimensão $n \times n$	seção 2.2
A^{-1}	inversa da matriz A	seção 2.2
A^T	transposta da matriz A	seção 2.2
A_a	aceleração devida ao assincronismo	seção 5.1
A_r	aceleração relativa de um TA	seção 5.1
A_s	aceleração seqüencial de um TA	seção 5.1
α_i	módulo do operador G_i (constante de contração)	subseção 2.2.4
B', $B"$	matrizes utilizadas no método DR	subseção 4.1.3
BJ	método bloco-Jacobi	subseção 2.2.1
C	conjunto dos números complexos	seção 2.1
c (ou c_i)	parâmetro utilizado pelo Administrador de um TA Generalizado ($c = 1 - \sum w_i$)	subseção 3.1.1
C_w	constante de Wu (do método DR)	subseção 4.1.3
C_ξ, C_x	constantes que relacionam ξ com x no método DR	subseção 4.1.3
C_{DR}	módulo do operador G_{DR} (constante contrativa)	subseção 4.1.3
CI	método das iterações contrativas	subseção 2.2.4
CS	método da solução por componente	subseção 2.2.3
DR	método Desacoplado Rápido	subseção 4.1.3
$D = D_1 \times \cdots \times D_m$	dominio de interesse ($x \in D$)	seção 2.1
D_i	subdominio de interesse ($x_i \in D_i$)	seção 2.1
D_w	dominio expandido ($W \in D_w$)	subseção 3.2.1

SÍMBOLO OU ABREVIATURA	SIGNIFICADO	DEFINIÇÃO ou PRIMEIRA REFERÊNCIA
d	medida do assincronismo (atraso máximo)	seção 2.1
$d_j^i(k)$	iteração correspondente a um subvetor x_j, utilizado no processador i na iteração k deste último	seção 2.1
ΔS	desajuste de potência ('power mismatch')	subseção 4.1.1
ΔP	desajuste de potência ativa	subseção 4.1.1
ΔQ	desajuste de potência reativa	subseção 4.1.1
$\Delta x_i(k)$	incremento de x calculado, na iteração k, pelo operador $\mathcal{G}_i$	seção 3.1
$E(x)$	resíduo ($E(x) = \\|\Phi(x)\\|$)	seção 3.1
$e = \begin{bmatrix} e_1 \\ \vdots \\ e_m \end{bmatrix}$	vetor erro, de dimensão n ($e_i = x_i - x_i^*$)	seção 2.1
ϵ	máxima variação de voltagem no dominio de interesse de um sistema elétrico	subseção 4.1.1
ε	tolerância ou erro aceitável	seção 3.1
$F(x) = \begin{bmatrix} F_1(x_1) \\ \vdots \\ F_m(x_m) \end{bmatrix}$	função ($F_i : \mathbb{R}^{n_i} \mapsto \mathbb{R}^{n_i}$) que no contexto deste trabalho é sempre considerada Lipschitz contínua	seção 2.2
$\Phi(x) = 0$	sistema de n equações algébricas	seção 2.1
$\Phi_i(x) = 0$	subsistema de n_i equações algébricas	seção 2.1
$\phi(\xi)$	operador correspondente ao método DR (Wu, 1977)	subseção 4.1.3
G	operador correspondente a um algoritmo	seção 2.1
$\mathcal{G}$	operador correspondente a um algoritmo	subseção 3.1.1
G_j	operador utilizado no processador j	seção 2.1
G_{ij}	operador que atualiza o subvetor i no processador j	subseção 3.2.1
$\widehat{G_i}$	operador G_{ii}, com $i \leq r$	seção 3.3
G_{ta}	operador correspondente a um TA operando sobre W	seção 3.3
$G_{DR}(x)$	operador correspondente ao método DR operando em coordenadas cartesianas	subseção 4.1.3

SÍMBOLO OU ABREVIATURA	SIGNIFICADO	DEFINIÇÃO ou PRIMEIRA REFERÊNCIA
γ	parâmetro de relaxação $(0 < \gamma < 1)$	seção 3.1
$H = (h_{ij})$	matriz de comparação	seção 2.1
$I(x)$	vetor das correntes elétricas como função das tensões x	subseção 4.1.1
I_j	corrente elétrica da barra j	subseção 4.1.1
I_j^*	complexo conjugado de $I_j \in \mathbf{C}$	subseção 4.1.1
J_Φ	matriz Jacobiana de $\Phi(x)$	subseção 2.2.2
L_i	módulo (ou constante contrativa) de $\mathcal{G}_i$	subseção 3.1.1
l_{F_i}	constante de Lipschitz de $F_i(x_i)$	seção 2.2
l_{ij}	constante bloco-Lipschitz do operador G_i	seção 2.1
l_{ijk}	constante bloco-Lipschitz do operador G_{ij}	seção 3.3
$\widehat{l_{ik}}$	constante bloco-Lipschitz do operador $\widehat{G_i}$	seção 3.3
m	número de blocos em que se particiona um problema	seção 2.1
$M = \text{diag}\,(M_1 \cdots M_m)$	matriz bloco-diagonal $(M_i \in \mathbb{R}^{n_i \times n_i})$	seção 2.2
$\mathbb{N}$	conjunto dos números inteiros	seção 2.1
$N = (N_{ij})$	matriz de $n \times n$ utilizada no 'splitting' da matriz A	seção 2.2
N_{ij}	submatriz de N, de dimensão $n_i \times n_j$	seção 2.2
$n = \sum n_i$	dimensão do sistema de equações algébricas	seção 2.1
n_i	dimensão do subvetor x_i	seção 2.1
$\bar{n}$	dimensão expandida	subseção 3.2.1
NR	método de Newton-Raphson	seção 2.3
η	eficiência, correspondente a S_p	seção 5.1
η_r	eficiência relativa de um TA, correspondente a A_r	seção 5.1
p	número de precessadores	subseção 3.2.1
PC	método da corda paralela	subseção 2.2.2
$\Psi = (\psi_{ij})$	matriz de atribução associada a um TA	subseção 3.2.1
P_k	potência ativa da barra k	subseção 4.1.1
Q_k	potência reativa da barra k	subseção 4.1.1

SÍMBOLO OU ABREVIATURA	SIGNIFICADO	DEFINIÇÃO ou PRIMEIRA REFERÊNCIA
q	número de barras PQ num sistema elétrico	subseção 4.1.3
r	número de subvetores de um TA Generalizado atualizados por um único processador	subseção 3.2.1
$\mathbb{R}$	conjunto dos números reais	seção 2.1
RCO	região de convergência ótima no mapa de iterações	subseção 3.1.1
R/X	relação entre a resistência e a reatância de uma rama	subseção 4.1.3
$\rho(H)$	raio espectral da matriz H	seção 2.1
σ	raio espectral da submatriz H_{11} do Exemplo 3.4	seção 3.4
$S_k = P_k + jQ_k$	potência complexa da barra k	subseção 4.1.1
SN	método de Newton simplificado	subseção 2.2.2
SOR	'Successive Over-Relaxation'	subseção 3.1.1
TA	*Team Algorithm*	capítulo 1
S_p	aceleração ou 'speedup'	seção 5.1
θ_k	fase da tensão elétrica na barra k	subseção 4.1.1
$\theta_{ki} = \theta_k - \theta_i$	diferença de fase entre as barras k e i	subseção 4.1.1
θ_{max}	máxima variação de fase no dominio de interesse de um sistema elétrico	subseção 4.1.1
$U = \text{diag}(u_1 \cdots u_m)$	matriz diagonal	seção 2.4
V_k	magnitude da tensão elétrica na barra k	subseção 4.1.1
w_i	peso utilizado pelo Administrador de um TA	seção 3.1
w_{ij}	peso (não negativo) utilizado pelo Administrador do processador j de um TA com Administrador distribuido	subseção 3.2.1
$W = \begin{bmatrix} W_1 \\ \vdots \\ W_m \end{bmatrix}$	vetor de estado expandido de um TA	subseção 3.1.3

SÍMBOLO OU ABREVIATURA	SIGNIFICADO	DEFINIÇÃO ou PRIMEIRA REFERÊNCIA
$x = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$	vetor de dimensão $n = \sum n_i$	seção 2.1
x_i	subvetor de dimensão n_i	seção 2.1
x^T	transposta do vetor x	seção 2.2
$x(k)$	vetor x na iteração k	seção 2.1
$x(0)$	condição inicial	subseção 2.2.2
$x_k = V_k e^{j\theta_k}$	tensão elétrica da barra k $(x_k \in \mathbf{C})$	subseção 4.1.1
x^*	solução do sistema de equações algébricas	seção 2.1
$x^i(k)$	vetor x disponível no processador i na iteração k	seção 2.1
$\bar{x}_i(k)$	solução local de $\Phi_i(x) = 0$, no processador i, na iteração k $(\bar{x}_i \in D_i)$	subseção 2.2.3
$\bar{x}^i(k)$	vetor x apos resolver na iteração k o subproblema local $\Phi_i(x) = 0$ no processador i	subseção 2.2.3
x_{ij}	versão do subvetor x_i calculada pelo Administrador do processador j	subseção 3.2.1
$\hat{x}^j(k)$	versão de x utilizada no processador j na iteração k, num TA Generalizado com Administrador implícito	seção 3.3
χ_i	versão de x calculado por $\mathcal{G}_i$	subseção 3.1.3
$\chi_i^3(k)$	versão de χ_i disponível no processador 3 na iteração k	subseção 3.1.3
χ_{ij}	versão do subvetor x_i calculada por G_j	subseção 3.2.1
ξ	vetor de estado do método DR	subseção 4.1.3
$Y = (y_{ki})$	matriz de admitância de um sistema elétrico	subseção 4.1.1
$y_{ki} = G_{ki} + jB_{ki}$	admitância do ramo entre as barras k e i	subseção 4.1.1
$z = \begin{bmatrix} z_1 \\ \vdots \\ z_m \end{bmatrix}$	vetor erro reduzido, de dimensão m $(z_i = \|e_i\|)$	seção 2.1

Capítulo 1

Introdução

No contexto da solução de *sistemas de equações algébricas* existem problemas para os quais são conhecidos vários métodos numéricos de solução. Para alguns casos, um dado método tem um bom desempenho, mas para outros o mesmo falha e portanto o método a ser escolhido deve ser diferente. Ocasionalmente, pode ocorrer que nenhum dos métodos disponíveis consegue resolver, individualmente, nosso problema. Porém, é possível que combinando vários métodos possa se encontrar a solução desejada.

O presente trabalho propõe técnicas para combinar e analisar diferentes métodos (ou algoritmos) num ambiente computacional paralelo-assíncrono, que provou ser altamente promissor nas primeiras experiencias computacionais, em problemas de porte médio e grande, realizadas no hipercubo iPSC/860 da INTEL ,disponível no Laboratório de Computação Paralela da Universidade Federal de Rio de Janeiro. São os chamados *Team Algorithms* (que abreviaremos por TA), que nesta década de 90 começam a ter um impulso renovado devido à disponibilidade de Sistemas Distribuídos (Paralelos) que têm arquitetura compatível para a computação dos TA. O interesse pelas combinações de métodos diferentes num contexto assíncrono pode ser percebido pelo crescente número de publicações relativas ao tema.

Neste primeiro capítulo faz-se uma breve introdução histórica dos *Team Algorithms*, enfatizando sua utilização no contexto assíncrono. Portanto, se inclui também uma breve resenha do assincronismo e suas principais características. O capítulo termina delineando os objetivos e esboçando a organização geral do presente trabalho.

1.1 Histórico dos "Team Algorithms"

A capacidade de combinar diferentes métodos para resolver problemas com o crescente grau de complexidade parece inerente à inteligência humana. Tentar descobrir quando foi a primeira vez que um homem combinou métodos diferentes para resolver um determinado problema, poderia-nos levar a um passado muito remoto. Portanto, nos

limitaremos nesta seção em citar trabalhos que combinam algoritmos computacionais diferentes e que contribuiram para uma formulação adequada à utilização em sistemas computacionais distribuídos.

Quando mencionamos os *sistemas computacionais distribuídos* nos referimos a um conjunto de vários processadores com a capacidade de se comunicar intercambiando mensagens, incluindo portanto, os computadores paralelos que possuem vários processadores e um sistema rápido e confiável de comunicação, quanto às redes de computadores físicamente distribuídas. Isto porque as características funcionais de ambos são conceitualmente similares e não existe uma clara divisão entre eles (Bertsekas e Tsitsiklis, 1989b; cap. 1.1).

Mesmo considerando somente a combinação de diferentes algoritmos iterativos num sistema seqüencial, devemos nos remeter no mínimo à década de 60. Já em Ortega e Rheinboldt (1968) temos uma resenha em que se estuda a convergência de métodos (iterativos) combinados.

Se consideramos aplicações a problemas da engenharia, devemos ressaltar o trabalho de Dusonchet et al. (1971), que propõe resolver o problema do Fluxo de Potência Elétrica utilizando o método de Jacobi para as barras de carga e o de Newton para as barras de tensão controlada, apresentando resultados experimentais significativos. Porém, o método proposto não foi posteriormente desenvolvido, provavelmente devido:

- ao excelente desempenho, no contexto seqüencial, do método Desacoplado Rápido derivado do método de Newton, que com sucessivos níveis de sofisticação, permite resolver uma imensa variedade de problemas práticos da área de sistemas de energia elétrica;
- à falta de uma tecnologia computacional amadurecida no contexto da computação distribuída, capaz de aproveitar o paralelismo intrínseco aos métodos combinados.

Mesmo sem considerar um contexto distribuído, o trabalho de Dusonchet et al. (1971), acima mencionado, foi especialmente citado por ter servido de motivação ao desenvolvimento dos *Team Algorithms*, especialmente no que se refere à idéia de particionar um problema de acordo com as características das equações envolvidas de forma a utilizar o método mais adequado para cada subconjunto de equações.

A utilização da palavra *'team'* em sistemas distribuídos já é encontrada em Marschak e Radner (1972), descrevendo sistemas com múltiplas tarefas que compartilham um mesmo objetivo. Porém, foi na década de 80 que os *Team Algorithms* começaram a ser estruturados como uma combinação de diferentes algoritmos (ou métodos) num sistema distribuído assíncrono. A primeira referência disponivel é o trabalho de Talukdar, Pyo e Giras (1982), que foi posteriormente estendido em Talukdar, Pyo e Mehrotra (1983), e que apresenta as primeiras idéias de um *Team Algorithm* como uma generalização dos trabalhos de Powell (1970), Levenberg (1944), Marquardt (1963) e de Westerberg e Director (1978).

Nos trabalhos de Talukdar et al. acima citados, a combinação de algoritmos é realizada sob a supervisão de um *Administrador* que utiliza uma memória global ou 'Blackboard', originalmente proposto por Erman et al. (1980), como meio de intercomunicação entre os diversos algoritmos que estão sendo executados pelos diferentes processadores de um sistema distribuído. São apresentados também exemplos algébricos nos quais a combinação de algoritmos (ou TA), consegue resolver problemas que nenhum dos algoritmos sendo combinados, resolve individualmente. Estes trabalhos definitivamente podem ser considerados como ponto de partida para a técnica dos *Team Algorithms* e claramente serviram de motivação ao presente trabalho, especialmente para o capítulo 3 que generaliza os TA para o caso de sistemas assíncronos de memória distribuída.

Um outro trabalho que enfatiza as vantagens da utilização de um *Team Algorithm* com um Administrador *inteligente* é o de Talukdar e Cardozo (1985). Porém, uma das formas mais atraentes dos TA é a combinação de técnicas de Inteligência Artificial com métodos tradicionais de cálculo intensivo. Em Talukdar et al. (1986) utiliza-se o modelo de 'blackboard' para combinar programas de sistemas especialistas (normalmente em OPS5 ou LISP) com programas de cálculo intensivo (normalmente em C ou Fortran) para solução de problemas distribuídos (Distributed Problem Solving ou DPS).

Outros trabalhos da década de 80 utilizam o modelo de 'blackboard', com memória compartilhada, como em Penny (1986); mas é o amadurecimento tecnológico dos sistemas de memória distribuída que motiva nesta década de 90, o crescente interesse nos TA, o que se pode notar pelo número de publicações veiculadas nos últimos anos.

Em Souza e Talukdar (1991) são apresentadas combinações de algoritmos genéticos (GA) com outros métodos tradicionais como os de Newton ou o de Levenberg-Marquardt, desta feita, denominando *A-teams* (Asynchronous teams) a estas combinações de métodos em sistemas distribuídos assíncronos, pois nesta proposta cada processador trabalha com seu próprio método, comunicando eventualmente resultados intermediários (quando disponíveis), sem nenhum tipo de sincronização.

Os *A-teams* são (novamente) discutidos em Talukdar, Ramesh e Nixon (1991). Nesta publicação é feita uma analogia entre os A-teams e uma sociedade de insetos, baseados no trabalho de Fox (1981), e se postula a possibilidade de obter um efeito sinergético; isto é, que a combinação de algoritmos resulta em bem mais que a simples *soma* das propriedades dos algoritmos combinados. Este trabalho também discute a utilização dos A-teams em sistemas de administração de energia (Energy Management Systems ou EMS) e árvores de eventos (Event Tree).

Uma discussão mais completa pode ser encontrada em Ramesh et al. (1991), na qual os A-teams são utilizados em aplicações tão diversas quanto: solução de sistemas de equações não lineares, combinação de algoritmos genéticos (GA) com algoritmos de cálculo intensivo, projeto de edifícios, controle de redes elétricas e outras aplicações em sistemas de potência.

Em Williams e Peng (1991) pode-se encontrar uma outra combinação de GA com algoritmos tradicionais de otimização (do tipo 'hill climbing') com utilização no contexto de 'Reinforcement Learning'.

Uma discussão interessante das características organizacionais dos A-teams é encontrada em Talukdar et al. (1992). Nesta publicação, baseada nas idéias de Fox (1981), se enfatiza que um A-team está constituido por um conjunto de 'entes' (ou processos) que trabalham independentemente, em forma assíncrona, e sem uma ordem pre-estabelecida de comunicação. Desta forma, a organização dos A-teams se mostra similar àquela utilizada por uma sociedade de insetos (de 'entes' simples), assim como à organização de uma comunidade de pesquisadores (de 'entes' relativamente complexos). Sendo assim, resulta promissor para os A-teams o fato de dispor de uma organização que se mostra efetiva para uma variedade de 'sociedades', formadas por 'entes' bem diferentes. Os A-teams são finalmente apresentados como uma alternativa promissora na área de sistemas de potência elétrica para tempo real, apesar de estar ainda num estágio inicial de pesquisa; porém, com resultados experimentais alentadores.

Resultados promissores utilizando A-teams são encontrados em Talukdar e Souza (1992), seja no projeto de edificios ou resolvendo o problema do caixeiro viajante (Travelling Salesman Problem ou TSP), no qual o A-team consegue superar os melhores algoritmos hoje disponíveis nos maiores problemas de solução exata conhecida (318 e 532 cidades).

Do exposto nesta seção, é natural perceber o grande interesse que estão despertando as diversas aplicações dos *Team Algorithms*, na medida em que os sistemas distribuídos (e computadores paralelos de memória distribuída) estão-se tornando cada dia mais acessiveis para organizacões comerciais e científicas do mundo todo.

1.2 Resenha Histórica do Assincronismo

A disponibilidade cada vez maior de *sistemas de computação distribuída*, nos leva naturalmente à pergunta de qual é a melhor forma de se aproveitar estes sistemas que colocam a nossa disposição um grande poder de cálculo a um baixo custo por instrução.

As primeiras propostas de implementação paralela tentaram simplesmente aproveitar os conhecimentos já adquiridos no contexto seqüencial, o que levou a formulações síncronas nas quais as tarefas intercambiam dados periodicamente nas barreiras de sincronização. Com isto, cada processador que termina a sua tarefa fica ocioso até que o último processador (o mais lento ou com mais trabalho) termine a sua parte. Evidentemente, se um processador consegue fazer um trabalho útil do invés de ficar esperando, é natural esperar que se obtenha algum ganho de eficiência.

Assim, em Rosenfeld e Driscoll (1967) e Rosenfeld (1968) aparecem as primeiras propostas do que se convencionou denominar *'Chaotic Relaxation'* e que foi formalmente analizado no aspecto da convergência em Chazam e Miranker (1969) para um algoritmo

iterativo que resolve um sistema linear de equações. Vários trabalhos tentaram ampliar estes primeiros resultados, principalmente a sistemas não lineares, como em Donnelly (1971), Miellou (1974) e Baudet (1978).

Rapidamente, as potenciais vantagens das implementações assíncronas foram reconhecidas. Discutidas em Kung (1976) e posteriormente em Bertsekas e Tsitsiklis (1988) e (1989b), elas podem ser resumidas a:

- redução das perdas de tempo por sincronização;
- facilidade de reinicialização e incorporação de novos parámetros;
- possibilidade de tolerancia a falhas, minimizando a presença de gargalos;
- aceleração na convergência dos algoritmos iterativos;
- flexibilidade e modularidade.

Na década de 80 observa-se um crescente interesse pelo assincronismo, com um grande número de trabalhos publicados, a maioria dos quais podem ser encontrados nas referências de Bertsekas e Tsitsiklis (1989b). Citar todos eles está fora do nosso escopo, pelo que nos limitaremos àqueles que influenciaram diretamente nosso trabalho. Especificamente, aqueles que fornecem modelos matemáticos ou estudam a convergência de algoritmos iterativos para a resolução do problema do ponto fixo.

Contribuções importantes utilizando a teoria de mapeamentos contrativos no estudo de convergência para o caso não linear podem ser encontrados em El Tarazi (1982), Talukdar, Pyo e Mehrotra (1983), Üresin e Dubois (1989) e em vários trabalhos de Bertsekas e Tsitsiklis (1989b). Outros estudos de convergência, utilizando funções de Liapunov podem ser encontradas em Kaszkurewicz et al. (1989) e Bhaya et al. (1991).

Neste contexto, modelos para métodos iterativos são apresentados em Bru et al. (1988), estudos considerando o resultado de pequenas desincronizações podem ser encontrados em Kleptsyn et al. (1984) e Asarin et al. (1990). O caso de dados discretos é analizado em Üresin e Dubois (1990). Um modelo estocástico pode ser encontrado em Beidas e Papavassilopoulos (1991), enquanto a convergência superlinear (Varga, 1962) é discutida em Frommer (1991). O leitor interessado pode recorrer às referências deste e de outros trabalhos citados para maiores detalhes.

Pela grande quantidade de trabalhos dedicados ao assincronismo, poderiamos pensar que este tipo de algoritmo é claramente superior a sua contrapartida síncrona, todavia isto nem sempre é verdade (Arjomandi et al., 1983). Efetivamente, uma das maiores dificuldades em se tratar os algoritmos assíncronos é a complexidade dos mesmos, o que implica a utilização de complexos modelos matemáticos que leva a uma análise complexa dos problemas. Na modelagem estudada em Bertsekas e Tsitsiklis (1989b) se distingue dois tipos de assincronismo:

- *total*, com nenhum processador parando definitivamente, antes de terminar a computação, mas permitindo atrasos de comunicação que podem tender a infinito;
- *parcial*, no qual existe uma chamada *Medida de Assincronismo* finita, que limita o atraso máximo de comunicação. Ressaltamos a existência de algoritmos que convergem com a simples existência da medida de assincronismo, enquanto outros requerem que a mesma seja suficientemente pequena, como em Barán e Corrêa (1990).

Outra complicação das implementações assíncronas, é a deteção da terminação, que requer de algoritmos especiais, como os publicados em Dijkstra e Sholten (1980), Francez (1980), Francez e Rodeh (1982), Dijkstra et al. (1983) e em Bertsekas e Tsitsiklis (1989a e 1989b). Isto porque quando o problema está distribuído entre os diversos processadores do sistema computacional, de forma que cada processador somente tem os dados do subproblema que está resolvendo, cada processador consegue verificar somente algumas equações do problema. Portanto, se necessita de algum tipo de coordenação, por causa do assincronismo, para assegurar que os diferentes processadores não estejam verificando se o sistema está 'convergido' com versões distintas (possivelmente desatualizadas) do vetor que será considerado solução aceitável do problema. Conseqüentemente, o algoritmo de terminação tem a responsabilidade de assegurar que todos os processadores verifiquem se seus respectivos subproblemas já estão resolvidos, utilizando o mesmo vetor 'solução'.

Depois de duas décadas de trabalhos principalmente de interesse teórico, chega-se a nossos dias com suficiente amadurecimento tecnológico para aproveitar o potencial dos sistemas distribuídos, e paralelos em particular, na solução de problemas computacionalmente intensivos, a custos acessíveis. Podemos esperar para os próximos anos um crescente número de aplicações práticas (IEEE Report, 1992) bem como novos resultados teóricos que permitam resolver problemas cada vez mais complexos.

1.3 Objetivos e Organização deste Trabalho

No contexto seqüencial, a combinação de diversos métodos com estruturas de dados diferentes não sempre resulta em algoritmos interessantes, pelo excessivo uso de memória em que isto implica. Porém, quando consideramos sistemas de memória distribuída, nos quais cada processador utiliza um método diferente, o fato de cada processador ter uma estrutura de dados particular, não necessariamente provoca queda significativa de eficiência. Com isto, os *Team Algorithms* se apresentam como potencialmente apropriados para os sistemas computacionais de memória distribuída. Neste sentido, enfatizamos o baixo custo por instrução dos sistemas computacionais distribuídos, quando comparados com os grandes computadores (mainframes).

Para viabilizar a solução de grandes sistemas de equações em sistemas computacionais distribuídos (e portanto a baixo custo), o problema deve ser particionado em

vários subproblemas menores e distribuído entre os diversos processadores do sistema computacional. Desta forma, cada processador utiliza o método mais adequado para o subproblema que está resolvendo, podendo os métodos diferir de um processador a outro.

O balanceamento de carga entre os diversos processadores dificilmente consegue ser bom, considerando que a partição do problema deve atender a critérios que permitam assegurar a solução do problema, alén da possibilidade de utilizar algoritmos de complexidades diferentes nos diversos processadores do sistema computacional. Isto leva a perdas de tempo muito grandes nas barreiras de sincronização, quando implementações síncronas são utilizadas. A alternativa é utilizar implementações assíncronas onde cada processador possa trabalhar a seu próprio ritmo, evitando assim os tempos ociosos devido ao fato de que outros processadores utilizam mais tempo por iteração.

Dadas as considerações acima, o principal objetivo deste trabalho é formalizar os *bloco Team Algorithms* num contexto assíncrono, de forma a viabilizar a solução de sistemas de equações algébricas de grande porte, em sistemas computacionais distribuídos, utilizando diferentes métodos na solução de cada um dos subproblemas. Portanto, a primeira tarefa é postular formas adequadas à combinação de métodos diferentes num sistema distribuído, com o claro objetivo de resover o problema global.

Uma vez propostos os mecanismos que permitem combinar os diferentes métodos e formalizado os bloco TA, é preciso poder estudar a convergência das combinações possíveis, de forma a escolher as melhores opções. Este foi talvez o maior desafio deste trabalho, considerando a dificuldade de se analisar sistemas assíncronos, mesmo quando todos os processadores utilizam o mesmo algoritmo.

Ainda são escassos os resultados experimentais em sistemas assíncronos. Mais especificamente, não temos conhecimento de nenhuma implementação assíncrona que permita resolver sistemas algébricos de grande porte, combinando métodos diferentes, num sistema distribuído. Sendo assim, torna-se fundamental a implementação dos bloco TA aqui postulados, de forma a verificar experimentalmente as propriedades dos mesmos. Com este objetivo foram implementadas diversas versões de *Team Algorithms* no hipercubo de 8 nós, o iPSC/860 da INTEL, disponível no Laboratório de Computação Paralela da UFRJ. Com isto, foi possível obter conhecimentos importantes sobre os Team Algorithms, bem como das virtudes e dificuldades próprias de uma implementação assíncrona em computadores paralelos.

Enfatizamos aquí que as primeiras publicações em que se discutem os *Team Algorithms* em sistemas assíncronos de memória distribuída, apareceram somente há uns poucos anos atrás, ainda nesta década de 90; portanto, é uma área muito nova de trabalho, e em certo sentido, ainda inexplorada. Conseqüentemente, espera-se com este trabalho chamar a atenção sobre a potencialidade dos *Team Algorithms* para diversas áreas da engenharia, apresentando resultados experimentais, que confirmem as expectativas positivas que se tem dos TA. Finalmente, espera-se sugerir algumas linhas de pesquisa e destacar potenciais aplicações, que permitam continuar com os trabalhos

até aquí realizados, já que o presente trabalho não pretende ser definitivo, mas simplesmente, o primeiro em formalizar os *Team Algorithms*, estudando a convergência dos mesmos e apresentando resultados experimentais de implementações assíncronas de médio e grande porte.

O presente trabalho foi dividido em 6 capítulos. O capítulo 2 estuda, num contexto assíncrono, os algoritmos bloco iterativos, que resolvem um problema de grande porte, particionando o problema em subproblemas menores, cada um dos quais pode ser resolvido num outro processador. Se introduz também o modelo matemático a ser utilizado, o que permite analisar a convergência de diversos métodos. A versão mais simples dos *bloco Team Algorithms* é apresentada, derivando-se também uma condição suficiente de convergência para o mesmo. O capítulo conclui postulando um *Indicador de Mérito* que permite comparar os diferentes métodos em termos de convergência.

A formalização matemática dos *Team Algorithms* é realizada no capítulo 3 que postula várias técnicas para combinar diferentes algoritmos no contexto assíncrono. A convergência é analisada utilizando um modelo generalizado que inclui como casos particulares a maior parte das propostas de TA utilizadas na solução de problemas algébricos, incluindo o uso de um *Administrador*, potencialmente *'inteligente'*. As diversas formas de combinação apresentadas, são finalmente comparadas, enfatizando-se as circunstâncias nas quais cada uma apresenta vantagens específicas.

No capítulo 4 analisa-se um problema exemplo, discutindo a implementação assíncrona de um *Team Algorithm* no hipercubo de 8 nós do Laboratório de Computação Paralela da UFRJ, o iPSC/860 da INTEL. O problema exemplo escolhido é o *Problema do Fluxo de Potência Elétrica* (Stott, 1974) para um sistema de 616 barras. Um primeiro caso do problema exemplo, permite verificar experimentalmente as vantagens do TA sem Administrador, junto às características da implementação. Um segundo caso, no qual o uso de um *Administrador* é indispensável para encontrar a solução do problema, é também apresentado, junto às diversas implementações possíveis. A análise dos resultados experimentais destas implementações é apresentada no capítulo 5, que enfatiza as principais características dos *Team Algorithms.*

As conclusões finais do trabalho são apresentadas no capítulo 6, no qual se resume as principais contribuições do trabalho e se sugere tópicos para futuras pesquisas.

Sumário do Capítulo 1

Neste capítulo foi introduzido a noção dos *Team Algorithms* como uma técnica que permite combinar métodos diferentes na solução de sistemas de equações algébricas, enfatizando a sua utilização nos sistemas distribuídos assíncronos.

Primeiramente, foi apresentado um histórico dos TA que permitiu expor as idéias fundamentais dos mesmos e ilustrar algumas aplicações; evidenciando o grande interesse que os *Team Algorithms* estão despertando na década de 90, na medida em que sistemas distribuídos estão se tornando mais acessíveis. Assim, foram enfatizadas as vantagens de se utilizar os TA num contexto assíncrono, de forma a aproveitar o baixo custo por instrução dos sistemas computacionais de memória distribuída, especialmente em presença de desbalanceamentos de carga computacional. Uma breve resenha histórica foi também dada, de modo a colocar em perspectiva as principais características do assíncronismo.

Os objetivos principais deste trabalho foram expostos, enfatizando-se o interesse em resolver sistemas de equações algébricas de grande porte utilizando métodos possivelmente diferentes nos distintos processadores de um sistema computacional de memória distribuída. Para isto, o trabalho pretende formalizar os *Team Algorithms*, estudar a sua convergência e implementar estas idéias de forma a avaliar os resultados experimentais que permitam confirmar as expectativas positivas que se tem dos TA.

Capítulo 2

Algoritmos Bloco Iterativos Assíncronos

Neste capítulo formalizam-se os algoritmos bloco iterativos num contexto assíncrono, estudando-se também a convergência dos mesmos. A proposta consiste em particionar um problema complexo, em vários subproblemas menores e distribuir estes subproblemas entre os diversos processadores do sistema computacional, de forma que cada processador compute exclusivamente o subproblema específico a ele alocado.

Quando dizemos que um problema é complexo, o fazemos no sentido discutido em Šiljak (1981), trabalho este que distingue três fatores para determinar a complexidade de um sistema dinámico: dimensão, ambiguidade e estruturação do problema.

O objetivo da implementação assíncrona a ser estudada, é encontrar um 'vetor solução' que satisfaça (possivelmente com uma dada tolerância) o sistema de equações algébricas dado. Para isto, é alocado a cada processador um subproblema junto com as componentes do vetor solução que serão atualizadas por esse processador. Na proposta deste capítulo, cada componente do vetor solução é atualizada por um e somente um processador.

Assim, no contexto assíncrono considerado, cada processador *tenta resolver* o subproblema a ele alocado, utilizando um algoritmo adequado que permita atualizar as componentes do vetor solução correspondentes a esse processador. Como o subproblema a ser resolvido num processador, pode depender também das componentes do vetor não atualizadas por ele, cada processador inclui no cálculo, quando necessarios, os dados mais atualizados que ele possui das componentes atualizadas nos outros processadores. Comunicações assíncronas (sem bloqueio) entre processadores são encorajadas, de forma que resultados parciais, ou melhores estimativas da solução, possam ser conhecidos e incluidos nos cálculos de todos os processadores.

Note que o conjunto avança para a solução global do problema de uma forma tipicamente iterativa. Isto porque uma vez calculados novos valores para as componentes

atualizadas por um dado processador, mesmo que estas resolvam o subproblema a ele alocado (solução local), estas devem ser comunicadas aos outros processadores do sistema, para verificar que a solução global para todo o sistema já foi alcançada. De não ser assim, a computação deve continuar e o processador considerado (que já encontrou uma solução local) pode eventualmente receber novos valores das outras componentes (calculadas em outros processadores) que façam com que o subproblema a ele alocado não continue resolvido. Novas iterações são assim necessarias neste processador.

Enfatizamos que para muitos problemas de engenharia, os métodos iterativos tem a vantagem de poder incluir na computação, conhecimentos *'a priori'* do engenheiro ou especialista na estimativa da condição inicial, o que pode resultar numa aceleração do processo. O método trabalha bem somente em certas regiões, que normalmente são conhecidas. Um bom exemplo neste sentido que será discutido no capítulo 4 é o problema do Fluxo de Potência Elétrica (ver Stott, 1974; Monticelli, 1983).

O esquema de solução proposto, que denominamos *algoritmo bloco iterativo*, não especifica o algoritmo particular a ser utilizado em cada processador. Assim, se usamos o método de Jacobi em cada processador, obtemos a versão bloco assíncrona do método de Jacobi (o simplesmente bloco Jacobi). Analogamente, se cada processador utiliza o método de Newton para 'tentar resolver' o subproblema a ele alocado, obtemos a versão bloco assíncrona do método de Newton. Da mesma forma, é possivel ter uma versão bloco assíncrona para cada método conhecido no contexto seqüencial. Contudo, neste trabalho estamos mais interessados na combinação de algoritmos diferentes nos diversos processadores do sistema computacional distribuído. Assim, cada processador pode escolher o método mais apropriado para resolver o subproblema a ele alocado, independentemente da escolha dos outros processadores. O algoritmo bloco iterativo assim constituído é denominado genericamente *bloco Team Algorithm* (ou bloco TA).

Enfatizamos aqui a simplicidade dos bloco TA acima introduzidos, pois nenhum *Administrador* é utilizado nesta proposta, que pode ser entendida como uma simples partição de um problema de grande porte em subproblemas menores, potencialmente mais simples de ser resolvidos; de forma que cada um destes subproblemas possa ser resolvido pelo método mais adequado, num outro processador.

Os *bloco Team Algorithms* assim considerados, se mostram altamente promissores para sistemas de grande porte, constituídos por subsistemas (ou blocos) levemente acoplados, com características bem diferenciadas que fazem com que nenhum algoritmo individualmente possa resolver, igualmente bem, todos os subproblemas. A escolha de algoritmos diferentes para os distintos subsistemas, dependendo de suas características específicas, se mostra assim a mais natural das propostas.

A motivação do estudo realizado neste capítulo vem de uma proposta seqüencial aplicada à solução de Sistemas de Potência no trabalho de Dusonchet et al. (1971) para resolver o problema de Fluxo de Potência Elétrica. Como já foi citado, esta primeira proposta propõe particionar o problema em dois subproblemas. Um com as barras de carga, que seria resolvido por uma variante do método de Jacobi, e o outro subproblema com as barras de tensão controlada que seria resolvido pelo método de Newton.

Esta primeira proposta que não prosperou no contexto seqüencial apesar dos primeiros resultados alentadores, tem uma nova perspectiva nos sistemas distribuídos, o que será discutido neste capítulo, generalizando as idéias básicas da proposta seqëncial para um problema algébrico mais geral, e formalizando a análise de algoritmos bloco iterativos num sistema computacional distribuído. Para isto, a seção 2.1 apresenta a formulação matemática do problema, assim como a formalização dos algoritmos bloco iterativos, derivando-se uma condição suficiente de convergência para os mesmos. A seção 2.2 discute a convergência de versões bloco assíncronas de vários métodos clássicos, especialmente quando utilizados na solução de sistemas quase lineares de equações. Os *bloco TA* são formalizados na seção 2.3. Finalmente, uma figura de mérito para comparações *'a priori'* de diferentes métodos bloco iterativos, é proposta na seção 2.4.

2.1 Formulação Matemática

Nesta seção consideramos um *sistema de n equações algébricas* dado por:

$$\Phi(x) = 0, \qquad x \in \mathbb{R}^n, \qquad \Phi : \mathbb{R}^n \to \mathbb{R}^n \tag{2.1}$$

definido num domínio $D \subset \mathbb{R}^n$. O objetivo é resolver este sistema de equações utilizando um *sistema distribuído* com m processadores, onde $m \leq n$ (sendo que para a maioria das aplicações $m \ll n$). Para isto, o problema é particionado, de forma que cada processador *'resolve'* somente uma parte (o subproblema) do sistema completo, comunicando os resultados parciais aos outros processadores para finalmente resolver o problema global dado por (2.1). Para formalizar estas idéias, introduz-se abaixo a notação e algumas definições a serem utilizadas.

Seja a *Decomposição Cartesiana* de $\mathbb{R}^n$ dada por[1]:

$$\mathbb{R}^n = \mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_m}, \qquad n_1 + \cdots + n_m = n. \tag{2.2}$$

e ademais $D \subset \mathbb{R}^n$ um domínio tal que

$$D = D_1 \times \cdots \times D_m, \qquad D_i \subset \mathbb{R}^{n_i}, \quad \forall\, i \in \{1, \ldots, m\} \tag{2.3}$$

Um vetor $x \in D$ pode ser particionado de acordo com:

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}, \qquad x_i \in D_i, \qquad \forall\, i \in \{1, \ldots, m\} \tag{2.4}$$

enquanto $\Phi(x)$ como:

$$\Phi(x) = \begin{bmatrix} \Phi_1(x) \\ \vdots \\ \Phi_m(x) \end{bmatrix}, \qquad \Phi_i : D \to \mathbb{R}^{n_i} \tag{2.5}$$

[1]Por simplicidade faremos a formalização dos algoritmos bloco iterativos no conjunto $\mathbb{R}$ dos reais, enfatizando que todos os resultados são facilmente estendidos ao conjunto $\mathbf{C}$ dos complexos.

Conseqüentemente, a equação a ser resolvida em cada processador i, que será chamada de *problema local*, é dada por:

$$\Phi_i(x) = 0, \qquad \forall\, i \in \{1, \ldots, m\} \tag{2.6}$$

Para resolver a equação (2.1), ou de forma equivalente (2.6), assumimos neste trabalho, que será utilizado um algoritmo iterativo, da forma

$$x(k+1) \;=\; G\big(x(k)\big)$$

O operador G que representa o algoritmo considerado tem um ponto fixo $x^* = G(x^*)$ que é assumido ser a solução de (2.1), i.e. $\Phi(x^*) = 0$. Assim, considerando a equação (2.6), o operador G pode ser escrito como:

$$G(x) = \begin{bmatrix} G_1(x) \\ \vdots \\ G_m(x) \end{bmatrix}, \qquad G_i(x) : D \to D_i, \quad \forall\, i \in \{1, \ldots, m\} \tag{2.7}$$

onde (2.7) é uma *partição conforme* associada às equações (2.4) e (2.5). Desta forma, cada processador i resolve sua equação *local* (2.6) usando um operador G_i que atualiza x_i, a partir do valor de x disponível nesse processador, transmitindo eventualmente o valor atualizado de x_i aos outros processadores; assim, uma iteração no processador i pode ser representada por:

$$x_i \leftarrow G_i(x), \qquad \forall\, i \in \{1, \ldots, m\} \tag{2.8}$$

onde $\leftarrow$ representa alocação do valor do segundo membro para a variável do primeiro membro (mais detalhes na Nota 2.1).

Ao longo de todo este trabalho, se fará a seguinte hipótese que formaliza o acima mencionado:

HIPÓTESE 2.1 (UNICIDADE DA SOLUÇÃO)
Dado o sistema de equações (2.1), que pode ser re-escrito como (2.6), definido num conjunto fechado $D \subset \mathbb{R}^n$ que satisfaz (2.3), existe um operador G da forma (2.7) e que opera segundo (2.8), tal que $G(D) \subset D$ e adicionalmente D apresenta um e somente um ponto fixo

$$x^* = \begin{bmatrix} x_1^* \\ \vdots \\ x_m^* \end{bmatrix}, \qquad x_i^* \in D_i, \qquad \forall\, i \in \{1, \ldots, m\}$$

do operador G. O ponto fixo x^* é por sua vez solução de (2.1). Conseqüentemente, podemos escrever que:

$$x_i^* = G_i(x^*), \qquad \forall i \in \{1,\ldots,m\} \tag{2.9}$$

e também

$$\Phi_i(x^*) = 0, \qquad \forall i \in \{1,\ldots,m\} \tag{2.10}$$

Enfatizamos aqui nosso interesse nas implementações assíncronas, cuja modelagem matemática implica na possibilidade de usar valores atrasados das variáveis (Kaszkurewicz et al., 1990), nas atualizações (ou iterações) representadas por (2.8). Especificamente, assumiremos que no momento da iteração k, o processador i que atualiza x_i, recebeu informação de um outro processador j, com um atraso variante no tempo, de $[k - d_j^i(k)]$ unidades. No presente trabalho faremos a hipótese de que o sistema é *parcialmente assíncrono* (ver Bertsekas e Tsitsiklis, 1989b), o que implica que todos os atrasos entre os diversos processadores são limitados uniformemente por um número inteiro e positivo d denominado *medida de assincronismo.* Isto é formalizado pela hipótese que segue:

HIPÓTESE 2.2 (ASSINCRONISMO PARCIAL)

$$\begin{array}{l} \exists\, d \in \mathbb{N}, \qquad \forall k \in \mathbb{N}, \qquad \forall i,j \in \{1,\ldots,m\}, \\ \quad \textit{tal que} \quad d_j^i(k) \in \{k, k-1, \ldots, k-d\}. \end{array} \tag{2.11}$$

Denotamos como $x^i(k)$ ao vetor x disponível no processador i, no momento da iteração k, i.e.

$$x^i(k) \stackrel{\text{def}}{=} \begin{bmatrix} x_1\big(d_1^i(k)\big) \\ \vdots \\ x_m\big(d_m^i(k)\big) \end{bmatrix}, \qquad \forall i \in \{1,\ldots,m\} \tag{2.12}$$

Usando esta notação podemos escrever o *algoritmo (bloco iterativo) assíncrono* baseado em (2.8) na forma:

$$x_i(k+1) \;=\; G_i\big(x^i(k)\big), \qquad \forall i \in \{1,\ldots,m\} \tag{2.13}$$

O algoritmo (bloco iterativo) síncrono, baseado em (2.8), pode ser simplesmente escrito como:

$$x_i(k+1) \;=\; G_i\big(x(k)\big) \qquad \forall i \in \{1,\ldots,m\} \tag{2.14}$$

impondo a condição $d_j^i(k) = k$, o que é feito, em termos de implementação, por meio de barreiras de sincronização.

A equação (2.13) dá a forma geral de um algoritmo bloco iterativo assíncrono, implementado da seguinte forma: cada processador i tenta resolver seu *subproblema*

local (2.6) atualizando x_i a cada iteração k, aplicando para isto um operador G_i sobre $x^i(k)$, o valor mais recente de x a ele disponivel (note que $x_j \; \forall j \neq i$ é constante numa dada iteração, pois o processador i atualiza somente x_i). Valores atualizados de x_i são comunicados aos outros processadores sem bloqueios ou interrupções, quando disponíveis. O processo continua iterando até que uma solução com dada tolerância seja encontrada.

Nota 2.1 A equação (2.13) pode também ser escrita como:

$$x_i(k+1) \;\leftarrow\; G_i\Big(x^i(k)\Big), \qquad\qquad \forall\, i \in \{1,\ldots,m\}$$

mas se prefere enfatizar a igualdade matemática quando a iteração k está especificada. Contudo, não se utiliza o signo $=$ em (2.8) por não ser especificada a iteração correspondente.

Considerando nosso interesse nos métodos bloco iterativos, introduzimos um conceito que possibilita a formulação matemática dos mesmos, assim como o correspondente estudo de convergência num contexto assíncrono. Sendo assim, definimos a continuação o conceito de *continuidade bloco-Lipschitz*, que é equivalente ao conceito de continuidade de Lipschitz (ver Lema 2.1).

DEFINIÇÃO 2.1 (CONTINUIDADE BLOCO-LIPSCHITZ)
Uma funcão $G(x)$ com a característica dada por (2.7) é denominada bloco-Lipschitz contínua (em uma dada norma $\|\cdot\|$) num domínio D, com referência à decomposição cartesiana (2.3), se $\forall x, y \in D$ satisfazendo a partição (2.4) é possível escrever a relação:

$$\|G(x) - G(y)\| \;\leq\; \sum_{i=1}^{m} L_i \|x_i - y_i\| \tag{2.15}$$

onde os escalares L_i serão denominados 'constantes bloco-Lipschitz'.

A utilização do conceito de continuidade bloco-Lipschitz em nossa formalização dos métodos bloco iterativo, não impõe nenhuma condição especialmente restritiva, dada sua equivalência com o conhecido conceito de continuidade de Lipschitz, como se demonstra no lema que segue:

LEMA 2.1 (FUNÇÃO BLOCO-LIPSCHITZ CONTÍNUA)
Uma função $G(x)$ é bloco-Lipschitz contínua se e somente se ela é Lipschitz contínua.

Prova.
Apresenta-se uma prova em norma-1 usando a relação:

$$\|x\|_1 = \sum_{i=1}^{m} \|x_i\|_1 \tag{2.16}$$

pois a generalização para uma outra norma-p é facilmente obtida a partir da relação:

$$c_1\|x\|_p \leq \|x\|_1 \leq c_2\|x\|_p \tag{2.17}$$

Parte 1:

Considerando primeiramente que G é Lipschitz contínua em norma-1 com constante de Lipschitz L:

$$\|G(x) - G(y)\|_1 \leq L\|x - y\|_1 = L\sum_{i=1}^{m}\|x_i - y_i\|_1$$

portanto, G é bloco-Lipschitz contínua com constantes $L_i = L$ em norma-1. Assim, por (2.17) obtemos:

$$c_1\|G(x) - G(y)\|_p \leq \|G(x) - G(y)\|_1 \leq L\sum_{i=1}^{m}\|x_i - y_i\|_1 \leq c_2 L\sum_{i=1}^{m}\|x_i - y_i\|_p$$

conseqüentemente, G é bloco-Lipschitz contínua com constantes $L_i = c_2L/c_1$ em norma-p.

Parte 2:

Assumindo que G é bloco-Lipschitz contínua em norma-1 com constantes L_i e denotando como L_{max} o valor máximo destas constantes:

$$\|G(x) - G(y)\|_1 \leq \sum_{i=1}^{m} L_i\|x_i - y_i\|_1 \leq L_{max}\sum_{i=1}^{m}\|x_i - y_i\|_1 = L_{max}\|x - y\|_1$$

portanto, G é Lipschitz contínua em norma-1, com constante $L = L_{max}$. Assim, por (2.17) obtemos:

$$c_1\|G(x) - G(y)\|_p \leq \|G(x) - G(y)\|_1 \leq L_{max}\|x - y\|_1 \leq c_2 L_{max}\|x - y\|_p$$

conseqüentemente, G é Lipschitz contínua em norma-p, com constante $L = c_2L_{max}/c_1$.

■

Exemplo 2.1 Seja $F(x) : D \to D \subset \mathbb{R}^2$ com $x \in \mathbb{R}^2$ onde $D = \{\, x \mid \|x\|_\infty \leq 1\}$

$$F(x) = \begin{bmatrix} F_1(x_1, x_2) \\ F_2(x_1, x_2) \end{bmatrix} = \begin{bmatrix} x_2(0.5x_1^2 + 0.3) \\ 0.4x_1^2 + 0.2x_2^2 \end{bmatrix}$$

$F(x)$ satisfaz em D a relação:

$$\|F(x) - F(y)\|_\infty \leq 1.8\|x - y\|_\infty, \qquad \text{de onde } L = 1.8 \tag{2.18}$$

e

$$\|F(x) - F(y)\|_\infty \leq \|x_1 - y_1\|_\infty + 0.8\|x_2 - y_2\|_\infty \tag{2.19}$$

de onde $L_1 = 1.0$ e $L_2 = 0.8$. Note que a relação $L \geq L_i$ é sempre satisfeita.

Para o presente trabalho assumimos que uma mesma norma $\|\cdot\|$ é usada em todos os espaços $\mathbb{R}^{n_i}$, $i = 1, \ldots, m$, assim como em $\mathbb{R}^n$, embora isto não seja uma condição imprescindível. Fazemos também a seguinte Hipótese:

HIPÓTESE 2.3 (BLOCO-LIPSCHITZ)
Cada operador $G_i(x)$ da equação (2.13) é bloco-Lipschitz contínuo em D, i.e.

$$\|G_i(x) - G_i(y)\| \leq \sum_{j=1}^{m} l_{ij}\|x_j - y_j\| \qquad \forall x, y \in D, \quad \forall\, i \in \{1, \ldots, m\} \tag{2.20}$$

Nota 2.2 As constantes bloco-Lipschitz (l_{ij}) do operador G dependem do algoritmo escolhido e claramente estão relacionadas com as características de $\Phi(x)$.

Com o objetivo de derivar uma condição suficiente de convergência para o algoritmo assíncrono (2.13), definimos o *vetor erro* $e = [e_1^T, \cdots, e_m^T]^T \in \mathbb{R}^n$ como:

$$\forall\, i, j \in \{1, \ldots, m\}, \qquad \begin{cases} e_i(k+1) & \stackrel{\text{def}}{=} & x_i(k+1) - x_i^* \\ e_j\left(d_j^i(k)\right) & \stackrel{\text{def}}{=} & x_j\left(d_j^i(k)\right) - x_j^* \end{cases} \tag{2.21}$$

e o *vetor erro reduzido* $z = [z_1, \cdots, z_m]^T \in \mathbb{R}^m$ como:

$$\forall\, i, j \in \{1, \ldots, m\}, \qquad \begin{cases} z_i(k+1) & \stackrel{\text{def}}{=} & \| e_i(k+1) \| \\ z_i\left(d_j^i(k)\right) & \stackrel{\text{def}}{=} & \| e_i\left(d_j^i(k)\right) \| \end{cases} \tag{2.22}$$

Subtraindo (2.9) de (2.13) obtemos $\forall\, i \in \{1, \ldots, m\}$:

$$e_i(k+1) = x_i(k+1) - x_i^* = G_i\left(x^i(k)\right) - G_i(x^*) \tag{2.23}$$

tomando normas e usando a hipótese 2.3:

$$\|e_i(k+1)\| = \|G_i\left(x^i(k)\right) - G_i\left(x^*\right)\| \leq \sum_{j=1}^{m} l_{ij}\|x_j\left(d_j^i(k)\right) - x_j^*\| \tag{2.24}$$

assim,

$$\|e_i(k+1)\| \leq \sum_{j=1}^{m} l_{ij}\|e_j\left(d_j^i(k)\right)\|, \qquad \forall\, i \in \{1, \ldots, m\} \tag{2.25}$$

que pela definição de z pode ser escrito como:

$$z_i(k+1) \leq \sum_{j=1}^{m} l_{ij}\, z_j\left(d_j^i(k)\right), \qquad \forall\, i \in \{1, \ldots, m\} \tag{2.26}$$

Para demonstrar a convergência do algoritmo assíncrono (2.13), é suficiente demonstrar que o vetor erro reduzido z que satisfaz a desigualdade (2.26) tende a zero quando $k \to \infty$. Para isto, usaremos o *lema assíncrono de comparação* derivado no Apêndice B, a partir do trabalho de Bhaya et al. (1991).

LEMA 2.2 (LEMA ASSÍNCRONO DE COMPARAÇÃO)
Sob as hipóteses 2.1 e 2.2, dada uma matriz não negativa $H = (h_{ij}) \geq 0$ *e um vetor variante no tempo e não negativo* $z \geq 0$, *satisfazendo a inequação:*

$$z_i(k+1) \leq \sum_{j=1}^{m} h_{ij}\, z_j\Big(d_j^i(k)\Big), \qquad \forall\, i \in \{1,\ldots,m\} \tag{2.27}$$

então, $\rho(H) < 1$ *é uma condição suficiente para que* $z(k)$ *tenda exponencialmente a zero quando* $k \to \infty$, *i.e.*

$$\lim_{k\to\infty} z(k) = 0 \qquad se \qquad \rho(H) < 1$$

onde ρ *denota o raio espectral.*

O resultado imediato da aplicação do Lema 2.2 à equação (2.26), tem como conseqüência o teorema que estabelece uma condição suficiente de convergência para o algoritmo assíncrono dado por (2.13).

TEOREMA 2.1 (CONDIÇÃO SUFICIENTE DE CONVERGÊNCIA)
Sob as hipóteses 2.1 a 2.3, o algoritmo assíncrono representado por

$$x_i(k+1) \;=\; G_i\Big(x^i(k)\Big), \qquad \forall\, i \in \{1,\ldots,m\}$$

converge para o único ponto fixo x^* *em* D *se*

$$\rho(H) < 1$$

onde H *é dado por:*

$$H \;=\; (h_{ij}) \;\in\; \mathbb{R}^{m\times m}, \qquad com\;\; h_{ij} = l_{ij}$$

A condição suficiente de convergência dada acima é especialmente conveniente no tratamento de algoritmos combinados, uma vez que cada operador G_i pode representar um método ou algoritmo distinto, como será visto nas próximas seções.

Nota 2.3 Para matrizes não negativas H, existem várias condições equivalentes a $\rho(H) < 1$, algumas das quais são bem mais fácil de se verificar na hora de aplicar o Teorema 2.1 (ver Lema A2 do Apêndice A, originalmente demonstrado em Kaszkurewicz et al., 1990).

Para facilitar referências futuras, introduzimos a continuação a definição de *Matriz de Comparação.*

DEFINIÇÃO 2.2 (MATRIZ DE COMPARAÇÃO)
Dada a inequação:

$$z_i(k+1) \leq \sum_{j=1}^{m} h_{ij}\, z_j\big(d_j^i(k)\big), \qquad \forall\, i \in \{1,\ldots,m\}$$

do Lema 2.2, a matriz não negativa $H = (h_{ij})$ *é denominada Matriz de Comparação.*

Enfatizamos que a *matriz de comparação* H é uma matriz agregada (Kaszkurewicz et al., 1990) à equação (2.13) com dimensão $(m \times m)$, menor que a dimensão $(n \times n)$ do sistema. Como exemplo, se resolvemos um sistema algébrico com $n = 1000$ equações, utilizando um método bloco iterativo que distribui o problema num hipercubo de 8 nós (como aquele utilizado para as implementações desta tese), a matriz de comparação teria uma dimensão de (8×8).

Para finalizar esta seção, comparamos o Teorema 2.1 acima derivado, com um teorema derivado por Talukdar et al. (1983), que também pode ser utilizado para estabelecer uma condição suficiente de convergência dos algoritmos bloco iterativos, num contexto assíncrono. Assim, podemos enunciar o seguinte teorema ('Theorem 4-2', Talukdar et al., 1983):

TEOREMA 2.2 (CONVERGÊNCIA DE ALGORITMOS ASSÍNCRONOS)
Sob as hipóteses 2.1 e 2.2, o algoritmo assíncrono representado por um operadar $G(x)$ *converge para a única solução* x^* *em* D *, se* $G(x)$ *é um operador contrativo em norma infinita no domínio considerado, i.e.*

$$\forall x, y \in D, \qquad \|G(x) - G(y)\|_\infty \leq L_\infty \|x - y\|_\infty \qquad \text{onde} \quad 0 \leq L_\infty < 1.$$

Para uma comparação significativa dos Teoremas 2.1 e 2.2, temos que partir das mesmas hipóteses. Portanto, assumimos verdadeira a Hipótese 2.3 e derivamos a condição suficiente de convergência que resulta de aplicar o Teorema 2.2:

$$\forall x, y \in D, \qquad \|G(x) - G(y)\|_\infty = \max_{i \in \{1,\ldots,m\}} \Big(\|G_i(x) - G_i(y)\|_\infty \Big)$$

e utilizando a Hipótese 2.3 obtemos

$$\begin{aligned}
\forall x, y \in D, \qquad \| G(x) - G(y) \|_\infty &\leq \max_{i \in \{1,\ldots,m\}} \left(\sum_{j=1}^{m} l_{ij}\, \|x_i - y_i\|_\infty \right) \\
&\leq \max_{i \in \{1,\ldots,m\}} \left(\sum_{j=1}^{m} l_{ij} \right) \max_{i \in \{1,\ldots,m\}} \Big(\|x_i - y_i\|_\infty \Big) \\
&\leq \max_{i \in \{1,\ldots,m\}} \left(\sum_{j=1}^{m} l_{ij} \right) \|x - y\|_\infty
\end{aligned}$$

que considerando a matriz de comparação $H = (h_{ij})$ com $h_{ij} = l_{ij}$ dada no Teorema 2.1, pode ser escrita como:

$$\forall x, y \in D, \qquad \|G(x) - G(y)\|_\infty \leq \|H\|_\infty \|x - y\|_\infty$$

conseqüentemente, $L_\infty = \|H\|_\infty$. Portanto, a condição suficiente de convergência de acordo com o Teorema 2.2 implica que: $\|H\|_\infty < 1$.

Lembrando que $\rho(H) \leq \|H\|_\infty$ (Varga, 1969), concluimos que a condição suficiente de convergência dada pelo Teorema 2.1 é menos restritiva que a condição derivada do Teorema 2.2, pois:

- sob as hipóteses 2.1 a 2.3, sempre que o Teorema 2.2 é aplicável, o Teorema 2.1 também o é. Além disso,
- pode existir um algoritmo bloco iterativo para o qual:

  $$\rho(H) < 1 \leq \|H\|_\infty$$

  e portanto, somente o Teorema 2.1 consegue assegurar a convergência num contexto assíncrono.

Conseqüentemente, como o Teorema 2.1 é menos restritivo, será ele o utilizado ao longo deste trabalho.

2.2 Versões Assíncronas de Algoritmos Clássicos

O objetivo desta seção é utilizar a formulação matemática e os resultados da seção anterior de forma a derivar condições suficientes de convergência, para uma variedade de algoritmos clássicos, quando implementados num contexto assíncrono como algoritmos bloco iterativos. Os resultados assim obtidos poderão ser utilizados no estudo de convergência dos bloco TA, que serão analisados na próxima seção.

De uma forma geral, o estudo de convergência de algoritmos conhecidos, num contexto seqüencial, pode ser ampliado para o contexto assíncrono dos algoritmos bloco iterativos; mas estudar todos os casos está fora do escopo deste trabalho. Porém, nesta seção apresenta-se uma metodologia geral de análise aplicada a vários algoritmos clássicos como o bloco Jacobi, o Newton simplificado, e o método das iterações contrativas entre outros. Como ilustração, com cada método se utilizam nossos resultados na resolução de *equações quase lineares* (analisadas em Ortega e Rheinboldt, 1970).

A escolha das equações quase lineares, como problema exemplo desta seção, foi motivada no trabalho de Bhaya et al. (1991); no qual, este tipo de problema é analisado no contexto assíncrono dos algoritmos iterativos tipo bloco Jacobi. Sendo assim, utilizaremos basicamente a formulação desse trabalho que considera o sistema de *equações quase lineares* da forma:

$$Ax = F(x), \qquad A \in \mathbb{R}^{n \times n}, \qquad F : \mathbb{R}^n \to \mathbb{R}^n \tag{2.28}$$

para o caso especial em que F satisfaz as seguintes hipóteses:

HIPÓTESE 2.4 (NÃO LINEARIDADE BLOCO-DIAGONAL)

$$\begin{aligned} &F_i : D_i \rightarrow F_i(D_i) : x_i \mapsto F_i(x_i), \qquad \forall i \in \{1,\ldots,m\} \\ &F : (x_1^T,\ldots,x_m^T)^T \mapsto (\, F_1(x_1)^T,\ldots,F_m(x_m)^T\,)^T \end{aligned} \tag{2.29}$$

e também que cada $F_i(x_i)$ é Lipschitz-contínua com constante l_{F_i}, i.e.

HIPÓTESE 2.5 (CONTINUIDADE DE LIPSCHITZ)

$$||F_i(x_i) - F_i(y_i)|| \leq l_{F_i}||x_i - y_i||, \qquad \forall i \in \{1,\ldots,m\}, \qquad \forall x_i, y_i \in D_i \tag{2.30}$$

Nota 2.4 Pelo Lema 2.1, $F(x)$ é bloco-Lipschitz contínua, satisfazendo a relação:

$$||F(x) - F(y)|| \leq \sum_{i=1}^{m} l_{F_i}||x_i - y_i|| \tag{2.31}$$

Ademais, se utiliza um *'splitting'* da matriz A dado por

$$A = M - N \tag{2.32}$$

onde $M \in \mathbb{R}^n$ é uma matriz bloco-diagonal satisfazendo:

$$M = \text{ diag } (M_1,\ldots,M_m), \qquad M_i \in \mathbb{R}^{n_i \times n_i}, \qquad \det M_i \neq 0, \qquad \forall i \in \{1,\ldots,m\}, \tag{2.33}$$

o que implica que M não é singular, enquanto N satisfaz:

$$N_{ij} \in \mathbb{R}^{n_i \times n_j}, \qquad N := (N_{ij}), \qquad \forall i,j \in \{1,\ldots,m\} \tag{2.34}$$

Assim, o sistema quase linear de equações dado por (2.28) pode ser re-escrito conforme (2.6) como:

$$M_i x_i = \sum_{j=1}^{m} N_{ij} x_j + F_i(x_i), \qquad \forall i \in \{1,\ldots,m\} \tag{2.35}$$

ou mais explicitamente, na forma $\Phi_i(x) = 0$, onde:

$$\Phi_i(x) = -M_i x_i + \sum_{j=1}^{m} N_{ij} x_j + F_i(x_i), \qquad \forall i \in \{1,\ldots,m\} \tag{2.36}$$

que por Hipótese 2.1 tem uma única solução x^* em D satisfazendo:

$$x_i^* = \sum_{j=1}^{m} M_i^{-1} N_{ij} x_j^* + M_i^{-1} F_i(x_i^*), \qquad \forall i \in \{1,\ldots,m\} \tag{2.37}$$

A equação (2.35) correspondente a cada processador, pode assim ser resolvida por diferentes métodos que são analisados nas subseções que se seguem.

2.2.1 Método bloco Jacobi

O método *bloco Jacobi* (ou BJ) já é bem conhecido na literatura (Bertsekas and Tsitsiklis, 1989), inclusive sendo uitlizado na solução de sistemas quase lineares de equações, no contexto assíncrono dos algoritmos bloco iterativos (Bhaya et al., 1991). No que segue, demonstramos que a condição suficiente de convergência (assíncrona) derivada em Bhaya et al. (1991, teorema 2.1), pode ser obtida como conseqüência direta do Teorema 2.1 da seção anterior.

O método bloco Jacobi assíncrono para resolver a equação (2.35) é dado por:

$$x_i(k+1) \;=\; \sum_{j=1}^{m} M_i^{-1}\, N_{ij}\, x_j\Big(d_j^i(k)\Big) \;+\; M_i^{-1}\, F_i\Big(\, x_i\Big(d_i^i(k)\Big)\,\Big), \qquad \forall i \in \{1,\ldots,m\} \tag{2.38}$$

assim, o operador G_i associado ao processador i pode ser escrito como:

$$G_i(x) \;=\; \sum_{j=1}^{m} M_i^{-1} N_{ij} x_j \;+\; M_i^{-1} F_i(x_i), \qquad \forall i \in \{1,\ldots,m\} \tag{2.39}$$

que sob as Hipóteses 2.4 e 2.5, é bloco-Lipschitz contínuo em D com constantes:

$$l_{ij} \;=\; \begin{cases} ||M_i^{-1}N_{ii}|| + l_{F_i}||M_i^{-1}|| & \text{se } j = i \\ ||M_i^{-1}N_{ij}|| & \text{se } j \neq i \end{cases} \tag{2.40}$$

e pelo Teorema 2.1 temos:

COROLÁRIO 2.1 *A versão assíncrona do método bloco Jacobi dado por (2.38) converge para a única solução x^* em D, sob as hipóteses 2.1, 2.2, 2.4 e 2.5 se $\rho(H_{BJ}) < 1$, onde $H_{BJ} = (l_{ij})$ é dado por (2.40).*

Este resultado, originalmente demonstrado em Bhaya et al.(1991), foi apresentado com o único propósito de ilustrar a simplicidade com a qual o Teorema 2.1 pode ser aplicado na derivação de condições suficientes de convergência, utilizando o conceito de continuidade bloco-Lipschitz.

2.2.2 Método da Corda Paralela

O método da corda paralela, conhecido na literatura como *'Parallel Chord Method'* (ou PC) é discutido em datalhes na sua versão seqüencial em Varga (1962). Trata-se na verdade de uma classe de algoritmos que inclui vários métodos, dos quais o Newton Simplificado (ou SN) é provavelmente o mais conhecido. A versão seqüencial do método resolve o sistema de equações (2.1) utilizando o seguinte algoritmo iterativo:

$$x(k+1) \;=\; x(k) \;-\; Q\;\Phi\Big(x(k)\Big), \qquad Q \in \mathbb{R}^{n\times n} \tag{2.41}$$

onde Q é uma matriz constante que para o caso específico do método de Newton Simplificado é dada pela inversa do Jacobiano $J_\Phi(x)$ de $\Phi(x)$, avaliado na condição inicial $x(0)$, i.e.

$$Q \;=\; \Big[\, J_\Phi\big(x(0)\big)\,\Big]^{-1} \tag{2.42}$$

Nota 2.5 O método de Newton Simplificado foi recentemente tratado no contexto síncrono dos sistemas distribuídos, junto com 'overlapping epsilon decomposition', em Zečević e Šiljak (1992).

Nesta seção, considera-se o método mais geral da corda paralela (PC) num contexto assíncrono no qual cada processador i trata de *resolver* seu subproblema local $\Phi_i(x) = 0$ atualizando o valor de x_i, considerando como constantes para essa iteração k, os valores de x_j, $\forall j \neq i$. Conseqüentemente, cada processador i usa sua própria matriz constante Q_i, de forma similar a (2.41); desta forma, a versão assíncrona do algoritmo bloco iterativo que utiliza o método da corda paralela (ou bloco PC), pode ser escrito $\forall i \in \{1,\ldots,m\}$ como:

$$x_i(k+1) \;=\; x_i\big(d_i^i(k)\big) \;-\; Q_i\,\Phi_i\big(x^i(k)\big), \qquad Q_i \in \mathbb{R}^{n_i \times n_i} \tag{2.43}$$

onde, para o problema (2.35), $\Phi_i(x)$ é dado por (2.36).

Considerando o sistema quase linear de equações (2.35) e denotando ao Jacobiano de $F_i(x_i)$ calculado na condição inicial $x(0)$ como $J_{F_i}(\,x_i(0)\,)$, obtemos a versão assíncrona do algoritmo bloco iterativo que utiliza o método de Newton simplificado (ou bloco SN), sendo Q_i dada por:

$$Q_i \;=\; \Big[\, -M_i + N_{ii} + J_{F_i}\big(\,x_i(0)\,\big)\,\Big]^{-1} \tag{2.44}$$

Para aplicar o Teorema 2.1 ao método bloco PC dado por (2.43) quando utilizado para resolver (2.35), calculamos as constantes bloco-Lipschitz do operador $G_i(x)$. Assim, de (2.43) temos:

$$\begin{aligned} G_i(x) \;&=\; x_i \;-\; Q_i\left[-M_i x_i + F_i(x_i) \;+\; \textstyle\sum_{j=1}^{m} N_{ij} x_j\right] \\ &=\; \Big[I_i + Q_i(M_i - N_{ii})\Big]\, x_i \;-\; Q_i F_i(x_i) \;-\; \textstyle\sum_{j=1,j\neq i}^{m} Q_i N_{ij} x_j \end{aligned} \tag{2.45}$$

que pelas Hipóteses 2.4 e 2.5 resulta em:

$$\begin{aligned} \|G_i(x) - G_i(y)\| \;&\leq\; \|I_i + Q_i(M_i - N_{ii})\|\,\|x_i - y_i\| \;+\; \|Q_i\|\,\|F_i(x_i) - F_i(y_i)\| \\ &\quad + \textstyle\sum_{j=1,j\neq i}^{m} \|Q_i N_{ij}\|\,\|x_j - y_j\| \\ &\leq\; \Big[\,\|I_i + Q_i(M_i - N_{ii})\| \;+\; \|Q_i\|\,l_{F_i}\,\Big]\,\|x_i - y_i\| \\ &\quad + \textstyle\sum_{j=1,j\neq i}^{m} \|Q_i N_{ij}\|\,\|x_j - y_j\| \end{aligned} \tag{2.46}$$

conseqüentemente, G_i é bloco-Lipschitz contínua com constantes l_{ij} dadas por:

$$l_{ij} = \begin{cases} \|\, I + Q_i(M_i - N_{ii})\,\| \;+\; l_{F_i}\,\|Q_i\| & \text{se } j{=}i \\ \|Q_i N_{ij}\| & \text{se } j \neq i \end{cases} \tag{2.47}$$

então, pelo Teorema 2.1 podemos afirmar que:

COROLÁRIO 2.2 *A versão assíncrona do método bloco iterativo da corda paralela dado por (2.45) converge para a única solução x^* em D, sob as hipóteses 2.1, 2.2, 2.4 e 2.5 se $\rho(H_{PC}) < 1$, onde $H_{PC} = (l_{ij})$ é dado por (2.47).*

Enfatizamos que a obtenção de uma condição suficiente de convergência tanto no método BJ como no PC para um contexto assíncrono, se limitou ao cálculo das constantes bloco-Lipschitz (l_{ij}) do algoritmo em questão, para posteriormente utilizar o Teorema 2.1. Desta forma a utilização do Teorema 2.1 mostra-se flexível o suficiente para ser utilizada no caso de diferentes algoritmos.

2.2.3 Método da Solução por Componente

O método da Solução por Componente, conhecido na literatura como *Component Solution Method* (ou CS) é discutido em detalhes na sec.3.1.2 de Bertsekas e Tsitsiklis (1989b). É um método iterativo no qual cada processador realmente resolve sua equação local (2.6) a cada iteração. A implementação assíncrona trabalha da seguinte forma: a cada iteração k, um dado processador i resolve em x_i sua equação local $\Phi_i(x) = 0$, considerando como parâmetros (constantes a cada iteração) os valores de x_j, $\forall j \neq i$ dos outros processadores, obtendo desta forma a *solução local* $\bar{x}_i(k)$ da iteração considerada. Portanto, a solução local $\bar{x}_i(k)$ pode ser considerada como o valor atualizado de x_i que resulta da aplicação do operador G_i na iteração k; sendo assim,

$$x_i(k+1) = \bar{x}_i(k), \qquad \forall i \in \{1, \ldots, m\} \tag{2.48}$$

Uma vez obtido o valor de x_i, este deve ser transmitido aos outros processadores. Quando novos valores de x_j, $\forall j \neq i$ são recebidos dos outros processadores, uma nova solução local $\bar{x}_i$ é calculada . O algoritmo bloco iterativo assim descrito prossegue até que uma dada tolerância ϵ seja satisfeita.

Usando uma notação similar àquela introduzida em (2.12), denotamos como $\bar{x}^i(k)$ ao vetor x disponível no processador i apois resolver a equação local (2.6) na iteração k ; i.e.

$$\bar{x}^i(k) \stackrel{\text{def}}{=} \begin{bmatrix} x_1\big(d_1^i(k)\big) \\ \vdots \\ \bar{x}_i(k) \\ \vdots \\ x_m\big(d_m^i(k)\big) \end{bmatrix} \tag{2.49}$$

Considerando a Hipótese 2.1 e lembrando que $\bar{x}_i(k)$ é a solução de (2.6) na iteração k, podemos escrever:

$$\bar{x}_i(k) = G_i\big(\bar{x}^i(k)\big), \qquad \forall i \in \{1, \ldots, m\} \tag{2.50}$$

note que $\bar{x}_i(k)$ aparece nos dois membros da equação (2.50) pois é a própria solução para um dado $x^i(k)$.

Em Bertsekas e Tsitsiklis (1989), é demonstrada a convergência do método CS num contexto síncrono, para a única solução x^* em D, quando G_i é uma contração. Para este trabalho, consideramos a versão assíncrona na qual cada G_i satisfaz a seguinte hipótese:

HIPÓTESE 2.6 (CONTRAÇÃO BLOCO-LIPSCHITZ)
Cada operator $G_i(x)$ dado por (2.13) é bloco-Lipschitz contínuo com $l_{ii} \in [0,1)$, i.e.

$$\forall x, y \in D, \qquad \|G_i(x) - G_i(y)\| \leq \sum_{j=1}^{m} l_{ij} \|x_i - y_i\| \qquad \text{onde} \quad 0 \leq l_{ii} < 1$$

Para calcular a *Matriz de Comparação* correspondente a este método subtrai-se a equação (2.9) de (2.50) e toma-se a norma $\|\cdot\|$ em ambos os membros; assim, usando (2.48) obtemos:

$$\|x_i(k+1) - x_i^*\| = \|G_i\big(\bar{x}^i(k)\big) - G_i\big(x^*\big)\|, \qquad \forall i \in \{1, \ldots, m\}$$

e pela Hipótese 2.6:

$$\|x_i(k+1) - x_i^*\| \leq l_{ii}\|x_i(k+1) - x_i^*\| + \sum_{j=1\, j\neq i}^{m} l_{ij}\|x_j\big(d_j^i(k)\big) - x_j^*\|, \qquad \forall i \in \{1, \ldots, m\}$$

ou usando a notação do erro reduzido (2.22):

$$z_i(k+1) \leq l_{ii}\, z_i(k+1) + \sum_{j=1\, j\neq i}^{m} l_{ij}\, z_j\big(d_j^i(k)\big), \qquad \forall i \in \{1, \ldots, m\}$$

subtraindo $l_{ii}\, z_i(k+1)$ e dividindo por $(1 - l_{ii}) > 0$ obtemos:

$$z_i(k+1) \leq \sum_{j=1\, j\neq i}^{m} \left(\frac{l_{ij}}{1 - l_{ii}}\right) z_j\big(d_j^i(k)\big), \qquad \forall i \in \{1, \ldots, m\} \tag{2.51}$$

Finalmente, podemos construir a Matriz de Comparação $H_{CS} = (h_{ij})$ para o método CS e aplicar o Lema 2.2 da mesma forma que na seção 2.1, para obter uma condição suficiente de convergência. Comparando as equações (2.27) e (2.51) obtemos a Matriz de Comparação:

$$H_{CS} = (h_{ij}), \qquad \text{onde} \quad h_{ij} = \begin{cases} 0 & \text{se } j = i \\ \dfrac{l_{ij}}{1 - l_{ii}} & \text{se } j \neq i \end{cases} \tag{2.52}$$

Conseqüentemente, utilizando os mesmos argumentos utilizados para derivar o Teorema 2.1 podemos afirmar que:

COROLÁRIO 2.3 *A versão assíncrona do método da solução por componente (CS) converge para a única solução x^* em D, sob as hipóteses 2.1, 2.2 e 2.6 se $\rho(H_{CS}) < 1$, onde $H_{CS} = (h_{ij})$ é dado pela equação (2.52).*

Para o caso particular do sistema quase linear de equações, G_i é dado pela equação (2.39) e a correspondente Matriz de Comparação $H_{CS} = (h_{ij})$, usando (2.40), está dada por:

$$h_{ij} = \begin{cases} 0 & \text{se } j = i \\ \dfrac{\|M_i^{-1}N_{ij}\|}{1 - (\|M_i^{-1}N_{ii}\| + l_{F_i}\|M_i^{-1}\|)} & \text{se } j \neq i \end{cases} \tag{2.53}$$

e o Corolário 2.3 pode ser aplicado para obter a condição de convergência desejada.

Antes de concluir a discussão do *método da solução por componente*, enfatizamos que o algoritmo utilizado em cada processador para resolver seu subproblema local (2.6) não necessariamente é o mesmo em todos os processadores. Assim, processadores que devem resolver subproblemas lineares podem utilizar qualquer variação do método de eliminação de Gauss, enquanto outros processadores que necessitam resolver sistemas não lineares de equações, podem utilizar outro método mais apropriado. Inclusive, é possível que algum processador utilize um método iterativo (como o Newton-Raphson) para resolver seu subproblema local. Neste último caso, uma iteração CS termina somente apois encontrar a solução local, o que pode levar por sua vez, várias iterações do algoritmo localmente utilizado (o Newton-Raphson em nosso exemplo). Conseqüentemente, combinações de diferentes algoritmos executados nos diversos processadores do sistema computacional distribuído, são possíveis e eventualmente até recomendáveis, como veremos mais adiante.

2.2.4 Método das Iterações Contrativas

Iterações Contrativas seqüenciais são discutidas na sec.3.1.1 de Bertsekas e Tsitsiklis (1989) como um método iterativo no qual o operador G é um mapeamento contrativo.

DEFINIÇÃO 2.3 (MAPEAMENTO CONTRATIVO)
Uma função $G(x)$ definida num domínio D é denominada 'Mapeamento Contrativo' em D sob uma norma $\|\cdot\|$, se existe uma costante α tal que

$$\forall x, y \in D, \qquad \|G(x) - G(y)\| \leq \alpha\|x - y\| \qquad \text{onde } 0 \leq \alpha < 1; \tag{2.54}$$

α é conhecido como constante de contração ou *módulo* de G.

A convergência do método das iterações contrativas, num contexto seqüencial, é assegurada pelo seguinte teorema, demonstrado em Ortega e Rheinboldt (1970):

TEOREMA 2.3 (CONVERGÊNCIA DE MAPEAMENTOS CONTRATIVOS)
Dado um operador G que satisfaz (2.54), ou seja, contrativo em D com módulo α, que ademais satisfaz a Hipótese 2.1; então, o algoritmo iterativo

$$x(k+1) \;=\; G\big(x(k)\big)$$

converge geometricamente para a única solução x^* em D, com uma taxa de convergência ('convergence rate') de α.

Para derivar uma condição suficiente de convergência para uma versão assíncrona dos algoritmos bloco iterativos que utilizam iterações contrativas (ou bloco CI), fazemos a seguinte hipótese:

HIPÓTESE 2.7 (MAPEAMENTO CONTRATIVO)
Cada operador $G_i(x)$ de (2.13) é um mapeamento contrativo em D, $\forall i \in \{1,\ldots,m\}$, i.e.

$$\forall x, y \in D, \qquad \|G_i(x) - G_i(y)\| \;\leq\; \alpha_i \, \|x - y\| \qquad \textit{onde} \quad 0 \leq \alpha_i < 1$$

Com esta hipótese podemos escrever a relação:

$$\|G_i\big(x^i(k)\big) \;-\; G_i\big(\bar{x}^i(k)\big)\| \;\leq\; \alpha_i \, \|x^i(k) - \bar{x}^i(k)\|, \qquad \forall i \in \{1,\ldots,m\}$$

onde o valor de $x^i(k)$ definido em (2.12) difere de $\bar{x}^i(k)$ dado em (2.49) somente na componente x_i; conseqüentemente,

$$\|G_i\big(x^i(k)\big) - G_i\big(\bar{x}^i(k)\big)\| \;\leq\; \alpha_i \, \|x_i\big(d_i^i(k)\big) - \bar{x}_i(k)\|, \qquad \forall i \in \{1,\ldots,m\}$$

e usando as relações (2.13) e (2.50) obtemos:

$$\|x_i(k+1) - \bar{x}_i(k)\| \;\leq\; \alpha_i \, \|x_i\big(d_i^i(k)\big) - \bar{x}_i(k)\| \qquad \forall i \in \{1,\ldots,m\} \qquad (2.55)$$

Derivamos a seguir uma cota superior ('upper bound') da *distância* entre a solução local $\bar{x}_i(k)$ da equação (2.6) na iteração k e a solução global x_i^* da mesma, medida na mesma norma $\|\cdot\|$ utilizada nas hipóteses 2.6 e 2.7 que assumimos verdadeiras bem como as hipóteses 2.1 e 2.2.

$$\begin{aligned} \|\bar{x}_i(k) - x_i^*\| \;&=\; \|G_i(\bar{x}^i) - G_i(x^*)\| \\ &\leq\; l_{ii}\|\bar{x}_i(k) - x_i^*\| + \textstyle\sum_{j=1\, j\neq i}^{m} l_{ij}\|x_j\big(d_j^i(k)\big) - x_j^*\| \end{aligned} \qquad (2.56)$$

subtraindo $l_{ii}\|\bar{x}_i(k) - x_i^*\|$ e usando a notação do erro reduzido z definido em (2.22),

$$(1 - l_{ii})\|\bar{x}_i(k) - x_i^*\| \;\leq\; \sum_{j=1\, j\neq i}^{m} l_{ij} z_j\big(d_j^i(k)\big), \qquad \forall i \in \{1,\ldots,m\} \qquad (2.57)$$

dividindo por $(1 - l_{ii}) > 0$, considerando a Hipótese 2.6

$$\|\bar{x}_i(k) - x_i^*\| \leq \sum_{j=1\, j\neq i}^{m} \frac{l_{ij}}{1 - l_{ii}} z_j\left(d_j^i(k)\right), \qquad \forall i \in \{1, \ldots, m\} \tag{2.58}$$

e da definição (2.22) do erro reduzido z, temos:

$$\begin{aligned} z_i(k+1) &= \|x_i(k+1) - \bar{x}_i(k) + \bar{x}_i(k) - x_i^*\| \\ &\leq \|x_i(k+1) - \bar{x}_i(k)\| + \|\bar{x}_i(k) - x_i^*\| \end{aligned}$$

usando a relação (2.55) obtemos

$$z_i(k+1) \leq \alpha_i \|x_i\left(d_i^i(k)\right) - \bar{x}_i(k)\| + \|\bar{x}_i(k) - x_i^*\|$$

onde

$$\begin{aligned} \|x_i\left(d_i^i(k)\right) - \bar{x}_i(k)\| &= \|x_i\left(d_i^i(k)\right) - x_i^* + x_i^* - \bar{x}_i(k)\| \\ &\leq \|x_i\left(d_i^i(k)\right) - x_i^*\| + \|x_i^* - \bar{x}_i(k)\| \end{aligned}$$

conseqüentemente,

$$z_i(k+1) \leq \alpha_i \|x_i\left(d_i^i(k)\right) - x_i^*\| + (1 + \alpha_i)\|\bar{x}_i - x_i^*\|$$

e usando a notação (2.22) do erro reduzido:

$$z_i(k+1) \leq \alpha_i\, z_i\left(d_i^i(k)\right) + (1 + \alpha_i)\|\bar{x}_i - x_i^*\|$$

assim, pela desigualdade (2.58)

$$z_i(k+1) \leq \alpha_i\, z_i\left(d_i^i(k)\right) + (1 + \alpha_i) \sum_{j=1\, j\neq i}^{m} \left(\frac{l_{ij}}{1 - l_{ii}}\right) z_j\left(d_j^i(k)\right) \tag{2.59}$$

finalmente, podemos escrever a Matriz de Comparação $H_{CI} = (h_{ij})$ do método bloco CI como:

$$H_{CI} = (h_{ij}), \qquad h_{ij} = \begin{cases} \alpha_i & \text{se } j = i \\ (1 + \alpha_i)\, \dfrac{l_{ij}}{1 - l_{ii}} & \text{se } j \neq i \end{cases} \tag{2.60}$$

Conseqüentemente, utilizando os mesmos argumentos utilizados para derivar o Teorema 2.1 podemos afirmar que:

COROLÁRIO 2.4 *A versão assíncrona do método bloco iterativo que utiliza contrações iterativas, converge para a única solução x^* em D, sob as hipóteses 2.1, 2.2, 2.6 e 2.7 se $\rho(H_{CI}) < 1$, onde $H_{CI} = (h_{ij})$ é dado pela equação (2.60).*

Nota 2.6 Se todos os módulos satisfazem a relação $\alpha_i = 0$, a equação (2.60) coincide com a (2.52) e o método bloco CI se converte no método da solução por componente.

Considerando o caso específico das equações quase lineares, nas quais as constantes bloco-Lipschitz dos operadores G_i são dadas pela equação (2.40), obtemos a seguinte Matriz de Comparação:

$$H_{CI} = (h_{ij}), \qquad h_{ij} = \begin{cases} \alpha_i & \text{se } j = i \\ (1+\alpha_i)\,\dfrac{\|M_i^{-1}N_{ij}\|}{1-(\|M_i^{-1}N_{ii}\| + l_{F_i}\|M_i^{-1}\|)} & \text{se } j \neq i \end{cases} \tag{2.61}$$

conseqüentemente, o Corolário 2.4 pode ser utilizado para obter uma condição suficiente de convergência.

Para finalizar esta discussão, enfatiza-se que as iterações contrativas são uma classe geral de algoritmos que inclui vários métodos quando aplicados numa região suficientemente próxima da solução, sendo o Newton-Raphson o mais conhecido (Bertsekas and Tsitsiklis, 1989b; Varga, 1962). Efetivamente, é provado na sec.3.2.1 de Bertsekas e Tsitsiklis (1989b) que, se o *Jacobiano* $J_i(x)$ de $\Phi_i(x)$ é definido positivo em D então, para todo $\alpha_i \in (0,\,1)$ existe um $\varepsilon > 0$ tal que

$$\|x_i(k+1) - \bar{x}_i(k)\| \leq \alpha_i \, \|x_i(k) - \bar{x}_i(k)\| \qquad \forall x_i(k) \quad \text{com} \quad \|x_i(k) - \bar{x}_i(k)\| \leq \varepsilon$$

consequentemente, a versão assíncrona dos algoritmos bloco iterativos que utilizam o método de Newton-Raphson, pode ser considerada uma iteração contrativa numa região bem próxima da solução, e o Corolário 2.4 pode ser utilizado para obter uma condição suficiente de convergência.

2.3 Bloco Team Algorithms

Para resolver um problema geral do tipo $\Phi(x) = 0$, utilizando m processadores, o primeiro passo é particionar o problema em m subproblemas, de forma que cada processador i possa resolver localmente um subproblema do tipo $\Phi_i(x) = 0$. O estudo detalhado dos métodos que permitem obter uma boa partição para um dado problema, está fora do escopo do presente trabalho. Conseqüentemente, assumimos que a partição do problema é conhecida, e nosso objetivo é resolver o problema para uma dada partição.

O leitor interessado especificamente nos métodos de partição, pode consultar os diversos trabalhos já publicados nas últimas décadas (ou seja: Carré, 1968; Sasson, 1970; Ogbuobiri et al., 1970; Undrill e Happ, 1971; Aboytes e Sasson, 1971; Mickle et al., 1977; Lee et al., 1979; Vanelli, 1983; Sezer e Šiljak, 1984 e 1986); bem como, trabalhos mais recentes(Irving e Sterling, 1990; Vale et al., 1992).

Uma vez particionado o problema, cada processador i deve ter condições de resolver seu subproblema local $\Phi_i(x) = 0$ utilizando para tanto um algoritmo que re-

presentamos por seu operador G_i. Em geral, os diversos subproblemas podem ter características bem diferenciadas, o que complica a escolha de um único algoritmo capaz de resolver igualmente bem todos e cada um dos subproblemas (como exemplo, o método de Newton pode ser ótimo para alguns subproblemas, mas apresentar um Jacobiano singular em outros). A escolha do algoritmo apropriado para cada processador, pode ser feita de forma independente em cada processador, i.e. para cada processador se escolhe o algoritmo que mais bem resolve seu subproblema local independentemente da escolha nos outros processadores. Desta forma, tem-se uma combinação de algoritmos eventualmente diferentes nos diversos processadores do sistema, que denominamos *Bloco Team Algorithm.*

As vantagens de se poder combinar distintos algoritmos em diferentes processadores ficam evidentes no exemplo do capítulo 4 onde se apresenta um problema de Fluxo de Potência particionado de forma que alguns subproblemas não podem ser resolvidos pelo método Desocoplado Rápido (derivado do Newton) e sim pelo método bloco Jacobi, enquanto que com outros subproblemas do sistema acontece exatamente o contrário (somente podem ser resolvidos pelo método Desacoplado Rápido). O resultado é que somente um *bloco TA* que utiliza o método adequado a cada subproblema consegue resolver o problema, pois nenhum dos algoritmos citados consegue resolver individualmente o problema, nem mesmo no caso de uma implementação seqüencial.

Especificamente, nesta seção estamos interessados na convergência dos *Bloco Team Algorithms* quando implementados num sistema paralelo assíncrono, nas condições descritas neste capítulo. Para isto, basta notar que na derivação do Teorema 2.1 não foi assumido que os algoritmos executados em cada processador devem ser os mesmos. Conseqüentemente, o citado teorema fornece uma condição suficiente de convergência para os *bloco TA*. Mais especificamente, cada G_i pode representar um algoritmo diferente, com suas próprias constantes bloco-Lipschitz l_{ij}; desta forma, o raio espectral $\rho(H)$ da *Matriz de Comparação* $H = (h_{ij})$ com $h_{ij} = l_{ij}$ pode ser calculado para verificar se $\rho(H) < 1$ e ter assim a convergência do *Bloco TA* assegurada. Para esclarecer estas idéias, consideremos o seguinte exemplo, que mesmo sendo simples permite ilustrar algumas características dos *Team Algorithms.*

Exemplo 2.2 Utilizando uma versão assíncrona com dois processadores ($m = 2$) resolver, utilizando um método iterativo, o sistema de equações não lineares

$$\begin{array}{rcl} x_1^2 + a_{11}x_1 + a_{12}x_2 & = & 0 \\ x_2^2 + a_{21}x_1 + a_{22}x_2 & = & 0 \end{array} \tag{2.62}$$

onde $\forall i, j \in \{1,2\}$ temos que $a_{ii} > 2$ e $0 \leq a_{ij} < 1$, para $i \neq j$; definido num domínio $D = \{x \in \mathbb{R}^2 | \ \|x\|_\infty \leq 1\}$ e assumindo que o sistema não está definido fora do domínio dado;

Note que a equação (2.62) é *quase-linear* e portanto pode ser escrita na forma $Ax = F(x)$ como:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -x_1^2 \\ -x_2^2 \end{bmatrix} \tag{2.63}$$

onde $F_i(x_i) = -x_i^2$ tem uma constante de Lipschitz $l_{F_i} = 2$ em D. Assim, os operadores G_i para cada método já estudado junto com as correspondentes matrizes de comparação $H = (h_{ij})$, podem ser obtidos diretamente das equações derivadas na seção anterior. A matriz de comparação pode também ser obtida calculando as constantes bloco-Lipschitz diretamente em cada operador G_i utilizando a relação

$$\forall i, j \in \{1,2\}, \qquad l_{ij} \geq \max_{x \in D} \left| \frac{\partial G_i}{\partial x_j} \right| \tag{2.64}$$

e lembrando o Teorema 2.1 ($h_{ij} = l_{ij}$). A Tabela 2.1 resume os operadores e as matrizes de comparação para vários métodos incluindo diferentes combinações de algoritmos ou *bloco Team Algorithms.*

Como pode-se observar na Tabela 2.1, um *bloco TA* se forma utilizando um algoritmo num processador e um método diferente no outro. Como exemplo, o 'Team A' utiliza o método de Newton-Raphson (NR) no processador 1 e o método da solução por componente (CS) no processador 2; conseqüentemente, o operador G_1 do 'Team A' e do NR são idênticos, com as mesmas constantes bloco-Lipschitz (l_{11} e l_{12}); analogamente, o operador G_2 do 'Team A' é idêntico ao operador do método CS e portanto com as mesmas constantes bloco-Lipschitz (l_{21} e l_{22}). Vemos assim que o 'Team A' tem uma matriz de comparação

$$H_A = \begin{bmatrix} \frac{2(a_{11}+a_{12}-1)}{(a_{11}-2)^2} & \frac{a_{12}}{a_{11}-2} \\ \\ \frac{a_{21}}{\sqrt{a_{22}^2-4a_{21}}} & 0 \end{bmatrix}$$

que não é senão a primeira linha de H_{NR} e a segunda linha de H_{CS}. Em geral, outras combinações de algoritmos são possíveis, como ilustra a Tabela 2.1; mas sempre o princípio é o mesmo: cada processador utiliza um operador G_i correspondente a esse método e a matriz de comparação H forma-se copiando simplesmente a linha i da matriz de comparação correspondiente. Como exemplos, a Tabela 2.1 apresenta 4 combinações diferentes.

Concluimos enfatizando que o Teorema 2.1 pode ser utilizado para derivar uma condição suficiente de convergência em sistemas distribuídos assíncronos, mesmo quando algoritmos diferentes são combinados nos distintos processadores. Isto é, o Teorema 2.1 fornece uma condição suficiente de convergência para os *bloco Team Algorithms.*

2.4 Comparação de Métodos em Termos de Convergência

Quando se quer resolver um dado problema com sua correspondente partição, primeiramente tentamos conhecer quais são os métodos disponíveis. Se vários métodos e combinações existem, é natural perguntar-se: *qual algoritmo ou combinação escolher?*

ALGORITMO (ou MÉTODO)	OPERADOR $G_i \quad (i=1,2)$	MATRIZ DE COMPARAÇÃO $(\forall i,j \in \{1,2\}, \quad j \neq i)$
método CS	$G_i = \frac{1}{2}\left(-a_{ii} + \sqrt{a_{ii}^2 - 4a_{ij}x_j}\right)$	$h_{ii} = 0$ $h_{ij} = a_{ij} \,/\, \sqrt{a_{ii}^2 - 4a_{ij}}$
método BJ	$G_i = \frac{-1}{a_{ii}}\left(x_i^2 + a_{ij}x_j\right)$	$h_{ii} = 2 \,/\, a_{ii}\,, \quad h_{ij} = a_{ij} \,/\, a_{ii}$
método NR	$G_i = (x_i^2 - a_{ij}x_j) \,/\, J_i(x)$ $J_i(x) = 2x_i + a_{ii}$	$h_{ii} = 2(a_{ii} + a_{ij} - 1) \,/\, (a_{ii} - 2)^2$ $h_{ij} = a_{ij} \,/\, (a_{ii} - 2)$
método SN	$G_i = x_i - Q_i(x_i^2 + a_{ii}x_i + a_{ij}x_j)$ $Q_i^{-1} = 2x_i(0) + a_{ii}$	$h_{ii} = 2\,\lvert Q_i\rvert + \lvert 1 - a_{ii}Q_i\rvert$ $h_{ij} = \lvert Q_i a_{ij}\rvert$
Team A: (NR) + (CS)	$G_1 = \left(x_1^2 - a_{12}x_2\right) \,/\, J_1(x)$ $G_2 = \frac{1}{2}\left(-a_{22} + \sqrt{a_{22}^2 - 4a_{21}x_1}\right)$	$h_{11} = 2(a_{11} + a_{12} - 1) \,/\, (a_{11} - 2)^2$ $h_{12} = a_{12} \,/\, (a_{11} - 2)$ $h_{21} = \frac{a_{21}}{\sqrt{a_{22}^2 - 4a_{21}}}\,, \quad h_{22} = 0$
Team B: (NR) + (BJ)	$G_1 = \left(x_1^2 - a_{12}x_2\right) \,/\, J_1(x)$ $G_2 = \frac{-1}{a_{22}}\left(x_2^2 + a_{21}x_1\right)$	$h_{11} = 2(a_{11} + a_{12} - 1) \,/\, (a_{11} - 2)^2$ $h_{12} = a_{12} \,/\, (a_{11} - 2)$ $h_{21} = a_{21} \,/\, a_{22}\,, \quad h_{22} = 2 \,/\, a_{22}$
Team C: (SN) + (CS)	$G_1 = x_1 - Q_1(x_1^2 + a_{11}x_1 + a_{12}x_2)$ $G_2 = \frac{1}{2}\left(-a_{22} + \sqrt{a_{22}^2 - 4a_{21}x_1}\right)$	$h_{11} = 2\lvert Q_1\rvert + \lvert 1 - a_{11}Q_1\rvert$ $h_{12} = \lvert Q_1 a_{12}\rvert$ $h_{21} = \frac{a_{21}}{\sqrt{a_{22}^2 - 4a_{21}}}\,, \quad h_{22} = 0$
Team D: (CS) + (BJ)	$G_1 = \frac{1}{2}\left(-a_{11} + \sqrt{a_{11}^2 - 4a_{12}x_2}\right)$ $G_2 = \frac{-1}{a_{22}}\left(x_2^2 + a_{21}x_1\right)$	$h_{11} = 0\,, \quad h_{12} = \frac{a_{12}}{\sqrt{a_{11}^2 - 4a_{12}}}$ $h_{21} = \frac{a_{21}}{a_{22}}\,, \quad h_{22} = \frac{2}{a_{22}}$

Tabela 2.1: Operadores e matrizes de comparação do Exemplo 2.2.

Em principio, uma resposta pode ser obtida experimentalmente implementando todos os métodos e combinações possíveis; mas não sempre isto é possível ou desejável. Muitas vezes necessitamos uma forma de estimar *'a priori'* um método (ou algumas poucas variantes) que possa resolver nosso problema razoavelmente bem. Neste caso, dois fatores tem que ser considerados:

(1) taxa de convergência que determina o número de iterações até a convergência e está intimamente relacionado com a matriz de iteração, da qual a Matriz de Compararação H é um majorante ('upper bound');

(2) tempo por iteração que está intimamente relacionado com o 'Hardware' disponível e o número de operações por iteração.

Assim sendo, postulamos aqui a utilização do raio espectral $\rho(H)$ da matriz de comparação, como um *Indicador de Mérito* da taxa de convergência. Desta forma, para um problema como o dado no Exemplo 2.2, poderiamos calcular *'a priori'* os raios espectrais e o número de operações por iteração para decidir sobre o método a ser implementado. O exemplo a seguir esclarece a idéia e apresenta resultados experimentais.

Exemplo 2.3 Resolver o problema do Exemplo 2.2 para a seguinte matriz A:

$$A = \begin{bmatrix} 2.1 & 0.9 \\ 0.9 & 800 \end{bmatrix} \tag{2.65}$$

A Tabela 2.2 mostra os algoritmos considerados na seção anterior, ordenados em ordem decrescente segundo os valores dos respectivos raios espectrais $\rho(H)$ da correspondente matriz de comparação. A última coluna mostra o número de operações por iteração. Os 3 melhores 'candidatos' da lista estão claramente ressaltados: o método CS por ter o menor raio espectral, o que nos leva a pensar que converge com o menor número de iterações, o método BJ por ter o menor número de operações por iteração o que implica menor tempo por iteração, e evidentemente, o Team D que combina estes dois métodos. Simulações foram realizadas supondo um atraso constante de duas iterações ($d_j^i(k) = 2$), escolhendo aleatoriamente a condição inicial $x(0) \in D$. Resultados experimentais sobre uma base de 100 casos são apresentados na Tabela 2.2, onde se indica para cada algoritmo, o número de problemas resolvidos e o número de vezes que a solução foi encontrada com um número de iterações menor o igual ao dos outros métodos. A última coluna indica o número total de operações por iteração, sendo que a indica adição, m multiplicação, d divisão e r raiz quadrada.

A primeira característica que se pode notar na Tabela 2.2 é que os algoritmos com $\rho(H) < 1$ resolvem todos os casos, como estabelece o Teorema 2.1, enquanto aqueles com $\rho(H) > 1$ podem resolver ou não o problema, dependendo da condição inicial. Pode-se notar ademais que o número de vezes que um algoritmo consegue resolver um problema com um número mínimo de iterações está correlacionado com o

ALGORITMO OU MÉTODO	$\rho(H)$	número de problemas resolvidos	soluções em número mínimo de iterações	operações por iteração
método CS	0.034	100	64	$2r + 6m + 4a$
Team D: (CS) + (BJ)	0.035	100	62	$r + 6m + 3a$
método BJ	0.95	100	30	$6m + 2a$
Team C: (SN) + (CS)	~ 40	67	22	$r + 7m + 4a$
método SN	~ 40	67	20	$8m + 4a$
Team A: (NR) + (CS)	~ 400	72	20	$r + d + 6m + 4a$
Team B: (NR) + (BJ)	~ 400	72	20	$d + 6m + 3a$
método NR	~ 400	72	20	$2d + 6m + 4a$

Table 2.2: `Resultados experimentais do Exemplo 2.3` .

raio espectral quando $\rho(H) < 1$ (o Teorema 2.1 é aplicavel). Aqui deve ser enfatizado que a relação $\rho(H_{CS}) < \rho(H_{BJ}) < 1$ não implica que para toda condição inicial o método CS converge em menos iterações que o BJ; mas, simplesmente indica que escolhendo aleatoriamente $x(0)$, é mais provável que o método CS possa convergir em menos iterações.

A utilização de $\rho(H)$ como indicador de mérito se mostra razoável em vários outros problemas experimentados. Porém, existem casos para os quais a complicação no cálculo das constantes bloco-Lipschitz leva a valores elevados de $\rho(H)$ que não refletem todas as vantagens do método considerado. Como exemplo, a Tabela 2.2 mostra que o Team A tem um raio espectral bem maior que o método SN ($\rho(H_A) \gg \rho(H_{SN})$) mas o supera em número de problemas resolvidos.

Observando na Tabela 2.2 o desempenho do Team D, vemos que é práticamente equivalente ao CS, porém com menos operações por iteração. Isto porque o subproblema $\Phi_2(x) = 0$ é relativamente simples de ser resolvido e qualquer método estudado pode resolve-lo, portanto o BJ é uma boa escolha, por ser o que usa menor número de operações por iteração. Contudo, o subproblema $\Phi_1(x) = 0$ somente é bem resolvido pelo mètodo CS. Resulta assim que o *bloco Team Algorithm* é uma opção bem promissora, o que poderia ser anticipado notando que $\rho(H_D) \sim \rho_{\text{mínimo}}$.

Para concluir este capítulo, enfatizamos uma característica que foi repetidamente encontrada na solução de diversos problemas, e que pode ser verificada na Tabela 2.2, é a relação:

$$\rho_{\text{mínimo}} = \rho(H_{CS}) \tag{2.66}$$

que leva ao método CS a ser o que geralmente converge no menor número de iterações, mesmo que isto não implique necessariamente menor tempo, pois geralmente, o método CS necessita um número relativamente grande de operações por iteração. A explicação é dada no seguinte teorema:

TEOREMA 2.4
Se existe um algoritmo com um operador G e matriz de comparação H que possa assegurar a convergência de um problema de acordo com o Teorema 2.1, i.e. $\rho(H) < 1$, então, a matriz de comparação do método de solução por componente H_{CS} satisfaz a relação:

$$\rho(H_{CS}) \leq \rho(H) < 1 \tag{2.67}$$

Prova.

Consideremos a matriz de comparação H dada por

$$H := (h_{ij}), \qquad \text{onde} \quad h_{ij} = l_{ij}, \tag{2.68}$$

então, segundo a equação (2.52) a matriz de comparação do método CS está dada por:

$$H_{CS} := (h_{ij}), \qquad \text{onde} \quad h_{ij} = \begin{cases} 0 & \text{se } j = i \\ \left(\frac{l_{ij}}{1-l_{ii}}\right) & \text{se } j \neq i \end{cases} \tag{2.69}$$

A prova deste teorema é baseada na seguinte relação dada em Stoer e Witzgall (1962) para uma matriz H não negativa:

$$\rho(H) = \inf_{U \in \mathcal{P}} ||U^{-1}HU||_\infty \tag{2.70}$$

onde $\mathcal{P}$ é o conjunto de todas as matrizes diagonais e positivas. Claramente, para uma matriz diagonal e positiva qualquer $U \in \mathcal{P}$, temos a relação (Varga, 1969):

$$\rho(H) \leq ||U^{-1}HU||_\infty \tag{2.71}$$

Agora, por hipótese e utilizando a relação (2.70) temos que

$$\frac{1}{u_i} \sum_{j=1}^{m} u_j l_{ij} < 1 \qquad \forall i \in \{1, \ldots, m\}$$

que multiplicando por $u_i l_{ii} \geq 0$ resulta em:

$$l_{ii} \sum_{j=1}^{m} u_j l_{ij} \leq u_i l_{ii}$$

isto é,

$$0 \leq u_i l_{ii}(1 - l_{ii}) - l_{ii} \sum_{j \neq i} u_j l_{ij}$$

somando $\sum_{j \neq i} u_j l_{ij}$

$$\sum_{j \neq i} u_j l_{ij} \leq u_i l_{ii}(1 - l_{ii}) + (1 - l_{ii}) \sum_{j \neq i} u_j l_{ij}$$

i.e.,

$$\sum_{j \neq i} u_j l_{ij} \leq (1 - l_{ii}) \sum_{j=1}^{m} u_j l_{ij}$$

e lembrando que $\rho(H) < 1$ implica $l_{ii} < 1$ (Horn e Johnson, 1988, Corolário 8.1.2), podemos dividir por $u_i(1 - l_{ii}) > 0$

$$\frac{1}{u_i} \sum_{j \neq i} u_j \left(\frac{l_{ij}}{1 - lii} \right) \leq \frac{1}{u_i} \sum_{j=1}^{m} u_j l_{ij} \qquad \forall i \in \{1, \ldots, m\}$$

assim, por hipótese e usando (2.68) e (2.69) podemos afirmar que:

$$||U^{-1} H_{CS} U||_\infty \leq ||U^{-1} H U||_\infty < 1 \tag{2.72}$$

Usando este resultado e a equação (2.70) para a matriz diagonal positiva U para a qual o mínimo acontece, podemos escrever:

$$||U^{-1} H_{CS} U||_\infty \leq \rho(H) = \inf_{U \in \mathcal{P}} ||U^{-1} H U||_\infty \tag{2.73}$$

finalmente, com (2.71) em (2.73) obtemos:

$$\rho(H_{CS}) \leq ||U^{-1} H_{CS} U||_\infty \leq \rho(H) \tag{2.74}$$

conseqüentemente, por hipótese:

$$\rho(H_{CS}) \leq \rho(H) < 1 \tag{2.75}$$

■

Nota 2.7 Satisfeita a Hipótese 2.1, se um dado algoritmo consegue provar sua convergência pelo Teorema 2.1, então o método da solução por componente (CS) também converge; e mais ainda, provavelmente o método CS consegue convergir num número menor de iterações que o algoritmo dado.

Sumário do Capítulo 2

Neste capítulo foram formalizados os Algoritmos Bloco Iterativos Assíncronos e estudado a convergência dos mesmos, o que permitiu derivar o Teorema 2.1 que estabelece uma condição suficiente de convergência para estes algoritmos, num contexto parcialmente assíncrono.

A partição de sistemas de equações algébricas de grande porte para sua posterior resolução, utilizando algoritmos bloco iterativos num contexto assíncrono, resulta portanto num alto grau de compatibilidade com os sistemas computacionais de memória distribuída, viabilizando assim a resolução de problemas de grande porte, nestes sistemas computacionais de menor custo por instrução e até em menor tempo (com considerável 'speedup').

Varias versões dos algoritmos bloco iterativos assíncronos foram estudadas; e condições suficientes de convergência foram derivadas para cada caso, enfatizando-se a aplicação destes resultados a sistemas de equações quase-lineares. Dentre estas versões estudadas, destacam-se os métodos bloco Jacobi (BJ), o Newton simplificado (SN), o método da solução por componente (CS) e o método das iterações contrativas (CI).

Como conseqüencia direta da análise individual acima mencionada, os *bloco Team Algorithms* foram formalizados e exemplificados, estabelecendo-se uma metodologia para derivar uma condição suficiente de convergência para cada possível bloco TA. Este capítulo termina postulando a utilização de um *Indicador de Mérito* que permite comparações 'a priori' entre as diferentes versões dos algoritmos bloco iterativos assíncronos. Resultados experimentais utilizando o indicador de mèrito proposto, em problemas exemplo, foram apresentados e discutidos, enfatizando-se a vantagem do método CS no que concerne ao Indicador de Mérito considerado.

Capítulo 3

Team Algorithm Generalizado

A solução de sistemas de grande porte, particionando um dado problema em subproblemas menores, de forma que em cada subproblema possa ser utilizado um método numérico diferente, dependendo das características próprias de cada subproblema, foi discutida no capítulo anterior, no que convencionamos em denominar *bloco Team Algorithm*. Na análise realizada, partimos da premissa de que para cada subproblema, existe um algoritmo seqüencial conhecido capaz de resolvê-lo localmente. Neste capítulo, relaxamos esta hipótese e buscamos resolver problemas para os quais existem (um ou mais) subproblemas que não podem ser resolvidos de forma aceitável, por um único algoritmo trabalhando individualmente no subproblema. A alternativa é então, utilizar uma combinação de algoritmos diferentes para tentar resolver aqueles subproblemas que não podem ser resolvidos. Essa alternativa de solução não é limitada a subproblemas determinados, podendo servir inclusive para a resolução do problema global.

A idéia de utilizar diversos algoritmos, possivelmente com estruturas de dados diferentes, para resolver um mesmo problema (como um todo), não se mostra atraente no contexto de computação seqüencial, pois isto geralmente implica no uso ineficiente de recursos computacionais. Já num contexto paralelo, esta visão pode cambiar radicalmente. Como exemplo, consideremos o tempo de processamento por iteração: enquanto em um sistema seqüencial deveriamos executar alternadamente um algoritmo por vez, aumentando consideravelmente o tempo de processamento por iteração; no contexto paralelo, processadores diferentes podem executar os diversos algoritmos em forma simultánea sem que isto implique necessariamente em um gasto adicional de tempo.

Vemos assim que o desenvolvimento da tecnologia dos sistemas distribuídos na década de 80, que buscava alguma forma de aproveitar o crescente número de processadores disponíveis nos sistemas distribuídos, levou naturalmente a propor a utilização de algoritmos diferentes, executados em forma simultánea nos diversos processadores do sistema, com o objetivo de resolver um mesmo problema com melhor desempenho e em menor tempo. As primeiras propostas neste sentido, foram dadas por Talukdar et al. (1982 e 1983). A idéia principal destes trabalhos foi resolver um dado problema

algébrico, utilizando dois (ou mais) algoritmos executados em processadores diferentes, coordenados por meio de um processo chamado de *Administrador* que utiliza uma memória global, denominada *blackboard*, como meio de comunicação entre os processos. A combinação assim obtida, denominada no trabalho de Talukdar et al. (1983) como *Team Algorithm*, consegue resolver problemas algébricos que nenhum dos algoritmos sendo combinados, resolve individualmente de forma aceitável.

A seção 3.1 apresenta o Team Algorithm proposto em Talukdar et al. (1983), formalizando o mesmo de forma a investigar a convergência desta primeira proposta, que é vista na seção 3.2 sob uma perspetiva de 'overlapping' total. Na seção 3.1 também discutem-se diversas políticas para implementar um *Administrador* e se apresentam resultados experimentais de simulações que encorajam a utilização dos TA (veja também Falcão e Barán, 1992).

A desvantagem da proposta original de Talukdar et al. (1982 e 1983) em termos de 'speedup' é a ineficiência introduzida pela necessidade que cada algoritmo tem de resolver o problema todo, sendo que para muitos casos práticos, é suficiente utilizar *'overlapping' parcial* das partes críticas do problema considerado. Postula-se assim a utilização do conceito de 'overlapping' parcial (ou 'partial overlapping'), baseado nos trabalhos de Ikeda e Šiljak (1980, 1981, 1984 e 1987). A idéia é de simplesmente duplicar as equações mais críticas do problema a ser resolvido, de forma que a decomposição do sistema determine quais algoritmos diferentes, resolvam as mesmas equações, quando estas são críticas, e combinem os resultados parciais por meio de um *Administrador*.

O algoritmo assim construido é denominado *Team Algorithm Generalizado*, ou simplesmente *Team Algorithm* (TA), pelo fato de generalizar vários métodos usados na resolução de sistemas algébricos de equações, como por exemplo: os *bloco TA* discutidos no Capítulo 2, as propostas de Talukdar et al. (1982 e 1983) assim como o 'partial overlapping' proposto em Ikeda e Šiljak (1980).

A introdução do conceito de *Team Algorithm Generalizado*, assim como diversas versões do mesmo para um contexto distribuído assíncrono, são apresentados na seção 3.2 juntamente com exemplos ilustrativos que enfatizam as condições nas quais o *TA Generalizado* apresenta suas principais vantagens, sob o ponto de vista de convergência. A formalização dos TA Generalizados, assim como os correspondentes estudos de convergência baseados no Teorema 2.1, são apresentados na seção 3.3 para um modelo geral. Finaliza-se o capítulo, apresentando na seção 3.4, uma comparação entre diversas versões de Team Algorithms, com o objetivo de dar algum critério para a escolha de uma boa implementação, para um dado problema.

3.1 Team Algorithm com Administrador

Muitos dos problemas computacionais conhecidos podem ser resolvidos por métodos bem diversos, cada qual com suas vantagens próprias. A escolha do melhor método nem sempre é simples, pois para qualquer escolha, um outro método pode ter algumas propriedades mais desejáveis. Desta forma, parece natural a idéia de combinar algoritmos, de forma a obter as propriedades desejadas.

Nesta seção, introduzimos o conceito de Team Algorithm com Administrador, o que permite resolver sistemas de equações algébricas, utilizando uma combinação de algoritmos iterativos diferentes, que podem estar resolvendo 'simultaneamente' as mesmas equações, e cujos resultados parciais são combinados num processo especial, chamado de *Administrador*. A introdução e posterior análise do Administrador assim como de suas possíveis variantes, nesta seção, permite enfatizar as vantagens potenciais de sua utilização; e serve como introdução dos *Team Algorithms Generalizados*, que serão estudados nas seções que seguem.

Os primeiros trabalhos que postulam a utilização de métodos diferentes num contexto paralelo assíncrono (Talukdar et al., 1982 e 1983), propõem resolver o sistema de equações algébricas (2.1) dado por:

$$\Phi(x) \;=\; 0 \tag{3.1}$$

combinando as excelentes qualidades de convergência do algoritmo de Newton-Raphson (ou NR) com uma variante do método do Gradiente (ou G), denominado nessa proposta 'Steepest Descent method'. No referido trabalho postula-se a combinação dos algoritmos citados, usando uma forma de soma ponderada dos incrementos dados a partir de cada um dos métodos, conforme a equação:

$$x(k+1) \;=\; x(k) \;+\; w_1(k)\,\Delta x_1(k) \;+\; w_2(k)\,\Delta x_2(k) \tag{3.2}$$

onde os $\Delta x_i(k)$, $(i = 1, 2)$ são os incrementos calculados a partir de cada método, na iteração k; enquanto $w_1(k)$ e $w_2(k)$ são *pesos* não negativos. Em particular, para a escolha de algoritmos acima mencionada temos:

$$\Delta x_1(k) \;=\; \Delta x_{NR}(k) \;=\; -\Big[J_\Phi(k)\Big]^{-1}\,\Phi\Big(x(k)\Big) \tag{3.3}$$

$$\Delta x_2(k) \;=\; \Delta x_G(k) \;=\; -\gamma\Big[J_\Phi(k)\Big]^{T}\,\Phi\Big(x(k)\Big) \tag{3.4}$$

onde: $J_\Phi(k) = \left\{\frac{\partial \Phi_i}{\partial x_j}\right\}$ é o Jacobiano de $\Phi(x)$ calculado na iteração k, enquanto γ é um parâmetro de relaxação $(0 < \gamma < 1)$ escolhido convenientemente, de forma a assegurar a convergência do algoritmo.

A versão síncrona do algoritmo (3.2) é bem simples: um processo chamado de *Administrador* envia o valor inicial $x(k)$ da iteração em curso a cada um dos algoritmos, para que estes calculem os Δx_i em paralelo, e recebe os valores calculados a partir de cada algoritmo para combiná-los escolhendo os *pesos* adequados. O algoritmo continua

iterando até que o resíduo $E(x)$ seja menor que uma dada tolerância (ou erro aceitável) ε, onde:

$$E(x) \stackrel{\text{def}}{=} \| \Phi(x) \| \tag{3.5}$$

Discutimos em seguida a versão síncrona de um problema exemplo apresentado em Talukdar et al. (1983); enfatizando que versões assíncronas do algoritmo dado por (3.2) são também possíveis.

Exemplo 3.1 Seja $x \in \mathbb{R}^4$, resolver o problema $\Phi(x) = 0$ sendo que

$$\Phi(x) = \begin{bmatrix} \Phi_1(x) \\ \Phi_2(x) \\ \Phi_3(x) \\ \Phi_4(x) \end{bmatrix} = \begin{bmatrix} x_1 + 10x_2 \\ \sqrt{5}(x_3 - x_4) \\ (x_2 - 2x_3)^2 \\ \sqrt{10}(x_1 - x_4)^2 \end{bmatrix} \tag{3.6}$$

O primeiro passo para resolver este problema utilizando qualquer dos métodos citados, é calcular o Jacobiano $J_\Phi(x)$ da função $\Phi(x)$ que está dado por

$$J_\Phi(x) = \begin{bmatrix} 1 & 10 & 0 & 0 \\ 0 & 0 & \sqrt{5} & -\sqrt{5} \\ 0 & a & -2a & 0 \\ b & 0 & 0 & -b \end{bmatrix} \tag{3.7}$$

onde $a = 2(x_2 - 2x_3)$ e $b = 2\sqrt{10}(x_1 - x_4)$.

Pode ser verificado que o Jacobiano $J_\Phi(x)$ não é singular se:

$$\begin{cases} x_2 \neq 2x_3 \\ x_1 \neq x_4 \end{cases} \tag{3.8}$$

Satisfeitas as inequações dadas em (3.8), a inversa de $J_\Phi(x)$ pode ser facilmente calculada:

$$\left[J_\Phi(x) \right]^{-1} = \begin{bmatrix} \frac{1}{21} & \frac{-20}{21\sqrt{5}} & \frac{-10}{21a} & \frac{20}{21b} \\ \frac{2}{21} & \frac{2}{21\sqrt{5}} & \frac{1}{21a} & \frac{-2}{21b} \\ \frac{1}{21} & \frac{1}{21\sqrt{5}} & \frac{-10}{21a} & \frac{-1}{21b} \\ \frac{1}{21} & \frac{-20}{21\sqrt{5}} & \frac{-10}{21a} & \frac{-1}{21b} \end{bmatrix} \tag{3.9}$$

Estabelecidas as equações, estamos em condições de estudar algumas políticas possíveis de serem utilizadas e implementadas no *Administrador*, bem como verificar experimentalmente em que condições o TA apresenta vantagens.

3.1.1 Administrador utilizando Pesos Constantes

Analisamos aqui a mais simples das propostas para a escolha dos *pesos* a serem utilizados pelo *Administrador*: uma escolha 'a priori', que permite ao algoritmo convergir, mantendo $w_1(k)$ e $w_2(k)$ constantes ao longo da execução. O trabalho do *Administrador* se limita assim à comunicação com os algoritmos sendo combinados e à aplicação da equação (3.2) que portanto tem a forma:

$$x(k+1) \;=\; x(k) \;+\; w_1\,\Delta x_1(k) \;+\; w_2\,\Delta x_2(k) \tag{3.10}$$

Nota 3.1 Evidentemente, cada um dos algoritmos sendo combinados, pode ser visto como um caso particular do Team Algorithm com Administrador de pesos $(1,0)$ e $(0,1)$ respectivamente.

Apresentamos a seguir os resultados experimentais obtidos para diferentes condições iniciais $x(0)$ no Exemplo 3.1, usando a norma-2 para o cálculo do resíduo dado por (3.5) e uma tolerância de $\varepsilon = 0.01$.

Caso 1: $x(0) = [1;\quad 0;\quad 0;\quad 1]^T$

Para esta condição inicial, o algoritmo de Newton-Raphson não pode ser utilizado, pois as condições da inequação (3.8) não são satisfeitas e portanto o Jacobiano é singular. Por sua parte, o algoritmo do Gradiente não consegue convergir em menos de 200 iterações[1].

A alternativa é de se utilizar um *Team Algorithm* para aproveitar as excelentes características de convergência do método de Newton-Raphson. Desta forma, a utilização do método do Gradiente quando combinado com o Newton-Raphson, permite ao TA sair da região onde o método NR tem dificuldades (devido ao Jacobiano ser singular). Tomemos como exemplo o caso $w_1 = 0.8$ e $w_2 = 0.8$, e analisemos o que acontece com o TA em cada iteração:

$$\begin{array}{lcl}
x(0) = [\;\; 1 \;\; 0 \;\; 0 \;\; 1\;]^T & & \\
\Phi(0) = [\;\; 1.00 \;\; -2.24 \;\; 0.00 \;\; 0.00\;]^T & & E(0) = 6.000 \\
\\
x(1) = [\;\; 0.99 \;\; -0.10 \;\; 0.05 \;\; 0.95\;]^T & & \\
\Phi(1) = [\;\; -0.03 \;\; -2.01 \;\; 0.04 \;\; 0.01\;]^T & & E(1) = 4.033 \\
\\
x(2) = [\;\; 0.25 \;\; -0.02 \;\; 0.09 \;\; 0.18\;]^T & & \\
\Phi(2) = [\;\; 0.03 \;\; -0.20 \;\; 0.04 \;\; 0.02\;]^T & & E(2) = 0.042 \\
\\
x(3) = [\;\; 0.12 \;\; -0.01 \;\; 0.06 \;\; 0.07\;]^T & & \\
\Phi(3) = [\;\; -0.02 \;\; -0.02 \;\; 0.02 \;\; 0.01\;]^T & & E(3) = 0.001 < \varepsilon
\end{array}$$

[1] O método do Gradiente requer 266 iterações para $w_2 = 1$ com $\gamma = 0.01278$.

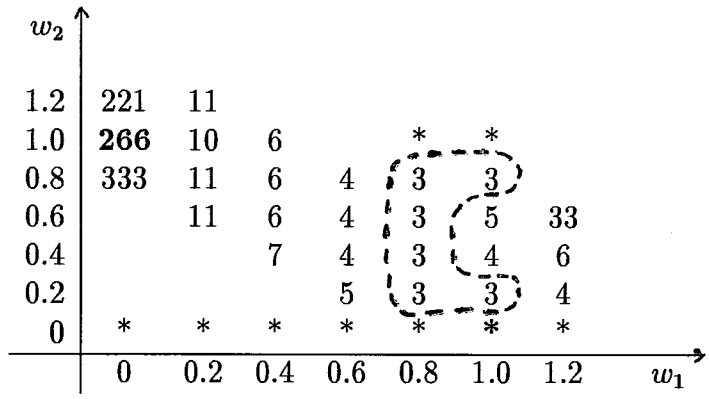

Figura 3.1: Mapa de Iterações para o Caso 1 do Exemplo 3.1.

Como o Jacobiano é singular na condição inicial, o método de Newton-Raphson não consegue calcular o incremento $\Delta x_1(0)$; conseqüentemente, o TA assume que $\Delta x_1(0) = 0$. Porém, o método do Gradiente consegue calcular um incremento $\Delta x_2(0) \neq 0$. O resultado disto é que o algoritmo combinado não anda muito rápido, mas o suficiente para sair da região onde o Jacobiano é singular. Na seguinte iteração, todo o potencial do Newton-Raphson entra em ação e o resíduo é reduzido em duas ordens de grandeza. O algoritmo converge finalmente na terceira iteração!.

Nota 3.2 O exemplo é suficentemente alentador para ilustrar as vantagens potenciais do Team Algorithm, que consegue resolver o problema em poucas iterações enquanto cada método individualmente, ou não converge, ou o faz em um número grande de iterações.

Para investigar o comportamento do algoritmo em estudo, levantamos o número I de iterações necessárias até a convergência, para diferentes *pesos*. Os valores da função:

$$I = f(w_1; w_2) \quad > 0 \tag{3.11}$$

podem ser representados no 'plano $w_1 - w_2$' (Figura 3.1), no que denominaremos *Mapa de Iterações.*

Algumas características do *Mapa de Iterações* dado na Figura 3.1 podem-ser enfatizadas, por serem comuns na maioria dos mapas levantados. A primeira característica é a existência de uma *Região de Convergência Ótima* (RCO) na qual se tem o menor número de iterações possíveis ($I = 3$). A medida em que o ponto de trabalho $(w_1; w_2)$ se afasta desta região, o número de iterações I cresce. Outra característica interessante é a linha marcada '*', que representa valores de $(w_1; w_2)$ para os quais o algoritmo não converge, e que no entanto fica próxima à RCO.

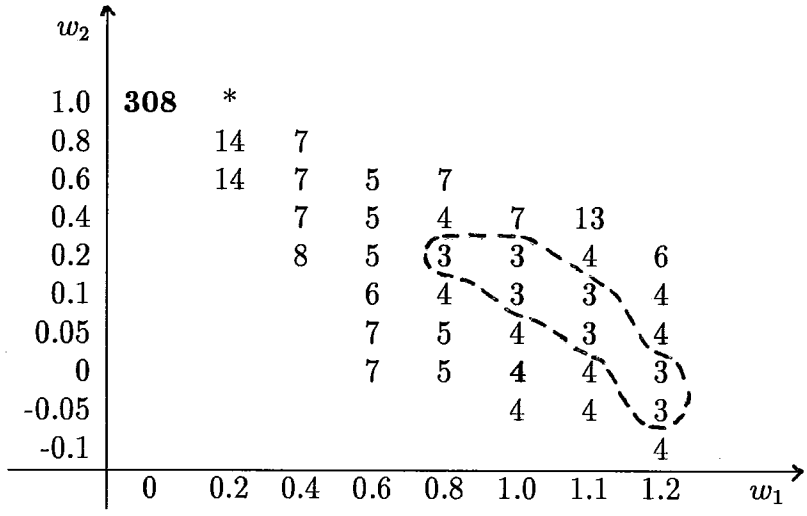

Figura 3.2: Mapa de Iterações para o Caso 2 do Exemplo 3.1.

Caso 2: $x(0) = [3; \quad -1; \quad 0; \quad 1]^T$

A Figura 3.2 apresenta o *Mapa de Iterações* para esta nova condição inicial $x(0)$. Mais uma vez, o TA converge em menos iterações que cada um de seus membros, pois o método do Gradiente converge em 308 iterações enquanto o Newton-Raphson em quatro. Este exemplo permite enfatizar uma característica interessante que é encontrada repetidas vezes em diversos problemas: a *Região de Convergência Ótima* inclui alguns pontos da reta $w_2 = 0$ que em geral ocorrem para $w_1 > 1$. Isto sugere que o algoritmo de Newton-Raphson pode ser muitas vezes acelerado usando um *Parâmetro de Sobre-Relaxação*[2] (ver Bertsekas e Tsitsiklis, 1989b; Varga, 1962; ou Ortega, 1988), tendo para muitos casos práticos um desempenho equivalente ao melhor TA com Administrador utilizando pesos constantes. Ainda mais, na grande maioria dos casos estudados, os pontos da RCO satisfazem a relação

$$w_1 + w_2 \geq 1$$

o que implica que um certo grau de sobre-relaxação *combinada* se mostra benéfico para os TA.

Para finalizar esta discusão de Administradores utilizando pesos constantes, se apresenta um último caso um tanto curioso, que ilustra a necessidade de pesquisar com cuidado esta família de algoritmos.

Caso 3: $x(0) = [0; \quad 1; \quad 1; \quad 0]^T$

A Figura 3.3 apresenta o *Mapa de Iterações* para esta nova condição inicial, que a semelhança do caso 1, não permite a utilização do método NR na primeira

[2]Idéia similar àquela utilizada nos algoritmos SOR (Successive Over-Relaxation).

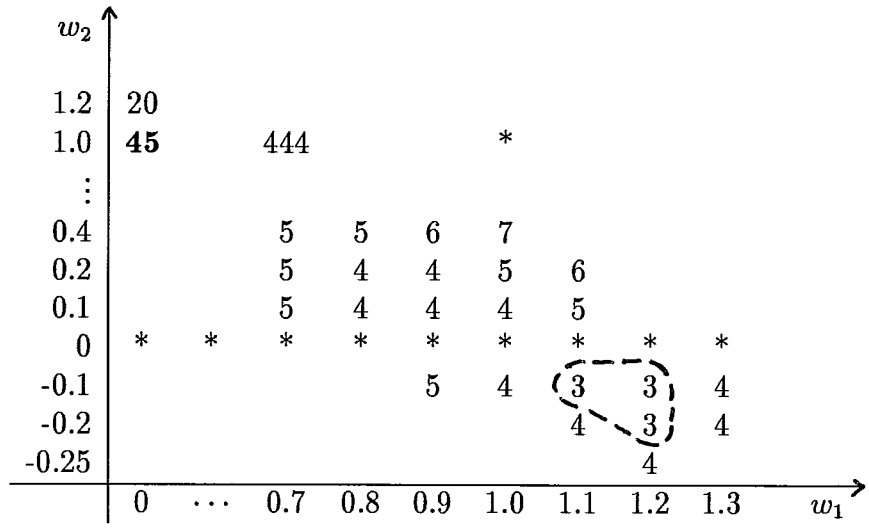

Figura 3.3: Mapa de Iterações para o Caso 3 do Exemplo 3.1.

iteração. Como já foi discutido, o cenário é típico para a aplicação do *Team Algorithm*. Não é surpreendente então que enquanto o Algoritmo do Gradiente requer dezenas de iterações, o TA consegue convergir em apenas três iterações.

A característica interessante deste caso, se situa na região do 'plano $w_1 - w_2$' onde o número de iterações I é mínimo, pois isto acontece para $w_2 < 0$ e $w_1 > 1$. Em outras palavras, a combinação ótima utiliza o Gradiente para diverger ligeiramente (peso negativo), de forma a permitir uma melhor convergência do Newton-Raphson. Uma situação que nos faz lembrar idéias do 'Simulated Annealing' (Kirkpatrick et al., 1983; Aragon et al., 1984; Mitra et al. 1986).

Para finalizar esta discussão dos TA com Administrador utilizando pesos constantes, derivamos uma condição suficiente de convergência para os mesmos, no contexto seqüencial. Para isto, representamos os algoritmos sendo combinados como operadores (contrativos) $\mathcal{G}_i$, com módulos L_i, i.e.,

$$\Delta x_i(k) \;=\; \mathcal{G}_i\big(x(k)\big) \;-\; x(k), \qquad \forall i \in \{1,2\} \tag{3.12}$$

com esta notação, o Team Algorithm dado por (3.10) pode ser representado por:

$$x(k+1) \;=\; c\,x(k) \;+\; w_1\,\mathcal{G}_1\big(x(k)\big) \;+\; w_2\,\mathcal{G}_2\big(x(k)\big) \tag{3.13}$$

onde

$$c \;=\; 1 \;-\; w_1 \;-\; w_2 \tag{3.14}$$

O TA assim representado é um operador (contrativo) $\mathcal{G}$ com módulo L dado por:

$$L \;=\; |\,c\,| \;+\; w_1\,L_1 \;+\; w_2\,L_2 \tag{3.15}$$

assim sendo, podemos escrever o seguinte corolário, derivado do Teorema 2.3:

COROLÁRIO 3.1 *o Team Algorithm dado por (3.10), ou de forma equivalente por (3.13), que satisfaz a Hipótese 2.1, converge para a única solução x^* em D se:*

$$|c| + w_1 L_1 + w_2 L_2 < 1 \qquad onde \quad c = 1 - w_1 - w_2 \tag{3.16}$$

Nota 3.3 A importância deste corolário está em assegurar a convergência do TA que conbina um algoritmo lento $\mathcal{G}_2$, de boa convergência ($L_2 < 1$), com um outro $\mathcal{G}_1$ bem mais rápido, mas que não possa assegurar a convergência por ter $L_1 \geq 1$, assumindo satisfeita a Hipótese 2.1. Efetivamente, é fácil verificar que é possível escolher os pesos w_1 e w_2, de forma a satisfazer a condição dada em (3.16). Como exemplo, para $L_2 < 1 \leq L_1$ podemos escolher:

$$w_1 = \epsilon; \qquad w_2 = 1 - \epsilon; \qquad onde \quad 0 < \epsilon < \frac{1 - L_2}{L_1 - L_2} \tag{3.17}$$

3.1.2 Administrador utilizando Pesos Variáveis

Com os pesos constantes, o trabalho do *Administrador* se limita à implementação da equação (3.2) e à comunicação com cada algoritmo. Pode-se questionar então, o que acontece se o *Administrador* tem 'inteligencia' suficiente como para escolher, a cada iteração, os melhores pesos segundo um dado critério. Desta forma, os *pesos* $w_1(k)$ e $w_2(k)$ da equação (3.2) são também função da iteração k.

Os critérios para a escolha 'inteligente' dos *pesos* podem ser vários. A idéia mais intuitiva parece ser a escolha, em cada iteração, dos pesos que minimizem o resíduo dado pela equação (3.5). O trabalho do *Administrador* consistiria em receber os $\Delta x_i(k)$ enviados pelos algoritmos a serem combinados e *calcular* os pesos que minimizem $E(\,x(k)\,)$. Sem discutir ainda a complexidade dos cálculos, e assumindo que eles podem ser realizados em tempo real, vejamos o desempenho do algorítmo para este caso.

Começemos com o caso 1 do Exemplo 3.1 e vejamos o que acontece a cada iteração quando minimizamos $E(x) = \|\Phi(x)\|_2$:

$$\begin{array}{lll}
x(0) = [\ 1 & 0 & 0 \quad 1\ ]^T & \\
\Phi(0) = [\ 1.00 & -2.24 & 0.00 \quad 0.00\ ]^T & E(0) = 6.000 \\
 & & & \\
x(1) = [\ 0.99 & -0.14 & 0.07 \quad 0.93\ ]^T & (w_1 = 0; \quad w_2 = 1.104) \\
\Phi(1) = [\ -0.43 & -1.92 & 0.08 \quad 0.01\ ]^T & E(1) = 3.876 \\
 & & & \\
x(2) = [\ -0.01 & -0.03 & 0.03 \quad 0.00\ ]^T & (w_1 = 1.11; \quad w_2 = -0.60) \\
\Phi(2) = [\ -0.29 & 0.07 & 0.01 \quad 0.00\ ]^T & E(2) = 0.087 \\
 & & & \\
x(3) = [\ 0.00 & 0.00 & 0.03 \quad 0.01\ ]^T & (w_1 = 0.34; \quad w_2 = 0.54) \\
\Phi(3) = [\ 0.01 & 0.04 & 0.00 \quad 0.00\ ]^T & E(3) = 0.002 < \varepsilon
\end{array}$$

O Administrador utilizando *pesos variáveis* permite a convergência em três iterações, o mesmo número que, quando o Administrador utilizou um peso constante ótimo, mas com o agravante de ter um resíduo maior após as três iterações, apesar do esforço adicional de recalcular os pesos a cada iteração.

Comparando os resíduos, vemos que efetivamente ele foi minimizado na primeira iteração. Porém, o *caminho* que segue o algoritmo quando utiliza pesos variáveis faz com que o resíduo não seja minimizado com a mesma velocidade do caso em que os pesos permanecem constantes.

Considerando agora o caso 2, exemplificamos que, com esta política de variação dos pesos, são necessárias cinco iterações para convergir, duas a mais que no caso de pesos constantes ótimos. O mesmo acontece com o caso 3, onde o TA que utiliza pesos variáveis necessita cinco iterações para convergir. Este desempenho, mais pobre, do algoritmo de pesos variáveis que minimiza $\|\Phi(x)\|_2$ se repete na maioria dos casos testados.

Ilustra-se a seguir que o desempenho melhora consideravelmente quando se utiliza a norma-∞ na definição do resíduo a ser minimizado. A escolha dos pesos é agora guiada pela minimização de $\|\Phi(x)\|_\infty$, mesmo que o critério de terminação continue sendo $\|\Phi(x)\|_2 < \varepsilon$. Vejamos a execução para o caso 1:

$$
\begin{array}{lll}
x(0) = [\ 1 \quad 0 \quad 0 \quad 1\ ]^T & & \\
\Phi(0) = [\ 1.00 \quad -2.24 \quad 0.00 \quad 0.00\ ]^T & & E(0) = 6.000 \\
\\
x(1) = [\ 0.97 \quad -0.26 \quad 0.13 \quad 0.87\ ]^T & & (w_1 = 0; \quad w_2 = 2.052) \\
\Phi(1) = [\ -1.65 \quad -1.65 \quad 0.28 \quad 0.03\ ]^T & & E(1) = 5.519 \\
\\
x(2) = [\ 0.14 \quad -0.01 \quad 0.12 \quad 0.09\ ]^T & & (w_1 = 1.04; \quad w_2 = 0) \\
\Phi(2) = [\ 0.06 \quad 0.06 \quad 0.06 \quad 0.01\ ]^T & & E(2) = 0.012 \\
\\
x(3) = [\ 0.07 \quad -0.01 \quad 0.05 \quad 0.05\ ]^T & & (w_1 = 1.20; \quad w_2 = -0.29) \\
\Phi(3) = [\ 0.01 \quad -0.01 \quad 0.01 \quad 0.00\ ]^T & & E(3) = 0.0003 < \varepsilon
\end{array}
$$

Pode-se notar que a convergência não ocorreu na segunda iteração por muito pouco, e de fato, foi com este critério que se obteve o menor resíduo depois de três iterações. Este melhor desempenho no caso da minimização de $\|\Phi(x)\|_\infty$ se repete em quase todos os casos testados. Assim, no caso 2, três iterações foram suficientes, enquanto no caso 3 que representa um caso muito especial, foram necessarias quatro iterações, uma a mais que no caso de pesos constantes ótimos.

Claramente, nestas propostas, o *Administrador* utiliza um tempo nada desprezível na escolha dos *pesos*, pois os algoritmos de minimização do resíduo não são em geral triviais (ver algoritmos de minimização em Hageman e Young, 1981; Ortega, 1988; ou Bertsekas e Tsitsiklis, 1989b). Portanto, não são evidentes as vantagens de utilizar algoritmos de minimização no Administrador, na maioria dos problemas estudados.

Outras políticas de minimização foram também relatadas em Falcão e Barán (1992), como a minimização de um *Erro Linear* E_r definido como:

$$E_r \stackrel{\text{def}}{=} \| x(k+1) - x^* \|_p \tag{3.18}$$

mas em geral, os resultados são similares, i.e. a favor de um Administrador simples utilizando pesos constantes, pois a utilização de algoritmos de minimização complexos, não apresentaram vantagens suficientes para justificar a sua utilização, justificando-se somente nos casos que a *Região de Convergência Otima* não possa ser estimada 'a priori'. Quando isto acontece, resultados experimentais indicam que a minimização de normas infinitas $\| \cdot \|_\infty$ são as mais recomendáveis.

Até aqui consideramos exclusivamente o problema do Exemplo 3.1, pois ele é suficiente para introduzir os principais conceitos a serem utilizados na formalização do *Team Algorithm Generalizado.* O leitor interessado em um estudo mais detalhado das idéias até aqui apresentadas, pode consultar Falcão e Barán (1992), trabalho baseado em Barán (1991). No referido trabalho, os TA são utilizados para resolver o problema do *Fluxo de Potência Elétrica*, que será discutido no próximo capítulo. O referido trabalho resolve nove problemas exemplos, quatro dos quais são problemas tipo da IEEE; apresentando resultados experimentais que mostram que em 8 dos 9 problemas testados, alguma versão do *Team Algorithm* consegue resolver o problema, com menos iterações ou menor resíduo de potência que o melhor dos algoritmos sendo combinados (o Desacoplado Rápido). Mapas de Iterações para cada um dos 9 problemas exemplo são apresentados em Barán (1991), junto com outras discussões que basicamente coincidem com a presente.

Do ponto de vista da análise de convergência, pode-se derivar uma condição suficiente para o algoritmo dado por (3.2), similar à dada pelo Corolário 3.1. Para isto, re-escreve-se (3.2) de forma a enfatizar que o TA combina algoritmos representados por operadores $\mathcal{G}_i$ de módulos L_i, a exemplo de (3.13):

$$x(k+1) \;=\; c(k)\,x(k) \;+\; w_1(k)\,\mathcal{G}_1\Big(x(k)\Big) \;+\; w_2(k)\,\mathcal{G}_2\Big(x(k)\Big), \tag{3.19}$$

onde $c(k) = 1 - w_1(k) - w_2(k)$. Assim, considerando que os pesos não negativos $(w_1(k), w_2(k) \geq 0)$, e aplicando o Teorema 2.3, temos:

COROLÁRIO 3.2 *o Team Algorithm dado por (3.2), ou de forma equivalente por (3.19), que satisfaz a Hipótese 2.1, converge para a única solução x^* em D se:*

$$|c(k)| \;+\; w_1(k)\,L_1 \;+\; w_2(k)\,L_2 \;<\; 1 \qquad \textit{onde} \quad c(k) = 1 - w_1(k) - w_2(k).$$

Concluimos esta discussão enfatizando a capacidade dos TA de aproveitar propriedades dos algoritmos sendo combinados. Assim, a velocidade de certos algoritmos que não conseguem convergir numa região de interesse (como o Newton-Raphson no Exemplo 3.1) pode ser aproveitada quando estes algoritmos são combinados com outros métodos com boas características de convergência, mesmo que mais lentos (como o método do Gradiente no exemplo citado).

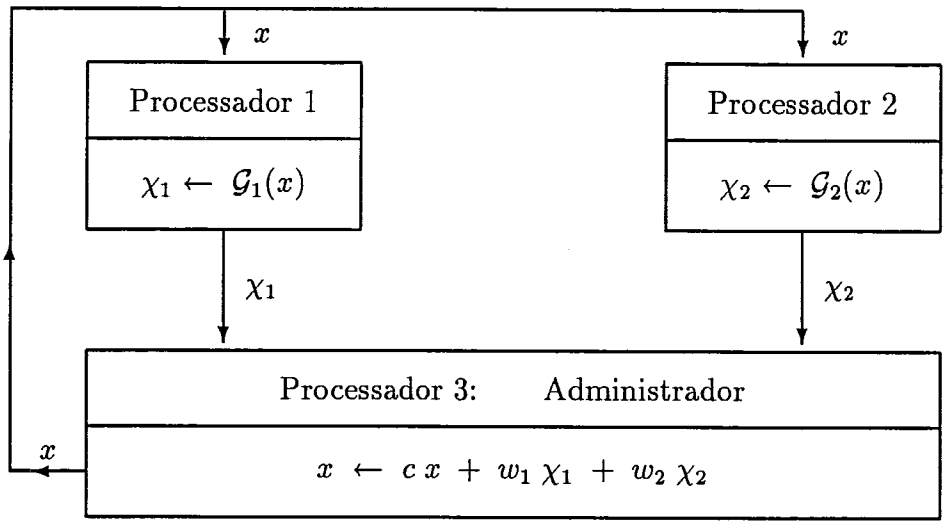

Figura 3.4: Implementação paralela de um TA com Administrador.

3.1.3 Administrador Assíncrono

Na implementação da versão paralela-síncrona, do *Team Algorithm com Administrador* estudado nesta seção, é natural utilizar três processadores, como ilustra a Figura 3.4; o processador 1 encarregado de executar o método NR, que representamos por um operador $\mathcal{G}_1$, o processador 2 encarregado do método do Gradiente com um operador $\mathcal{G}_2$, e o processador 3, implementando o Administrador, que se comunica com os dois primeiros e implementa a equação (3.2). Por simplicidade na análise que segue, consideramos o TA com Administrador utilizando pesos constantes. Sendo assim, as iterações do TA paralelo-síncrono, podem ser matematicamente representadas pelas equações:

$$\begin{aligned} \chi_1(k+1) &= \mathcal{G}_1\big(x(k)\big) \\ \chi_2(k+1) &= \mathcal{G}_2\big(x(k)\big) \\ x(k+1) &= c\,x(k) + w_1\,\chi_1(k+1) + w_2\,\chi_2(k+1) \end{aligned} \tag{3.20}$$

onde $c = 1 - w_1 - w_2$, e $\chi_i(k+1) \in \mathbb{R}^n$ representa o valor de $x \in \mathbb{R}^n$ calculado, na iteração k, pelo processador i.

De forma similar, a versão paralela-assíncrona do TA com Administrador utilizando pesos constantes (ilustrada na Figura 3.4), pode ser matematicamente representada pelas equações:

$$\begin{aligned} \chi_1(k+1) &= \mathcal{G}_1\big(x^1(k)\big) \\ \chi_2(k+1) &= \mathcal{G}_2\big(x^2(k)\big) \\ x(k+1) &= c\,x(k) + w_1\,\chi_1^3(k) + w_2\,\chi_2^3(k) \end{aligned} \tag{3.21}$$

onde lembramos que pela notação introduzida em (2.12), $x^i(k)$ representa o valor mais atualizado de x disponível no processador i, na iteração k. Conseqüentemente,

$\chi_i^3(k)$, $(i = 1, 2)$, representa o valor mais atualizado de χ_i na iteração k, no processador 3 (o Administrador).

O TA assíncrono dado por (3.21), funciona de seguinte forma: os processadores 1 e 2, se limitam a receber o vetor x enviado pelo Administrador e aplicar o operador $\mathcal{G}_i$ respectivo, para logo transmitir o novo valor de χ_i ao Administrador. Por sua vez, este recebe valores atualizados de χ_i e realiza constantemente os calculos correspondentes, atualizando o valor de x que logo será transmitido aos outros processadores. Outra variante do Administrador assíncrono sería atualizar x somente após receber algum valor novo de χ_i.

Para estudar a convergência do TA assíncrono dado por (3.21), definimos um *vetor de estado expandido* W como:

$$W = \begin{bmatrix} W_1 \\ W_2 \\ W_3 \end{bmatrix} \stackrel{\text{def}}{=} \begin{bmatrix} \chi_1 \\ \chi_2 \\ x \end{bmatrix} \qquad \text{onde} \quad \begin{cases} W_1 = \chi_1 \\ W_2 = \chi_2 \\ W_3 = x \end{cases} \tag{3.22}$$

conseqüentemente, a equação (3.21) pode ser re-escrita utilizando a notação do vetor de estado expandido:

$$\begin{bmatrix} W_1(k+1) \\ W_2(k+1) \\ W_3(k+1) \end{bmatrix} = \begin{bmatrix} \mathcal{G}_1\big(W_3^1(k)\big) \\ \mathcal{G}_2\big(W_3^2(k)\big) \\ c\,W_3(k) \;+\; w_1\,W_1^3(k) \;+\; w_2\,W_2^3(k) \end{bmatrix} \tag{3.23}$$

Enfatizamos que o TA assíncrono dado por (3.23), pode ser considerado como um algoritmo bloco iterativo assíncrono; conseqüentemente, o Teorema 2.1 pode ser utilizado para obter uma condição suficiente de convergência para o mesmo.

Considerando os operadores $\mathcal{G}_i$ como contrativos em D, com módulos L_i, a matriz de comparação do algoritmo bloco iterativo assíncrono, representado por (3.23), é dada por:

$$H = \begin{bmatrix} 0 & 0 & L_1 \\ 0 & 0 & L_2 \\ w_1 & w_2 & |c| \end{bmatrix} \tag{3.24}$$

Derivada a matriz de comparação correspondente ao TA representado por (3.21), ou de forma equivalente por (3.23), podemos escrever o seguinte corolário derivado do Teorema 2.1:

COROLÁRIO 3.3 *Sob as hipóteses 2.1 e 2.2, adequadamente reformuladas para considerar o vetor de estado extendido W (em lugar de x), a versão assíncrona do Team Algorithm com Administrador utilizando pesos constantes, dado por (3.21), ou de forma equivalente por (3.23), converge para a única solução x^* em D, se $\rho(H) < 1$, onde H é dado por (3.24).*

Enfatizamos aqui que a Nota 3.3 é perfeitamente aplicável ao TA assíncrono, considerando que a escolha de pesos dada na equação (3.17), para o caso em que $L_2 < 1 \leq L_1$, também assegura que $\rho(H) < 1$.

A implementação do TA assíncrono dado por (3.21), é bem mais flexível que a correspondente versão síncrona, dado por (3.20), pois permite que o sistema continue trabalhando mesmo quando um dos algoritmos, digamos $\mathcal{G}_1$, seja bem mais demorado que o outro. Contudo, a estrutura do algoritmo, que trabalha com todo o vetor x em cada processador, não permite toda a flexibilidade dos *bloco TA* estudados no Capítulo 2, devido à necessidade dos processadores receberem alguma informação para poderem realizar um trabalho útil. Como exemplo, se o Administrador for muito lento (por exemplo, por estar utilizando um algoritmo de minimização), nenhum trabalho útil será realizado nos processadores 1 e 2 até que recebam um valor atualizado de x. Além disso, em Souza e Talukdar (1991) se discute a possibilidade da utilização de novos algoritmos na combinação. Neste caso, o Administrador deveria ser 're-projetado', pois a escolha de pesos deve ser adaptada à nova implementação.

Esta falta de flexibilidade do TA assíncrono com Administrador, dado por (3.21), levou à formulação dos *A-teams* com Administrador distribuído, por Souza e Talukdar (1991). Nesse trabalho, propõe-se eliminar o Administrador como processador independente e distribuir as funções administrativas entre os processadores, por meio de um encapsulamento (shell). Nesta proposta de Souza e Talukdar, uma memória global é utilizada como mecanismo de comunicação entre os processadores. O A-team assim construido, ilustrado na Figura 3.5, trabalha da seguinte forma: cada processador executa iterativamente o algoritmo a ele atribuido. O valor do resíduo $E(x)$ é monitorado em cada processador para verificar os progressos obtidos. Se os algoritmos conseguem obter melhores estimativas da solução, periodicamente esses valores são escritos na memória compartilhada e os algoritmos continuam trabalhando. Se algum algoritmo não consegue progredir, uma nova estimativa x da solução é lida da memória compartilhada (exemplo: o melhor ponto disponível), e o algoritmo é re-inicializado a partir deste novo ponto.

Note que o A-team assim construido convergiria sem problemas para o Exemplo 3.1, pois mesmo que o Jacobiano $J_\Phi(\, x(0)\,)$ seja singular, o método do Gradiente consegue avançar e escrever na memória compartilhada um novo valor de x, que uma vez lido pelo processador que executa o método NR, pode servir para que este resolva o problema em poucas iterações. Nesta implementação, a função administrativa se limita a decidir, em cada processador, quando (ou em que condições) ler ou escrever na memória compartilhada.

Esta discusão sobre os A-teams permite comprender que a grande vantagem desta combinação assíncrona de algoritmos, que não utiliza um processo Administrador explícito, é a possibilidade de convergir em regiões bem mais amplas que qualquer das regiões nas quais poderia trabalhar cada algoritmo individualmente, com a possibilidade adicional de, eventualmente, poder encontrar mais de uma solução (caso exista). Em contrapartida, quando estamos interessados numa única solução, somente

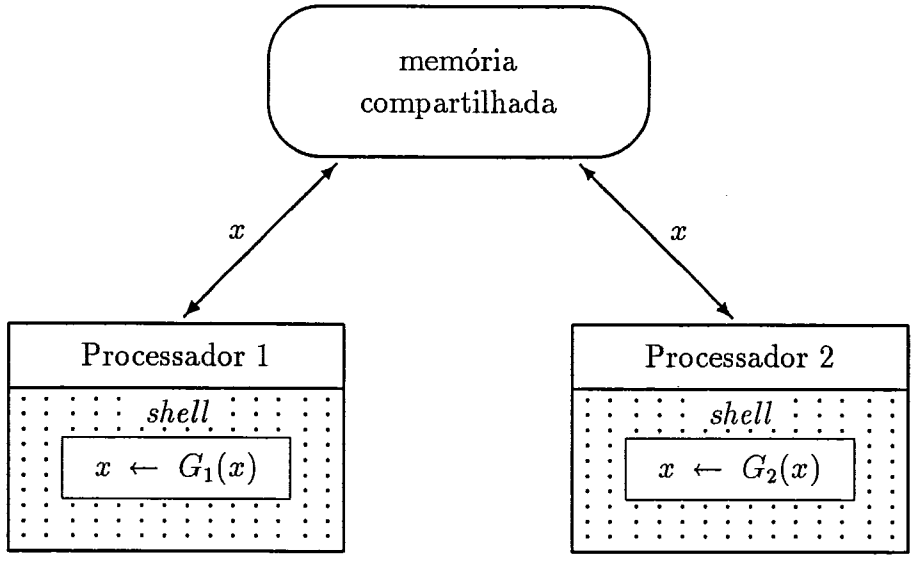

Figura 3.5: Implementação paralela de um A-team.

um processador está fazendo algum trabalho realmente útil por vez, o que leva a prejudicar enormemente a aceleração para problemas mais simples (na realidade existe a possibilidade de 'slow down').

Enfatizamos a facilidade com que os A-teams podem ser ampliados incorporando novos algoritmos, pois cada processador somente está em comunicação com a memória compartilhada e portanto é independente dos outros processadores. Esta independência com que cada processador trabalha em forma assíncrona, sem nenhum *controle central*, levou a comparar este tipo de organização com uma sociedade de insetos, capaz de criar grandes obras a pesar dos rudimentários *'entes'* que constituem a sociedade, assim como comparar-lo com a sofisticada sociedade de scientistas, capaz de produzir os avanços científicos de que somos testemunhas, postulando-se assim a potencialidade dos A-teams de obter um efeito *sinergético* (Talukdar et al., 1991 e 1992).

Note que a implementação de um de A-team em sistemas computacionais de memória distribuída, é também possível. Para isto, é suficiente que cada processador utilize uma memória local, e se comunique com os outros processadores via transmissão de mensagens, em lugar de utilizar a memória compartilhada.

Para finalizar esta seção, enfatizamos que um Team Algorithm com Administrador consegue combinar algoritmos diferentes de forma a aproveitar as características de cada um. Assim, no exemplo 3.1, foi possível aproveitar a velocidade de convergência do método NR, mesmo numa região onde ele não converge, quando combinado com o método do Gradiente que consegue convergir nessa região, mas de forma muito lenta. Contudo, a utilização do vetor todo na combinação, não se mostra muito flexível, e não

permite viabilizar a resolução de problemas de grande porte, em sistemas computacionais de memória distribuída, a diferença dos bloco TA estudados no Capítulo 2. A possibilidade de combinar as vantagens dos TA com Administrador e dos bloco TA do Capítulo 2, é discutida na proxima seção, onde se apresentam os *Team Algorithms Generalizados*, como uma alternativa capaz de combinar as características das propostas até aqui estudadas.

3.2 Team Algorithm Generalizado

Os Team Algorithms com Administrador discutidos na seção anterior apresentam a interessante característica de poder aproveitar as propriedades dos algoritmos sendo combinados, quando estes operam resolvendo o mesmo conjunto de equações. Assim, foi apresentado o Exemplo 3.1, no qual a velocidade de um algoritmo que não necessariamente converge na região de interesse é aproveitada quando o algoritmo é combinado, por meio de um Administrador, com um outro que pode assegurar a convergência nessa região. Contudo, outras vantagens próprias do particionamento utilizado pelos *bloco TA* do Capítulo 2, como a aceleração que se obtem resolvendo em paralelo subproblemas menores, ou a viabililidade de resolver problemas de grande porte em computadoras de memória distribuída, estão ausentes no TA com Administrador discutido na Seção 3.1, devido à atualização do vetor *todo*, em cada processador (e por cada algoritmo).

A idéia central do presente trabalho é propor, formalizar e analisar o conceito de *Team Algorithm Generalizado*, que por simplicidade continuaremos chamando de TA, que reuna as vantagens próprias de cada uma das combinações de algoritmos até aqui estudadas. Assim, para manter as propriedades de boa aceleração bem como da viabilização na resolução de problemas de grande porte em sistemas distribuídos, se propõe manter as partições dos *bloco TA* do Capítulo 2, com a possibilidade de duplicar (triplicar, etc.) as equações mais críticas (um conceito que será esclarecido abaixo), de forma que, somente estas sejam resolvidas por algoritmos distintos implementados em processadores diferentes, com o gerenciamento de um Administrador. Desta forma, assegura-se que o *Team Algorithm Generalizado* mantenha as propriedades do TA com Administrador, sem perder as vantagens dos bloco TA.

Efetivamente, pode se notar no Exemplo 3.1 que o algoritmo do Gradiente não precisa modificar as quatro componentes de x para que o Jacobiano deixe de ser singular, pois resolvendo somente para três componentes de x, o método NR já terá condições de prosseguir. Vemos então que é possível duplicar duas equações, e resolver o novo sistema de dimensão seis (em lugar de quatro), com uma partição tal que três equações sejam resolvidas pelo método do Gradiente e as outras três pelo método NR (enfatizando que as equações *repetidas* a serem combinadas pelo Administrador devem ser resolvidas tanto por um quanto pelo outro algoritmo).

A idéia de *replicar* certas equações para facilitar a resolução de alguns problemas, foi primeiramente proposta em Ikeda e Šiljak (1980), no contexto dos sistemas

dinâmicos, com o nome de 'overlapping decomposition'. Uma generalização destas decomposições com 'overlapping', é apresentada em outro trabalho de Ikeda e Šiljak (1981), enquanto uma discussão mais detalhada se encontra em Šiljak (1981).

O conceito de 'overlapping' é generalizado no *Princípio de Inclusão* para sistemas dinâmicos, em Ikeda et al. (1984). Variantes do tema são novamente tratadas em uma série de artigos como os de Ohta e Šiljak (1985), Hodzic e Šiljak (1986) e Ikeda e Šiljak (1987). Uma aplicação em estudos de estabilidade de redes elétricas, no contexto dos métodos numéricos, é encontrada em Brucoli et al. (1987).

A comparação entre uma partição simples, como aquela utilizada com os bloco TA, e uma partição com 'overlapping' parcial (replicação de algumas equações), similar à que será utilizada com os TA generalizados, é analizada, para o caso linear, em Calvet e Titli (1989); neste trabalho também são apresentados resultados numéricos para o caso de redes elétricas, enfatizando as vantagens do 'overlapping' parcial.

A consideração de 'overlapping' parcial num algoritmo de decomposição (partição de um problema em subproblemas) é apresentada em Sezer e Šiljak (1991), permitindo obter uma decomposição com 'overlapping'. Recentemente, um trabalho de Zečević e Šiljak (1992) combina técnicas de 'overlapping' parcial e decomposição para resolver (2.1) num contexto paralelo síncrono, utilizando uma variação do método de Newton simplificado (SN).

Enfatiza-se que a novidade de nossa proposta na forma de utilizar 'overlapping' parcial, com respeito aos trabalhos acima citados, não é a duplicação de algumas equações ('partial overlapping'), mas o uso do Administrador para gerenciar aproximações diferentes de uma mesma componente.

3.2.1 Introdução dos Team Algorithms Generalizados

No caso dos *bloco TA* estudados no Capítulo 2, foi assumido que a partição do problema era conhecida, e nos limitamos a estudar um problema com sua correspondente partição. Mantendo a mesma linha de trabalho, assumimos conhecida a partição em *blocos* do problema a ser resolvido, sendo que cada subvetor, correspondente a cada bloco, pode ser atualizado por um dos métodos disponíveis, ou por vários métodos combinados, utilizando um Administrador.

Para facilitar a leitura, repetimos a continuação a formulação matemática do problema, dada pelas equações (2.1) a (2.6) do Capítulo 2. Seja o sistema de equações algébricas

$$\Phi(x) = 0, \qquad x \in \mathbb{R}^n, \qquad \Phi : \mathbb{R}^n \rightarrow \mathbb{R}^n \tag{3.25}$$

definido num domínio $D \subset \mathbb{R}^n$, e seja a decomposição cartesiana de $\mathbb{R}^n$ dada por:

$$\mathbb{R}^n = \mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_m}, \qquad n_1 + \cdots + n_m = n. \tag{3.26}$$

e ademais $D \subset \mathbb{R}^n$ um domínio tal que

$$D = D_1 \times \cdots \times D_m, \qquad D_i \subset \mathbb{R}^{n_i}, \quad \forall\, i \in \{1, \ldots, m\} \tag{3.27}$$

então, um vetor $x \in D$ pode ser particionado de acordo com:

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}, \qquad x_i \in D_i, \qquad \forall\, i \in \{1, \ldots, m\} \tag{3.28}$$

Desta forma, $\Phi(x)$ pode ser particionado de acordo com:

$$\Phi(x) = \begin{bmatrix} \Phi_1(x) \\ \vdots \\ \Phi_m(x) \end{bmatrix}, \qquad \Phi_i : D \to \mathbb{R}^{n_i} \tag{3.29}$$

conseqüentemente, a equação a ser resolvida é dada por:

$$\Phi_i(x) = 0, \qquad \forall\, i \in \{1, \ldots, m\} \tag{3.30}$$

Para representar a informação de quais algoritmos devem atualizar cada subvetor x_i, e quais subvetores são atualizados em cada processador por seu correspondente método, definimos a *Matriz de Atribuição* $\Psi = \{\psi_{ij}\}$ do TA Generalizado correspondente.

DEFINIÇÃO 3.1 (MATRIZ DE ATRIBUIÇÃO)

Seja um Team Algorithm com:

- *uma partição do vetor x dada por (3.28);*
- *p métodos possivelmente diferentes, representados por seus respectivos operadores $G_1, \ldots, G_p$, implementados em até p processadores, onde $m \geq p$;*

A esse Team Algorithm associamos uma matriz $\Psi = \{\psi_{ij}\}$, com elementos binários $\psi_{ij} \in \{0, 1\}$, de dimensão $(m \times p)$, onde ψ_{ij} indica se o subvetor x_i é ou não atualizado pelo operador G_j. A matriz Ψ assim definida, é denominada Matriz de Atribuição do Team Algorithm. Essa matriz de atribuição é ordenada de forma que as primeiras r linhas, e somente elas, representem os subvetores atualizados por somente um algoritmo.

Dado que a partir da definição da Matriz de Atribuição, podemos construir Team Algorithms com diferentes graus de 'overlapping', nos referiremos a essa matriz como *Matriz de Atribuição associada a um TA Generalizado.* Para ilustrar o conceito da Matriz de Atribuição associada a um TA Generalizado, consideramos o exemplo que segue:

Exemplo 3.2 Seja um TA que combina dois algoritmos diferentes ($p = 2$), que representamos pelos operadores G_1 e G_2. Com este TA, queremos resolver um problema particionado em três blocos ($m = 3$), de forma que:

- x_1 seja atualizado somente por G_1,
- x_2 seja atualizado somente por G_2, e
- x_3 seja atualizado tanto por G_1 como por G_2.

O Team Algorithm do exemplo 3.2 tem associado uma *Matriz de Atribuição*:

$$\Psi = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \tag{3.31}$$

onde como exemplo, a coluna 1 de Ψ indica que G_1 atualiza x_1 e x_3, enquanto a linha 3 indica que x_3 é atualizado tanto por G_1 como por G_2. Note que neste exemplo $r = 2$ pois tanto x_1 quanto x_2, e somente eles, são atualizados por um único algoritmo (x_1 por G_1 e x_2 por G_2).

Lembramos aqui que o problema $\Phi(x) = 0$ a ser resolvido, satisfaz as equações (3.25) a (3.30) e portanto tem dimensão n, de acodo com a discussão da Seção 2.1. Porém, quando consideramos 'overlapping', podemos usar a definição de *dimensão expandida* $\bar{n}$ dada em Ikeda e Šiljak (1980), que considera o número total de equações sendo resolvidas, ou seja, inclui as equações replicadas com a correspondente multiplicidade. Como exemplo, para a matriz de atribuição (3.31) temos:

$$n = n_1 + n_2 + n_3 \qquad \text{e} \qquad \bar{n} = n_1 + n_2 + n_3 + n_3 = n + n_3 \; ;$$

isto porque o subproblema $\Phi_3(x) = 0$ é resolvido tanto por G_1 quanto por G_2.

Antes de considerar a implementação assíncrona do TA cuja matriz de atribuição associada é dada por (3.31), se apresentam a seguir uma série de características da *Matriz de Atribuição* Ψ.

Características da matriz de Atribuição Ψ

1. Todo subvetor x_i deve ser atualizado por no mínimo um algoritmo e no máximo pelos p algoritmos utilizados na combinação:

$$1 \leq \sum_{j=1}^{p} \psi_{ij} \leq p \qquad \forall i \in \{1, \ldots, m\}$$

2. Todo operador G_j deve atualizar no mínimo um subvetor e no máximo os m subvetores de x:

$$1 \leq \sum_{i=1}^{m} \psi_{ij} \leq m \qquad \forall j \in \{1, \ldots, p\}$$

3. O Team Algorithm com Administrador proposto em Talukdar et al. (1983), e discutido na seção 3.1, é um caso particular com 'overlapping' total no qual:

$$r = 0, \qquad m = 1, \qquad p = 2 \qquad \text{e} \qquad \Psi = [\,1 \quad 1\,]$$

4. O bloco TA discutido no Capítulo 2, é um caso particular no qual a Matriz de Atribuição é a matriz identidade (ou uma permutação desta). Trata-se do caso sem 'overlapping':

$$\Psi = I \qquad \text{sendo} \quad p = m = r$$

5. O número r de subvetores atualizados por um e somente um algoritmo satisfaz a relação:

$$0 \leq r \leq p \leq m \leq n$$

6. Se $r > 0$ se cumple que:

$$\forall i \in \{1, \ldots, r\} \qquad \sum_{j=1}^{p} \psi_{ij} = 1$$

7. Dada a dimensão n de um problema, a *dimensão expandida* $\bar{n}$, quando considerando 'overlapping', satisfaz a relação:

$$\bar{n} = \sum_{i=1}^{m} \sum_{j=1}^{p} \psi_{ij}\, n_i \quad \geq \quad n = \sum_{i=1}^{m} n_i$$

No que segue, consideramos exclusivamente o contexto parcialmente assíncrono que satisfaz a Hipótese 2.2. Neste contexto, para um TA Generalizado que tenha um subvetor x_i atualizado por mais de um processador, são disponíveis diferentes *versões* desse subvetor, segum seja o algoritmo que o atualiza. Por este motivo, denotamos como:

χ_{ij} ... à versão do subvetor x_i calculada pelo operador G_j;

e utilizamos a notação:

x_i ... para o subvetor calculado no Administrador a partir dos χ_{ij}.

Como exemplo, no TA assíncrono da Figura 3.6, com uma Matriz de Atribuição associada dada por (3.31), a versão do subvetor 3 (x_3) atualizada pelo processador 1 é denotada por χ_{31}, enquanto χ_{32} denota a versão atualizada por G_2. A semelhança do TA assíncrono estudado na seção anterior e dado por 3.21, o Administrador de pesos constantes é executado no processador 3, combinando desta vez os subvetores χ_{31} e χ_{31} de forma a gerar x_3 segum:

$$x_3(k+1) = c\,x_3(k) + w_1\,\chi_{31}(\,d_1^3(k)\,) + w_2\,\chi_{32}(\,d_2^3(k)\,) \tag{3.32}$$

onde $c = 1 - w_1 - w_2$. Enfatiza-se que $x_3,\ \chi_{31},\ \chi_{32} \in \mathbb{R}^{n_3}$, pois são versões diferentes do mesmo subvetor.

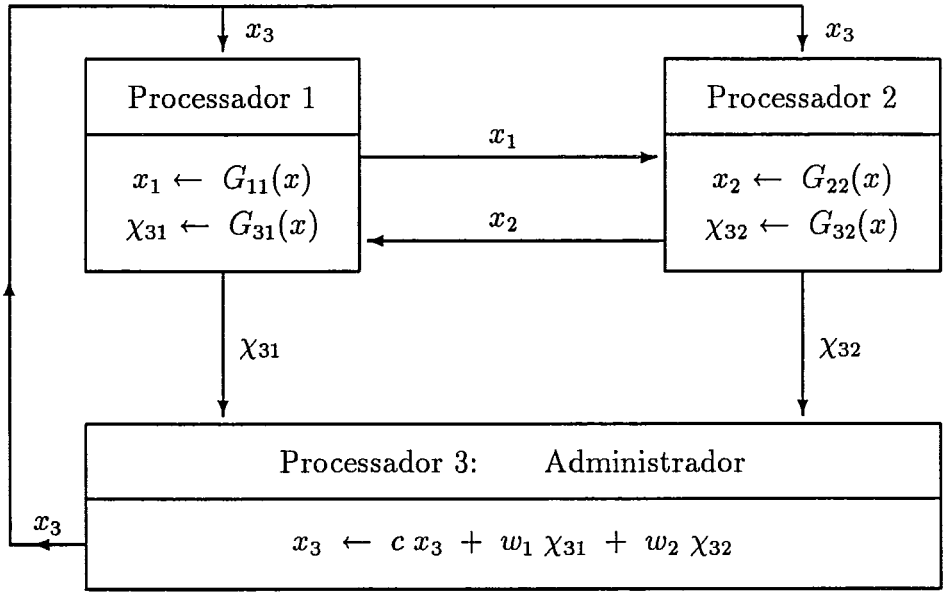

Figura 3.6: Implementação assíncrona da versão A de um TA.

Para escrevermos uma iteração do TA Generalizado, precisamos estender a notação dos operadores, introduzida em (2.7). Nessa oportunidade denotavamos por G_j ao operador executado no processador j, que atualiza o subvetor x_j. Como no contexto dos TA Generalizados, o operador G_j pode atualizar também um subvetor x_i, denotamos esta operação por G_{ij}. Assim, assumimos que o operador G_j executado no processador j pode ser representado por:

$$G_j(x) = \begin{bmatrix} G_{1j}(x) \\ \vdots \\ G_{mj}(x) \end{bmatrix}, \qquad G_{ij}(x) : D \rightarrow D_i, \quad \forall i \in \{1, \ldots, m\}, \quad \forall j \in \{1, \ldots, p\} \tag{3.33}$$

Com a notação introduzida, estamos em condições de estudar várias possíveis versões assíncronas de um *Team Algorithm Generalizado*, resumidas na Tabela 3.2 ao final do capítulo. Para isto, enfatizamos que o Administrador pode ser totalmente implementado num único processador, como no TA Generalizado da Figura 3.6 que implementa a equação (3.32) no processador 3. Este Administrador é chamado de *Administrador Centralizado*, para diferencia-lo do TA com *Administrador Distribuído*, no qual as funções administrativas estão repartidas entre os processadores do sistema computacional, e portanto não é possível identificar um único Administrador, a semelhança dos A-teams discutidos na subseção 3.1.3.

Para simplificar a apresentação dos conceitos fundamentais dos *Team Algorithms Generalizados*, no que segue consideramos exclusivamente os TA com Administradores utilizando pesos constantes.

Versão A: TA com Administrador Centralizado em processador dedicado

Um primeiro exemplo de TA assíncrono com Administrador centralizado em processador dedicado, foi estudado na seção anterior, para o caso de 'overlapping' total (ver Figura 3.4). Para exemplificar a utilização de 'overlapping' parcial, consideramos o *Team Algorithm Generalizado* com Administrador centralizado em processador dedicado, que tem associado a matriz de atribuição dada por (3.31). Este TA é esquematizado na Figura 3.6 e pode ser representado pelas equações abaixo:

$$\begin{aligned}
\begin{bmatrix} x_1(k+1) \\ \chi_{31}(k+1) \end{bmatrix} &= \begin{bmatrix} G_{11}(x^1(k)) \\ G_{31}(x^1(k)) \end{bmatrix} \\
\begin{bmatrix} x_2(k+1) \\ \chi_{32}(k+1) \end{bmatrix} &= \begin{bmatrix} G_{22}(x^2(k)) \\ G_{32}(x^2(k)) \end{bmatrix} \\
[\, x_3(k+1)\,] &= [\, c\,x_3(k) + w_1\,\chi_{31}(d_1^3(k)) + w_2\,\chi_{32}(d_2^3(k))\,]
\end{aligned} \tag{3.34}$$

onde $c = 1 - w_1 - w_2$.

Devemos ressaltar que a introdução das variáveis χ_{31} e χ_{32}, faz com que um novo vetor de estado W, que chamamos *vetor de estado expandido*, tenha que ser definido para possibilitar a análise de convergência utilizando a mesma técnica introduzida no Capítulo 2 para os *bloco TA*.

$$W(k) = \begin{bmatrix} W_1(k) \\ W_2(k) \\ W_3(k) \\ W_4(k) \\ W_5(k) \end{bmatrix} \stackrel{\text{def}}{=} \begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \\ \chi_{31}(k) \\ \chi_{32}(k) \end{bmatrix} \quad \text{onde} \quad \begin{cases} W_1 = x_1 \in \mathbb{R}^{n_1} \\ W_2 = x_2 \in \mathbb{R}^{n_2} \\ W_3 = x_3 \in \mathbb{R}^{n_3} \\ W_4 = \chi_{31} \in \mathbb{R}^{n_3} \\ W_5 = \chi_{32} \in \mathbb{R}^{n_3} \end{cases} \tag{3.35}$$

Nota 3.4 Para os bloco TA estudados no Capítulo 2 temos que: $W_i = x_i$, $\forall i \in \{1,\ldots,m\}$, e portanto $W = x \in D \subset \mathbb{R}^n$, pois não existe duplicação de variáveis, dado que nenhum tipo de 'overlapping' é utilizado.

Considerando a versão A do TA Generalizado com Administrador centralizado, representado matematicamente por (3.34), e lembrando a relação (3.28), temos que:

$$x \in D = D_1 \times D_2 \times D_3;$$

sendo assim, o vetor de estado expandido W, pertence a um *Domínio Expandido* D_w dado por:

$$W \in D_w = D_1 \times D_2 \times D_3 \times D_3 \times D_3;$$

pois lembramos que χ_{31}, χ_{32} e x_3 são versões diferentes do mesmo subvetor.

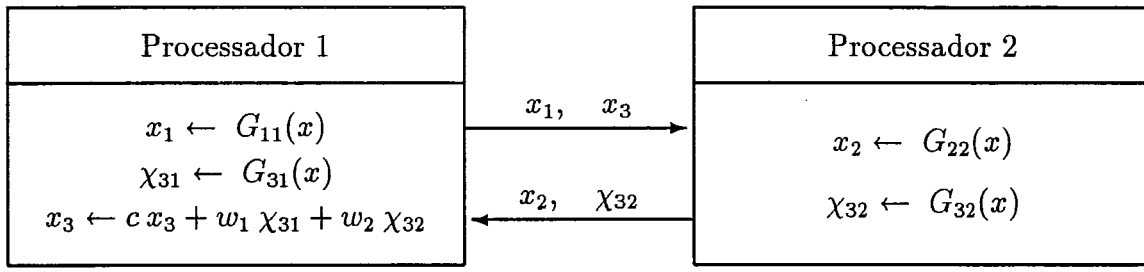

Figura 3.7: `Implementação assíncrona da versão B de um TA.`

O estudo das condições de convergência dos TA, no caso geral, será realizado na próxima seção. Mesmo assim, enfatizamos que o TA Generalizado dado por (3.34), pode ser considerado como um algoritmo bloco iterativo assíncrono que atualiza W; conseqüentemente, o Teorema 2.1 pode ser utilizado para obter uma condição suficiente de convergência, uma vez que as correspondentes hipóteses sejam atendidas no domínio expandido D_w.

Versão B: TA com Administrador Centralizado sem processador dedicado

A implementação assíncrona da versão A do TA Generalizado, utiliza um processador dedicado para implementar o Administrador. Porém, em muitos problemas de engenharia como os tratados no próximo capítulo, o número de variáveis *replicadas* é pequeno com respeito à dimensão do problema. Ainda mais, a complexidade (medida pelo número de operações por iteração) dos algoritmos sendo combinados, é bem maior que a do Administrador que se limita a fazer uma soma *ponderada* de vetores. Sendo assim, nem sempre se justifica dedicar um processador exclusivamente à implementação do Administrador. Várias alternativas são então possíveis, sendo provavelmente a mais simples aquela esquematizada na Figura 3.7, para o TA Generalizado que tem associado a matriz de atribuição dada por (3.31), na qual a função do Administrador é incluida no processador 1.

A versão B do TA Generalizado com Administrador Centralizado, esquematizado na Figura 3.7, pode ser visto como uma simples variante da versão A; tanto assim, que ambas versões tem o mesmo vetor de estado expandido W, dado por (3.35).

Nota 3.5 Outra alternativa para evitar a utilização de um processador dedicado às funções do Administrador, seria a distribuição das funções administrativas entre os diversos processadores do sistema. Assim, de cada conjunto de processadores que atualizam um mesmo subvetor, escolhe-se um processador para atualizar esse subvetor,

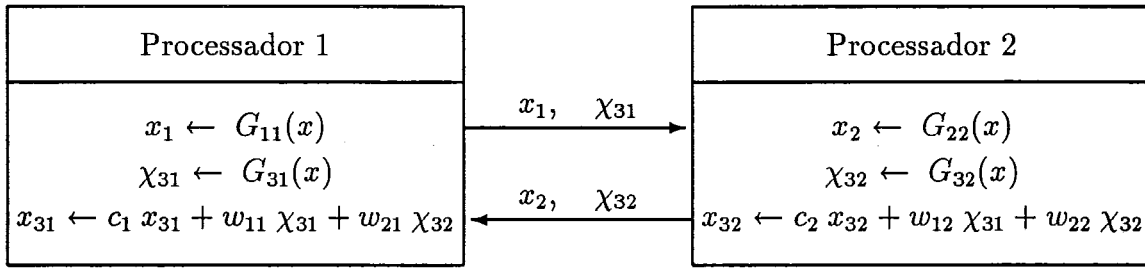

Figura 3.8: Implementação assíncrona da versão C de um TA.

utilizando a correspondente soma ponderada. Desta forma, nenhum processador é sobrecarregado com todo o trabalho do Administrador.

Versão C: TA com Administrador replicado em cada processador

Uma outra alternativa de implementar uma versão assíncrona do TA com *Administrador Distribuído* que tem associado a matriz de atribuição dada por (3.31), é ilustrada na Figura 3.8. Nesta proposta, cada processador implementa seu próprio Administrador, independentemente deste estar replicado em outros processadores. Neste caso, cada processador j tem a sua versão do subvetor x_3, que é denotada como x_{3j}. Note também que os pesos utilizados em cada replicação do Administrador não necessariamente devem ser os mesmos. Para considerar esta liberdade adicional dos pesos, denotamos como w_{ij} o peso associado ao subvetor $\chi_{i\kappa}$ (calculado no processador κ), quando utilizado no Administrador do processador j, como ilustra a Figura 3.8. O TA assíncrono assim constituído pode ser representado pelas seguintes equações:

$$\begin{bmatrix} x_1(k+1) \\ \chi_{31}(k+1) \\ x_{31}(k+1) \end{bmatrix} = \begin{bmatrix} G_{11}(\,x^1(k)\,) \\ G_{31}(\,x^1(k)\,) \\ c_1\, x_{31}(k) \;+\; w_{11}\, \chi_{31}(k) \;+\; w_{21}\, \chi_{32}(\,d_2^1(k)\,) \end{bmatrix}$$

$$\begin{bmatrix} x_2(k+1) \\ \chi_{32}(k+1) \\ x_{32}(k+1) \end{bmatrix} = \begin{bmatrix} G_{22}(\,x^2(k)\,) \\ G_{32}(\,x^2(k)\,) \\ c_2\, x_{32}(k) \;+\; w_{12}\, \chi_{31}(\,d_1^2(k)\,) \;+\; w_{22}\, \chi_{32}(k) \end{bmatrix} \tag{3.36}$$

Considerando a versão C do TA Generalizado com Administrador distribuído e replicado, representado por (3.36), podemos definir o correspondente vetor de estado expandido W como:

$$W(k) = \begin{bmatrix} W_1(k) \\ W_2(k) \\ W_3(k) \\ W_4(k) \\ W_5(k) \\ W_6(k) \end{bmatrix} \stackrel{\text{def}}{=} \begin{bmatrix} x_1(k) \\ x_2(k) \\ x_{31}(k) \\ x_{32}(k) \\ \chi_{31}(k) \\ \chi_{32}(k) \end{bmatrix} \quad \text{onde} \quad \begin{cases} W_1 = x_1 \in \mathbb{R}^{n_1} \\ W_2 = x_2 \in \mathbb{R}^{n_2} \\ W_3 = x_{31} \in \mathbb{R}^{n_3} \\ W_4 = x_{32} \in \mathbb{R}^{n_3} \\ W_5 = \chi_{31} \in \mathbb{R}^{n_3} \\ W_6 = \chi_{32} \in \mathbb{R}^{n_3} \end{cases} \tag{3.37}$$

conseqüentemente,

$$W \in D_w = D_1 \times D_2 \times D_3 \times D_3 \times D_3 \times D_3;$$

O 'overlapping' parcial discutido na introdução do presente capítulo, e tratado nos trabalhos de Ikeda e Šiljak (1980, 1981, 1984 e 1987), pode ser considerado como um caso particular da versão C do TA com Administrador distribuído e replicado. Efetivamente, se fazemos a seguinte escolha dos pesos

$$w_{11} = w_{22} = 1; \qquad w_{21} = w_{12} = 0$$

vemos que a função do Administrador é simplesmente enfatizar a igualdade $x_{3j} = \chi_{3j}$, $j = 1, 2$. Em outras palavras, cada processador j $(j = 1, 2)$ se limita a resolver seu subproblema local

$$\begin{bmatrix} \Phi_j(x) \\ \Phi_3(x) \end{bmatrix} = 0, \qquad \text{onde} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_{3j} \end{bmatrix}$$

o que é equivalente a expandir o problema original $\Phi(x) = 0$ de forma que as equações correspondentes ao subproblema $\Phi_3(x) = 0$ sejam repetidas em cada processador.

3.2.2 Um Exemplo Ilustrativo de 'Overlapping' Parcial

Para aproveitar as vantagens que possibilita um TA com 'overlapping' parcial, é interessante estudar condições sob as quais esta técnica apresenta vantagens. Um primeiro estudo para o caso linear é encontrado em Calvet e Titli (1989); porém, casos mais complexos devem ainda ser estudados.

Para enfatizar algumas condições nas quais um TA Generalizado apresenta vantagens evidentes sobre os bloco TA estudados no Capítulo 2, se apresenta a seguir um sistema quase-linear simples ($n = m = 3$), que facilita a análise das características de convergência utilizando a metodologia introduzida no Capítulo 2.

Exemplo 3.3 Resolver, utilizando o método de Jacobi, o sistema de equações quase-lineares $Ax = F(x)$ dado por:

$$\begin{bmatrix} a & b & 0 \\ e & c & e \\ 0 & b & a \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} d \\ f(x_2) \\ d \end{bmatrix} \tag{3.38}$$

onde a matriz $A = (a_{ij})$, com $a_{ij} \geq 0$, é não singular ($D = ac - be \neq 0$), a função $f(x_2)$ é Lipschitz continua com constante L, e a razão b/a é 'grande'.

O sistema de equações dado por (3.38) pode ser particionado de diversas formas, dando lugar a diferentes alternativas de solução, num contexto assíncrono. Se analisam a seguir 5 casos típicos:

Caso 1: método de Jacobi, com $n = m = p = 3$.

O algoritmo assíncrono do método 'point' Jacobi, pode ser descrito pelas equações:

$$\begin{aligned} [x_1(k+1)] &= \left[\frac{d}{a} - \frac{b}{a} x_2(d_2^1(k)) \right] \\ [x_2(k+1)] &= \left[\frac{1}{c}\left(-e\, x_1(d_1^2(k)) + f(x_2(k)) - e\, x_3(d_3^2(k))\right) \right] \\ [x_3(k+1)] &= \left[\frac{d}{a} - \frac{b}{a} x_2(d_2^3(k)) \right] \end{aligned} \tag{3.39}$$

Para a análise de convergência do algoritmo assíncrono (3.39), podemos utilizar o Teorema 2.1; com este objetivo, derivamos a correspondente matriz de comparação H_1:

$$H_1 = \begin{bmatrix} 0 & (b/a) & 0 \\ (e/c) & (L/c) & (e/c) \\ 0 & (b/a) & 0 \end{bmatrix} \tag{3.40}$$

desta forma, para valores suficientemente grandes da razão (b/a) temos que:

$$\rho(H_1) > 1$$

conseqüentemente, o Teorema 2.1 não pode ser utilizado para assegurar a convergência do algoritmo. Além disto, observando (3.39) podemos notar que tanto x_1 quanto x_3 divergem quando (b/a) é muito grande. Portanto, o algoritmo assíncrono dado por (3.39), em geral, diverge.

Caso 2: método bloco Jacobi, com $p = m = 2$.

O algoritmo assíncrono que utiliza o método bloco Jacobi (BJ), sem 'overlapping', é descrito pelas equações:

$$\begin{aligned} [x_1(k+1)] &= \left[\frac{d}{a} - \frac{b}{a} x_2(d_2^1(k)) \right] \\ \begin{bmatrix} x_2(k+1) \\ x_3(k+1) \end{bmatrix} &= \frac{1}{D} \begin{bmatrix} a & -e \\ -b & c \end{bmatrix} \begin{bmatrix} f(x_2(k)) - e\, x_1(d_1^2(k)) \\ d \end{bmatrix} \end{aligned} \tag{3.41}$$

onde lembramos que $D = ac - be \neq 0$. Este algoritmo assíncrono tem uma matriz de comparação H_2 correspondente, dada por:

$$H_2 = \begin{bmatrix} 0 & (b/a) \\ (be/D) & (bL/D) \end{bmatrix} \tag{3.42}$$

que mais uma vez caracteriza-se por ter:

$$\rho(H_2) > 1$$

conseqüentemente, o Teorema 2.1 não pode ser utilizado para assegurar a convergência do algoritmo. Ainda mais, observando (3.41) podemos notar que x_1 diverge quando (b/a) é muito grande. Portanto, o algoritmo assíncrono dado por (3.41), em geral, diverge.

Vemos assim que não é possível resolver o problema (3.38) num contexto assíncrono, utilizando versões sem 'overlapping' do algoritmo de Jacobi. Porém, utilizando um 'overlapping' parcial referente à equação $\Phi_2(x) = 0$, o que no caso é equivalente a replicá-la, é possível construir diversas versões de TA Generalizados para as quais, no contexto da formulação do Capítulo 2, o raio espectral da matriz de comparação $\rho(H)$ não necessariamente tende a infinito quando a razão (b/a) é muito grande; e portanto a convergência pode ser assegurada utilizando o Teorema 2.1.

O novo sistema de equações a ser resolvido, denominado *sistema expandido* em Ikeda e Šiljak (1980), tem uma dimensão expandida $\bar{n} = 4$ e pode ser representado por:

$$\begin{bmatrix} a & b & 0 & 0 \\ e & c & 0 & e \\ e & 0 & c & e \\ 0 & 0 & b & a \end{bmatrix} \begin{bmatrix} x_1 \\ x_{21} \\ x_{22} \\ x_3 \end{bmatrix} = \begin{bmatrix} d \\ f(x_{21}) \\ f(x_{22}) \\ d \end{bmatrix} \tag{3.43}$$

Consideramos a seguir alguns TA Generalizados que resolvem (3.38) utilizando 'overlapping' parcial, o que equivale a resolver (3.43) combinando no Administrador as duas versões de x_2 (x_{21} e x_{22}). Além de utilizar 'overlapping' parcial, os TA que se analisam a seguir, nesta subseção, tem em común os parâmetros: $p = 2$ e $m = 3$, assim como uma escolha de pesos no Administrador que atende a relação $c \geq 0$ (isto para simplificar a análise de convergência dos TA Generalizados que seguem).

Caso 3: TA com Administrador centralizado em processador dedicado
(Versão A da Tabela 3.2)

A semelhança da versão A do TA Generalizado com Administrador centralizado em processador dedicado (ilustrado na Figura 3.6), o algoritmo assíncrono correspondente é dado pelas equações:

$$\begin{bmatrix} x_1(k+1) \\ \chi_{21}(k+1) \end{bmatrix} = \frac{1}{D} \begin{bmatrix} c & -b \\ -e & a \end{bmatrix} \begin{bmatrix} d \\ f(x_2(d_3^1(k))) - e\, x_3(d_2^1(k)) \end{bmatrix}$$

$$\begin{bmatrix} \chi_{22}(k+1) \\ x_3(k+1) \end{bmatrix} = \frac{1}{D} \begin{bmatrix} a & -e \\ -b & c \end{bmatrix} \begin{bmatrix} f(x_2(d_3^2(k))) - e\, x_1(d_1^2(k)) \\ d \end{bmatrix} \qquad (3.44)$$

$$[x_2(k+1)] = [c\, x_2(k) + w_1\, \chi_{21}(d_1^3(k)) + w_2\, \chi_{22}(d_2^3(k))]$$

que tem uma matriz de comparação H_3 correspondente, dada por:

$$H_3 = \begin{bmatrix} 0 & (be/D) & (bL/D) \\ (be/D) & 0 & (bL/D) \\ w_1 & w_2 & c \end{bmatrix} \qquad (3.45)$$

que não é função direta de (b/a) e portanto $\rho(H_3)$ pode ser menor que 1. Sendo assim, satisfeitas as condições do Teorema 2.1, a convergência do TA está assegurada.

Uma outra variante do Caso 3 com Administrador centralizado, semelhante à versão B do TA Generalizado com Administrador centralizado sem processador dedicado, pode ser obtida implementando o Administrador num dos processadores encarregados de implementar o método BJ, como por exemplo o processador 1. Neste caso, as versões diferentes do subvetor x_2 no processador 1 podem ser reduzidas a uma única variável x_2, como ilustra o seguinte exemplo:

Caso 4: TA com Administrador implícito no Processador 1
(Versão E da Tabela 3.2)

Para este exemplo, não temos uma equação explícita para o Administrador. Porém, ela está incluida implicitamente no processador 1. O algoritmo assíncrono correspondente pode ser descrito pelas equações:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \frac{1}{D} \begin{bmatrix} c & -b \\ -e & a \end{bmatrix} \begin{bmatrix} d \\ f(\overline{\chi}_1(k)) - e\, x_3(d_2^1(k)) \end{bmatrix}$$

com $\overline{\chi}_1(k) = (c+w_1)\, x_2(k) + w_2\, \chi_{22}(d_2^1(k))$, e

$$\begin{bmatrix} \chi_{22}(k+1) \\ x_3(k+1) \end{bmatrix} = \frac{1}{D} \begin{bmatrix} a & -e \\ -b & c \end{bmatrix} \begin{bmatrix} f\big(x_2(d_1^2(k))\big) - e\, x_1(d_1^2(k)) \\ d \end{bmatrix} \qquad (3.46)$$

onde a função do Administrador, de combinar mediante uma soma ponderada as diferentes versões de x_2, é realizada implicitamente no processador 1 como parte dos cálculos na avaliação de $f(\cdot)$. Dai surge o nome de *Administrador implícito*.

A matriz de comparação H_4 correspondente a este algoritmo é dada por:

$$H_4 = \frac{b}{D} \begin{bmatrix} L(c+w_1) & (Lw_2+e) \\ (L+e) & 0 \end{bmatrix} \qquad (3.47)$$

a qual, para uma escolha apropriada de pesos, tem um raio espectral

$$\rho(H_4) = b(L+e)/D$$

que não depende dos pesos e satisfaz a relação:

$$\rho(H_4) \leq \rho(H_3) \tag{3.48}$$

Consideramos a seguir um TA com Administrador distribuído, semelhante à versão C do TA com Administrador distribuído, representado na Figura 3.8. Mais uma vez, consideramos a função administrativa como implícita em cada um dos processadores.

Caso 5: TA com Administrador Replicado e Implícito.
(Versão D da Tabela 3.2)

A exemplo do Caso 4, não temos uma equação explícita para a função do Administrador. O correspondente método BJ assíncrono pode ser descrito pelas equações:

$$\begin{bmatrix} x_1(k+1) \\ x_{21}(k+1) \end{bmatrix} = \frac{1}{D} \begin{bmatrix} c & -b \\ -e & a \end{bmatrix} \begin{bmatrix} d \\ f(\overline{\chi}_1(k)) - e\, x_3(d_2^1(k)) \end{bmatrix}$$

$$\begin{bmatrix} x_{22}(k+1) \\ x_3(k+1) \end{bmatrix} = \frac{1}{D} \begin{bmatrix} a & -e \\ -b & c \end{bmatrix} \begin{bmatrix} f(\overline{\chi}_2(k)) - e\, x_1(d_1^2(k)) \\ d \end{bmatrix} \tag{3.49}$$

$$\text{com } \overline{\chi}_1(k) = (c_1 + w_{11})\, x_{21}(k) + w_{21}\, x_{22}(d_2^1(k))$$
$$\text{e } \overline{\chi}_2(k) = (c_2 + w_{22})\, x_{22}(k) + w_{12}\, x_{21}(d_1^2(k)).$$

A matriz de comparação H_5, correspondente a este algoritmo, é dada por:

$$H_5 = \frac{b}{D} \begin{bmatrix} (c_1 + w_{11})L & (Lw_{21} + e) \\ (Lw_{12} + e) & (c_2 + w_{22})L \end{bmatrix} \tag{3.50}$$

a qual, para uma escolha apropriada de pesos, tem o raio espectral $\rho(H_5) = b(L+e)/D$, que não depende dos pesos e satisfaz a igualdade:

$$\rho(H_5) = \rho(H_4) = \frac{b}{D}(L+e) \tag{3.51}$$

Enfatizamos novamente que o 'overlapping' parcial introducido em Ikeda e Šiljak (1980) é um caso particular do TA apresentado no Caso 5, para a seguinte escolha de pesos:

$$w_{11} = w_{22} = 1, \quad w_{12} = w_{21} = 0 \qquad \text{que implica em:} \quad c_1 = c_2 = 0$$

Efetivamente, com esta escolha de pesos obtemos o algoritmo bloco Jacobi da equação (3.43), que é representado pelas equações:

$$\begin{bmatrix} x_1(k+1) \\ x_{21}(k+1) \end{bmatrix} = \frac{1}{D} \begin{bmatrix} c & -b \\ -e & a \end{bmatrix} \begin{bmatrix} d \\ f\big(x_{21}(k)\big) - e\, x_3(\, d_2^1(k)\,) \end{bmatrix}$$

$$\begin{bmatrix} x_{22}(k+1) \\ x_3(k+1) \end{bmatrix} = \frac{1}{D} \begin{bmatrix} a & -e \\ -b & c \end{bmatrix} \begin{bmatrix} f\big(x_{22}(k)\big) - e\, x_1(\, d_1^2(k)\,) \\ d \end{bmatrix} \tag{3.52}$$

que, claramente, tem o raio espectral dado por (3.51); dado que $\rho(H_5)$ não depende dos pesos.

Enfatizamos aqui, que todos os TA Generalizados com 'overlapping' que foram apresentados podem ter suas condições de convergência asseguradas pelo Teorema 2.1, mesmo quando a razão (b/a) é muito grande, diferentemente dos métodos sem 'overlapping', que não conseguem convergir. Portanto, fica claramente estabelecido que a utilização de 'overlapping' parcial pode acarretar certas vantagens.

Resta determinar qual das 3 versões do TA Generalizado com 'overlapping' é a mais recomendável. Para responder a esta pergunta, consideramos as equações (3.48) e (3.51) e assim podemos obter a relação:

$$\rho(H_5) \;=\; \rho(H_4) \;\leq \rho(H_3) \tag{3.53}$$

portanto podemos concluir que o uso de um Administrador implícito nos Casos 4 e 5 é o mais recomendável, um fenómeno que se repetirá em contextos mais gerais abaixo.

Isto pode ser entendido porque o uso de um processador dedicado para implementar o Administrador, pode introduzir maiores atrasos num sistema assíncrono. De fato, a matriz de comparação H_3 do TA com Administrador centralizado em processador dedicado, tem os elementos da linha 3 satisfazendo a relação:

$$\sum_{j=1}^{3} h_{3j} \;=\; w_1 + w_2 + c \;=\; 1$$

enquanto os elementos das outras linhas de H_3 satisfazem a relação:

$$\sum_{j=1}^{3} h_{ij} \;=\; \frac{b}{D}(e+L) \;=\; \rho(H_4) \;=\; \rho(H_5), \qquad \forall i \in \{1,2\}$$

conseqüentemente, o motivo pelo qual o caso 3 apresenta um maior (ou igual) raio espectral que os casos com Administrador implícito, é a presença do Administrador em processador dedicado.

De especial interesse são os problemas nos quais um dado bloco i, que denominamos *bloco crítico* ($i = 2$ no exemplo 3.3), está fortemente ligado a outros (dois) blocos; portanto, qualquer partição sem 'overlapping' que separa algum dos três blocos considerados, corta uma ligação forte, o que fica refletido na correspondente matriz

de comparação, por um elevado valor das constantes bloco-Lipschitz que relacionam esses blocos. Sendo assim, explica-se a dificuldade de convergência dos algoritmos sem 'overlapping'.

Porém, replicando a equação correspondente ao *bloco crítico*, não é necessario cortar nenhuma ligação forte, e o correspondente TA Generalizado consegue, em geral, resolver o problema. Assim, no Exemplo 3.3, x_2 está fortemente ligado a x_1 e x_3 por ter $b \gg a$. Resolver x_2 em forma separada de x_1 ou x_3 não permite assegurar a convergência do algoritmo. A única solução encontrada foi replicar a equação correspondente ao bloco crítico em x_2, de forma a resolve-lo em forma conjunta com x_1 e x_3, segundo seja a copia do bloco crítico considerada.

Outra situação na qual o uso de um Administrador pode apresentar apreciáveis vantagens é quando se mostra difícil assegurar que $G(D) \subset D$. Nestes casos, o Administrador pode verificar se um novo valor de x está na região de interesse D e calcular 'inteligentemente' os pesos de forma a assegurar que $x(k+1) \in D$. Um exemplo neste sentido será apresentado no próximo capítulo, quando se estuda o problema do Fluxo de Potência Elétrica, no qual a solução de interesse deve estar na faixa de $1 \pm \epsilon$ p.u. O Administrador pode assim trabalhar com pesos constantes enquanto a condição é satisfeita, mas modificar os pesos no caso do algoritmo gerar valores fora do domínio de interesse. Com isto, o Administrador utilizado nos problemas exemplo do capítulo 5, consegue evitar a divergência prematura de um algoritmo assíncrono.

Concluimos esta seção resumindo as situações nas quais podemos esperar vantagens na utilização de um TA Generalizado, apesar do custo computacional adicional que pode significar o fato de replicar algumas equações e implementar um Administrador.

Quando usar um Team Algorithm Generalizado

1. Quando queremos utilizar um algoritmo rápido que nem sempre converge isoladamente e portanto necessitamos combiná-lo com outro de melhor convergência, mesmo que mais lento, para assegurar a convergência. É o caso do Exemplo 3.1.

2. Quando não é possível assegurar que $G(D) \subset D$, o Administrador pode ser utilizado para evitar uma divergência prematura de um dos algoritmos sendo combinados. Os problemas de fluxo de potência elétrica, do próximo capítulo, exemplificam este caso, segum comentário acima.

3. Quando existem *blocos críticos*, fortemente acoplados a outros blocos, fato que não permite obter partições simples que resolvam o problema considerado. Replicar os blocos críticos pode melhorar a convergência. É o caso do exemplo 3.3.

3.3 Formulação Matemática e Análise de Convergência

Nesta seção formalizamos matematicamente o conceito de *Team Algorithm Generalizado* e derivamos condições suficientes de convergência para algumas das versões apresentadas nas seções anteriores. O estudo de diferentes versões de TA Generalizados com Administrador, nesta seção, tem por objetivo assentar as bases, de forma a fornecer ferramentas suficientes a análise das diversas variantes possíveis.

Sem perder a generalidade em nossa análise, consideraremos exclusivamente o Administrador que utiliza pesos constantes, para simplificar a notação. Porém, lembramos que os resultados podem ser facilmente estendidos à análise de Administradores utilizando pesos variáveis, como foi realizado na Seção 3.1, quando introduzimos a versão síncrona dos TA com Administrador. Neste sentido, lembra-se ao leitor que o teorema de convergência que serve de base a nossas demonstrações, apresentado em Kaszkurewicz et al. (1990), é válido também para os problemas variantes no tempo, como seria o caso do Administrador que utiliza pesos variáveis.

A análise de convergência que se pretende fazer é similar àquela realizada no Capítulo 2, quando foram apresentados os bloco TA. Enfatizamos que a partição do problema e a opção pelo tipo de algoritmo para cada partição, são considerados *dados*, representados na matriz de atribuição Ψ. Para facilitar a análise que segue, relembramos as três hipóteses básicas que foram utilizadas na análise do Capítulo 2, utilizando agora a notação específica deste capítulo. Começamos com a Hipótese 2.1 que estabelece a unicidade da solução do algoritmo utilizado para resolver um dado problema. Note a inclusão da nova notação para os operadores, introduzida por (3.33), que repetimos a seguir:

$$G_j(x) = \begin{bmatrix} G_{1j}(x) \\ \vdots \\ G_{mj}(x) \end{bmatrix}, \qquad G_{ij}(x) : D \to D_i, \quad \forall i \in \{1,\ldots,m\}, \quad \forall j \in \{1,\ldots,p\}$$

HIPÓTESE 3.1 (UNICIDADE DA SOLUÇÃO)
Dado o sistema de equações (2.1), que pode ser re-escrito como (2.6), definido num conjunto fechado $D \subset \mathbb{R}^n$ que satisfaz (2.3), existem p operadores G_j tais que

$$G_{ij}(D) \subset D_i \qquad \forall i \in \{1,\ldots,m\}, \quad \forall j \in \{1,\ldots,p\}$$

ou seja, $G_j(D) \subset D \quad \forall j \in \{1,\ldots,p\}$, e adicionalmente D apresenta um e somente um ponto fixo

$$x^* = \begin{bmatrix} x_1^* \\ \vdots \\ x_m^* \end{bmatrix}, \qquad x_i^* \in D_i \subset \mathbb{R}^{n_i}, \qquad \forall i \in \{1,\ldots,m\}$$

dos operadores G_j. O ponto fixo x^* é por sua vez solução de (2.1). Conseqüentemente, podemos escrever que:

$$x_i^* = G_{ij}(x^*) \qquad \forall i \in \{1,\ldots,m\}, \quad \forall j \in \{1,\ldots,p\} \tag{3.54}$$

e também

$$\Phi_i(x^*) = 0, \qquad \forall i \in \{1,\ldots,m\} \tag{3.55}$$

A Hipótese 2.2 que estabelece a condição de assíncronismo parcial foi a segunda hipótese utilizada, e pode ser aqui repetida sem modificações:

HIPÓTESE 3.2 (ASSINCRONISMO PARCIAL)

$$\begin{array}{ll} \exists\, d \in \mathbb{N}, & \forall k \in \mathbb{N}, \quad \forall i,j \in \{1,\ldots,m\}, \\ \textit{tal que} & d_j^i(k) \in \{k, k-1, \ldots, k-d\}. \end{array} \tag{3.56}$$

Por último, foi considerado no Capítulo 2, a Hipótese 2.3 estabelecendo que os algoritmos utilizados são bloco-Lipschitz continuos no domínio de interesse. A modificação da notação dos operadores, faz com que seja necessário incluir um terceiro índice nas constantes bloco-Lipschitz, desta forma temos:

HIPÓTESE 3.3 (BLOCO-LIPSCHITZ)
Cada operador $G_{ij}(x)$ dado por (3.33) é bloco-Lipschitz continuo em D, i.e.

$$\forall x,y \in D, \quad \|G_{ij}(x) - G_{ij}(y)\| \leq \sum_{k=1}^{m} l_{ijk} \,\|\, x_k - y_k \,\|, \qquad \begin{array}{l} \forall i \in \{1,\ldots,m\}, \\ \forall j \in \{1,\ldots,p\} \end{array} \tag{3.57}$$

Finalmente, introduzimos para a análise que segue, uma última hipótese que permite assegurar que o subvetor x_i calculado pelo Administrador, pertence ao domínio de interesse (D_i). Isto é necessario para assegurar que o Administrador não faça uma escolha dos pesos que leve ao TA Generalizado a divergir.

HIPÓTESE 3.4 (ESCOLHA DE PESOS NO ADMINISTRADOR)
O domínio D considerado nas hipóteses 3.1 e 3.3 é convexo e a escolha de pesos realizada no Administrador é tal que todo subvetor x_i por ele calculado pertence a D_i.

Nota 3.6 Denotando como $G_{ta}(W)$ ao operador que representa ao TA Generalizado operando sobre o correspondente vetor de estado expandido W, onde $W \in D_w$, o correspondente TA Generalizado pode ser representado por:

$$W \leftarrow G_{ta}(W)$$

então, as hipóteses 3.1 e 3.4 asseguram que

$$G_{ta}(W) \subset D_w$$

como será facilmente notado na formalização dos TA Generalizados.

Para simplificar o estudo dos TA num contexto assíncrono, assumimos uma matriz de atribuição Ψ ordenada, de forma que

$$\forall i \in \{1,\ldots,r\}, \qquad \psi_{ij} = \begin{cases} 1 & \text{se } i = j \\ 0 & \text{se } i \neq j \end{cases}$$

o que implica em uma ordenação das variáveis de forma que todo subvetor x_i com $i \leq r$ é atualizado pelo processador i (e somente por ele). Sendo assim, denotamos como $\widehat{G_i}$ ao operador G_{ii} com $i \leq r$, que atualiza x_i. Conseqüentemente, denotaremos como $\widehat{l_{ik}}$, $\forall k \in \{1,\ldots,m\}$, as constantes bloco-Lipschitz de $\widehat{G_i}$.

No que segue, assumimos $r \geq 1$, de forma a escrever as equações dos *Team Algorithms Generalizados* para o caso geral.

Versão A: TA com Administrador Centralizado em processador dedicado

Chamamos *versão A de um TA* ao Team Algorithm Generalizado com Administrador centralizado em processador dedicado, como foi introduzido na subseção 3.2.1 para o Exemplo 3.2, onde foi apresentada a versão A do TA que tem associado a matriz de atribuição (3.31). O referido TA do Exemplo 3.2, esquematizado na Figura 3.6, tem um vetor de estado expandido W dado por (3.35), que pode ser facilmente generalizado. Sendo assim, o vetor de estado expandido W_a da versão A de um TA, para o caso geral, é dado por:

$$W_a = \begin{bmatrix} W_1 \\ \vdots \\ W_m \end{bmatrix} \tag{3.58}$$

$$\text{onde} \quad \forall i \in \{1,\ldots,m\}, \qquad W_i = \begin{cases} [x_i] & \in \mathbb{R}^{n_i} \quad \text{para} \quad 1 \leq i \leq r \\ \begin{bmatrix} x_i \\ \psi_{i1}\chi_{i1} \\ \vdots \\ \psi_{ip}\chi_{ip} \end{bmatrix} & \in \mathbb{R}^{(p+1)n_i} \quad \text{para} \quad r < i \leq m \end{cases}$$

onde explicitamente multiplicamos χ_{ij} por $\psi_{ij} \in \{0,1\}$ para enfatizar que algumas componentes de W_a são nulas, e portanto poderiam ser eliminadas do vetor de estado expandido, quando o estudo se refere a um problema particular.

Para exemplificar a notação introduzida em (3.58), re-escrevemos a seguir o vetor de estado expandido da versão A do TA Generalizado do Exemplo 3.2:

$$W_a = \begin{bmatrix} W_1 \\ W_2 \\ W_3 \end{bmatrix}, \qquad \text{onde} \quad \begin{cases} W_1 = x_1 \\ W_2 = x_2 \\ W_3 = \begin{bmatrix} x_3 \\ \chi_{31} \\ \chi_{32} \end{bmatrix} \end{cases} \tag{3.59}$$

Conhecido o vetor de estado expandido, podemos representar ao TA com Administrador centralizado que atualiza W_a, num contexto assíncrono, pelas equações:

$$\begin{aligned} [\, x_i(k+1)\,] &= [\, \widehat{G_i}(x^i(k))\,] && \text{para } 1 \le i \le r \\ \begin{bmatrix} x_i(k+1) \\ \psi_{i1}\chi_{i1}(k+1) \\ \vdots \\ \psi_{ip}\chi_{ip}(k+1) \end{bmatrix} &= \begin{bmatrix} cx_i(k) + \sum_{j=1}^{p} w_j \psi_{ij}\chi_{ij}(d_j^i(k)) \\ G_{i1}(\, x^1(k)\,) \\ \vdots \\ G_{ip}(\, x^p(k)\,) \end{bmatrix} && \text{para } r < i \le m \end{aligned} \tag{3.60}$$

onde lembramos que $c + \sum_{j=1}^{p} w_j = 1$ por (3.14).

Nota 3.7 A relação (3.14) pode ser facilmente derivada no contexto dos TA Generalizados assíncronos, impondo a condição de que se todas as versões de um subvetor x_i são dadas pela mesma solução x_i^*, a atualização do Administrador deve ser a mesma solução x_i^*, i.e.

$$\text{o Administrador calcula:} \qquad x_i(k+1) = c\,x_i^* + \sum_{j=1}^{p} w_j\, x_i^*$$

$$\text{onde} \quad x(k+1) = x_i^*$$

conseqüentemente,

$$1 = c + \sum_{j=1}^{p} w_j \tag{3.61}$$

Considerando a versão A do TA Generalizado do Exemplo 3.2, resulta fácil verificar que a representação geral do algoritmo, dado por (3.60), se reduz à equação (3.34), apresentada em oportunidade de representar o referido TA do Exemplo 3.2.

Uma condição suficiente de convergência para o algoritmo assíncrono representado por (3.60) pode ser facilmente derivada utilizando o Teorema 2.1. Para isto, observando (3.60), resulta simples verificar que $G_{ta}(W_a) \subset D_w$, conforme Nota 3.6.

Com o objetivo de utilizar o Teorema 2.1, calculamos a matriz de comparação H_a da versão A do TA assíncrono representado por (3.60), utilizando as constantes bloco-Lipschitz da Hipótese 3.3:

$$H_a = \left(H_{ij}\right), \qquad i,j = 1,\ldots,m \tag{3.62}$$

onde as submatrizes H_{ij} que representam a dependência de W_i com W_j são dadas por:

$$H_{ij} = \begin{cases} \widehat{l_{ij}} & \in \mathbb{R} & \text{para } 1 \le i \le r \text{ e } 1 \le j \le r \\ \left[\widehat{l_{ij}}\, 0 \cdots 0\right] & \in \mathbb{R}^{1\times(p+1)} & \text{para } 1 \le i \le r \text{ e } r < j \le m \\ \begin{bmatrix} 0 \\ l_{i1j} \\ \vdots \\ l_{ipj} \end{bmatrix} & \in \mathbb{R}^{p+1} & \text{para } r < i \le m \text{ e } 1 \le j \le r \\ \begin{bmatrix} |c| & \psi_{i1}w_1 & \cdots & \psi_{ip}w_p \\ l_{i1i} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{ipi} & 0 & \cdots & 0 \end{bmatrix} & \in \mathbb{R}^{(p+1)\times(p+1)} & \text{para } r < i \le m \text{ e } j = i \\ \begin{bmatrix} 0 & 0 & \cdots & 0 \\ l_{i1j} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{ipj} & 0 & \cdots & 0 \end{bmatrix} & \in \mathbb{R}^{(p+1)\times(p+1)} & \text{para } r < i,j \le m \text{ e } j \ne i \end{cases}$$

Para exemplificar este resultado, damos a seguir a matriz de comparação H_a correspondente à versão A do TA Generalizado do Exemplo 3.2, representado por (3.34):

$$H_a = \begin{bmatrix} \widehat{l_{11}} & \widehat{l_{12}} & \widehat{l_{13}} & 0 & 0 \\ \widehat{l_{21}} & \widehat{l_{22}} & \widehat{l_{23}} & 0 & 0 \\ 0 & 0 & |c| & w_1 & w_2 \\ l_{311} & l_{312} & l_{313} & 0 & 0 \\ l_{321} & l_{322} & l_{323} & 0 & 0 \end{bmatrix} \tag{3.63}$$

Calculada a matriz de comparação H_a, podemos escrever o seguinte corolário derivado do Teorema 2.1,

COROLÁRIO 3.4 *A versão assíncrona do Team Algorithm Generalizado, que utiliza um Administrador centralizado num processador dedicado (versão A), dado por (3.60), converge para a única solução x^* em D, sob as hipóteses 3.1 a 3.4, se $\rho(H_a) < 1$, sendo a matriz de comparação H_a correspondente, dada por (3.62).*

Considerando a simplicidade da versão B do TA com Administrador centralizado, apresentado na seção 3.1, assim como suas semelhanças com a versão A, passamos a analisar diretamente a versão C, que utiliza um Administrador distribuído e replicado em cada processador.

Versão C: TA com Administrador distribuído e replicado

Consideramos a seguir a formulação geral da versão C do TA: um Team Algorithm Generalizado com Administrador distribuído e replicado, do tipo ilustrado na Figura 3.8 para o Exemplo 3.2. Assim, utilizando o método com o qual derivamos uma condição suficiente de convergência para a versão A, podemos facilmente obter um resultado equivalente ao Corolário 3.4.

Na ocasião de apresentar a versão C do TA considerado no Exemplo 3.2, o vetor de estado expandido W correspondente, foi representado por (3.37). Generalizando este conceito, o vetor de estado expandido W_c da versão C do TA com Administrador distribuído e replicado, pode ser representado por:

$$W_c = \begin{bmatrix} W_1 \\ \vdots \\ W_m \end{bmatrix} \tag{3.64}$$

$$\text{onde} \quad \forall i \in \{1,\ldots,m\}, \quad W_i = \begin{cases} [x_i] & \in \mathbb{R}^{n_i} \qquad \text{para } 1 \leq i \leq r \\ \begin{bmatrix} x_{i1} \\ \vdots \\ x_{ip} \\ \psi_{i1}\chi_{i1} \\ \vdots \\ \psi_{ip}\chi_{ip} \end{bmatrix} & \in \mathbb{R}^{2pn_i} \qquad \text{para } r < i \leq m \end{cases}$$

lembrando que x_{ij} é a versão do subvetor x_i calculada no processador j.

Para exemplificar a notação introduzida em (3.64), re-escrevemos a seguir o vetor de estado expandido da versão C do TA Generalizado do Exemplo 3.2:

$$W_c = \begin{bmatrix} W_1 \\ W_2 \\ W_3 \end{bmatrix}, \qquad \text{onde} \quad \begin{cases} W_1 = x_1 \\ W_2 = x_2 \\ W_3 = \begin{bmatrix} x_{31} \\ x_{32} \\ \chi_{31} \\ \chi_{32} \end{bmatrix} \end{cases} \tag{3.65}$$

O TA com Administrador distribuído e replicado que atualiza o vetor de estado expandido W_c dado por (3.64), pode ser representado pelas equações:

$$
\begin{aligned}
[\, x_i(k+1)\,] \quad &= \quad [\, \widehat{G}_i(\, x^i(k)\,)\,] \qquad\qquad \text{para } 1 \leq i \leq r \\
\begin{bmatrix} x_{i1}(k+1) \\ \vdots \\ x_{ip}(k+1) \\ \psi_{i1}\chi_{i1}(k+1) \\ \vdots \\ \psi_{ip}\chi_{ip}(k+1) \end{bmatrix} \quad &= \quad \begin{bmatrix} c_1 x_{i1}(k) + \Sigma_{j=1}^{p} w_{j1}\,\psi_{ij}\,\chi_{ij}(\, d_j^1(k)\,) \\ \vdots \\ c_p x_{ip}(k) + \Sigma_{j=1}^{p} w_{jp}\,\psi_{ij}\,\chi_{ij}(\, d_j^p(k)\,) \\ G_{i1}(\, x^1(k)\,) \\ \vdots \\ G_{ip}(\, x^p(k)\,) \end{bmatrix} \qquad \text{para } r < i \leq m
\end{aligned}
\tag{3.66}
$$

onde denotamos como $x^j(k)$ ao valor de x disponível no processador j, na iteração k, ao aplicar os operadores G_{ij}, i.e.

$$
x^j(k) = \begin{bmatrix} x_1(\, d_1^j(k)\,) \\ \vdots \\ x_r(\, d_r^j(k)\,) \\ x_{r+1,\,j}(k) \\ \vdots \\ x_{mj}(k) \end{bmatrix}, \qquad \forall j \in \{1,\ldots,p\}
$$

e adicionalmente, verifica-se que $\forall j \in \{1,\ldots,p\}$ temos a relação $c_j + \sum_{i=1}^{p} w_{ij} = 1$ (justificativa similar à Nota 3.7).

Considerando a versão C do TA Generalizado do Exemplo 3.2, resulta fácil verificar que a representação geral do algoritmo, dado por (3.66), se reduz à equação (3.36), utilizada para representar o referido TA do Exemplo 3.2.

Com o objetivo de derivar uma condição suficiente de convergência para o algoritmo assíncrono representado por (3.66), enfatizamos que mais uma vez é possível verificar que $G_{ta}(W_a) \subset D_w$, a partir de (3.66).

Assim, a matriz de comparação H_c, correspondente à versão C do TA Generalizado representado por (3.66), é dada por:

$$
H_c = \big(H_{ij}\big), \qquad i,j = 1,\ldots,m \tag{3.67}
$$

onde as submatrizes H_{ij} que representam a dependência de W_i com W_j são dadas por:

$$H_{ij} = \begin{cases} \widehat{l_{ij}} & \in \mathbb{R} \quad \text{para} \quad 1 \leq i \leq r \quad \text{e} \quad 1 \leq j \leq r \\ \widehat{l_{ij}}\left[\psi_{i1} \cdots \psi_{ip}\, 0 \cdots 0\right] & \in \mathbb{R}^{1\times 2p} \quad \text{para} \quad 1 \leq i \leq r \quad \text{e} \quad r < j \leq m \\ \begin{bmatrix} 0 \\ \vdots \\ 0 \\ l_{i1j} \\ \vdots \\ l_{ipj} \end{bmatrix} & \in \mathbb{R}^{2p} \quad \text{para} \quad r < i \leq m \quad \text{e} \quad 1 \leq j \leq r \\ \begin{bmatrix} H_{ij}^{11} & H_{ij}^{12} \\ H_{ij}^{21} & H_{ij}^{22} \end{bmatrix} & \in \mathbb{R}^{2p\times 2p} \quad \text{para} \quad r < i \leq m \quad \text{e} \quad r < j \leq m \end{cases}$$

onde:

$$H_{ij}^{22} = \begin{pmatrix} 0 \end{pmatrix} \qquad \in \mathbb{R}^{p\times p}$$

$$H_{ij}^{21} = \operatorname{diag}\{l_{i1j} \cdots i_{ipj}\} \qquad \in \mathbb{R}^{p\times p}$$

$$H_{ii}^{11} = \operatorname{diag}\{|\, c_1 \,| \cdots |\, c_p \,|\} \qquad \in \mathbb{R}^{p\times p}$$

$$H_{ii}^{12} = \begin{bmatrix} (\psi_{i1} w_{11}) & \cdots & (\psi_{ip} w_{p1}) \\ \vdots & \ddots & \vdots \\ (\psi_{i1} w_{1p}) & \cdots & (\psi_{ip} w_{pp}) \end{bmatrix} \in \mathbb{R}^{p\times p}$$

e finalmente, para $i \neq j$ $\qquad H_{ij}^{11} = H_{ij}^{12} = \begin{pmatrix} 0 \end{pmatrix} \in \mathbb{R}^{p\times p}$

Para exemplificar este resultado, damos a seguir a matriz de comparação H_c correspondente à versão C do TA Generalizado do Exemplo 3.2, representado por (3.36):

$$H_c = \begin{bmatrix} \widehat{l_{11}} & \widehat{l_{12}} & \widehat{l_{13}} & 0 & 0 & 0 \\ \widehat{l_{21}} & \widehat{l_{22}} & 0 & \widehat{l_{23}} & 0 & 0 \\ 0 & 0 & |\, c_1 \,| & 0 & w_{11} & w_{21} \\ 0 & 0 & 0 & |\, c_2 \,| & w_{12} & w_{22} \\ l_{311} & l_{312} & l_{313} & 0 & 0 & 0 \\ l_{321} & l_{322} & 0 & l_{323} & 0 & 0 \end{bmatrix} \tag{3.68}$$

Tendo derivado a matriz de comparação, podemos aplicar diretamente o Teorema 2.1 e obter o seguinte corolário:

COROLÁRIO 3.5 *A versão assíncrona do Team Algorithm Generalizado, com Administrador distribuído e replicado em cada processador (versão C), dado por (3.66), converge para a única solução x^* em D, sob as hipóteses 3.1 a 3.4, se $\rho(H_c) < 1$, sendo a matriz de comparação H_c correspondente, dada por (3.67).*

Versão D: TA com Administrador distribuído implicitamente

No estudo de convergência da versão C do TA com Administrador distribuído, que até aqui fizemos, mantivemos explicitamente as equações adicionais correspondentes aos diversos Administradores, o que levou à replicação, em cada processador, das variáveis de estado afetadas por um 'overlapping' parcial. Assim, para cada subvetor x_i afetado por um 'overlapping', temos duas versões no mesmo processador: uma variável x_{ij} calculada no Administrador, e outra variável χ_{ij} calculada pelo algoritmo correspondente, sendo que na realidade, estamos interessados somente em uma delas.

O resultado desta duplicação aparentemente inútil, é a inclusão, na matriz de comparação H_c, de linhas com $\sum_j h_{ij} \geq 1$, correspondentes aos Administradores, que podem prejudicar na obtenção da condição $\rho(H) < 1$, bem como na minimização de $\rho(H)$.

A solução para este incoveniente é a representação implícita das equações correspondendentes ao Administrador, a exemplo do que foi apresentado nos casos 4 e 5 do Exemplo 3.3. Assim, cada processador j tem uma única versão do subvetor x_i , que denotamos x_{ij}, que resulta da atualização realizada pelo operador G_{ij} correspondente.

O vetor de estado expandido W_d da versão D do TA com Administrador distribuído implicitamente, pode assim ser representado por:

$$W_d = \begin{bmatrix} W_1 \\ \vdots \\ W_m \end{bmatrix} \tag{3.69}$$

$$\text{onde} \quad \forall i \in \{1,\ldots,m\}, \qquad W_i = \begin{cases} [x_i] & \in \mathbb{R}^{n_i} \qquad \text{para } 1 \leq i \leq r \\ \begin{bmatrix} x_{i1} \\ \vdots \\ x_{ip} \end{bmatrix} & \in \mathbb{R}^{pn_i} \qquad \text{para } r < i \leq m \end{cases}$$

Para exemplificar W_d, dado por (3.69), damos a seguir o correspondente vetor de estado expandido, para a versão D do TA Generalizado do Exemplo 3.2:

$$W_d = \begin{bmatrix} W_1 \\ W_2 \\ W_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_{31} \\ x_{32} \end{bmatrix} \qquad \text{onde} \quad \begin{cases} W_1 = x_1 \\ W_2 = x_2 \\ W_3 = \begin{bmatrix} x_{31} \\ x_{32} \end{bmatrix} \end{cases} \tag{3.70}$$

O Team Algorithm correspondente pode ser representado pelas equações:

$$\begin{aligned} [\, x_i(k+1)\,] &= [\, \widehat{G_i}(\, \hat{x}^i(k)\,)\,] \qquad \text{para } 1 \le i \le r \\ \begin{bmatrix} x_{i1}(k+1) \\ \vdots \\ x_{ip}(k+1) \end{bmatrix} &= \begin{bmatrix} G_{i1}(\, \hat{x}^1(k)\,) \\ \vdots \\ G_{ip}(\, \hat{x}^p(k)\,) \end{bmatrix} \qquad \text{para } r < i \le m \end{aligned} \tag{3.71}$$

onde denotamos como $\hat{x}^j(k)$ ao valor de x utilizado no processador j, na iteração k, ao aplicar os operadores G_{ij}, i.e.

$$\hat{x}^j(k) = \begin{bmatrix} \hat{x}_1^j(k) \\ \vdots \\ \hat{x}_m^j(k) \end{bmatrix}, \qquad \forall j \in \{1, \ldots, p\}; \tag{3.72}$$

$$\text{onde} \quad \hat{x}_i^j(k) = \begin{cases} x_i(\, d_i^j(k)\,) & \text{para } 1 \le i \le r \\ \sum_{l=1}^{p} w_{lj}\, x_{il}(\, d_l^j(k)\,) & \text{para } r < i \le m \end{cases}$$

$$\text{com} \quad \sum_{l=1}^{p} w_{lj} = 1, \qquad \forall j \in \{1, \ldots, p\} \tag{3.73}$$

A relação (3.73) pode ser derivada a partir da Nota 3.7, considerando a relação (3.61) e notando que já não tem sentido utilizar um correspondente parâmetro c_l, dado que não existem duas versões de x_l.

Representado o TA com Administrador distribuído implicitamente, podemos calcular a matriz de comparação H_d correspondente:

$$H_d = \left(H_{ij} \right), \qquad i, j = 1, \ldots, m \tag{3.74}$$

onde as submatrizes H_{ij} que representam a dependência de W_i com W_j são dadas por:

$$H_{ij} = \begin{cases} \widehat{l_{ij}} & \in \mathbb{R} \quad \text{para} \quad 1 \leq i \leq r \quad \text{e} \quad 1 \leq j \leq r \\ \widehat{l_{ij}} \left[w_{1i} \cdots w_{pi} \right] & \in \mathbb{R}^{1 \times p} \quad \text{para} \quad 1 \leq i \leq r \quad \text{e} \quad r < j \leq m \\ \begin{bmatrix} l_{i1j} \\ \vdots \\ l_{ipj} \end{bmatrix} & \in \mathbb{R}^{p} \quad \text{para} \quad r < i \leq m \quad \text{e} \quad 1 \leq j \leq r \\ \begin{bmatrix} l_{i1j} w_{11} & \cdots & l_{i1j} w_{p1} \\ \vdots & \ddots & \vdots \\ l_{ipj} w_{1p} & \cdots & l_{ipj} w_{pp} \end{bmatrix} & \in \mathbb{R}^{p \times p} \quad \text{para} \quad r < i \leq m \quad \text{e} \quad r < j \leq m \end{cases}$$

Como exemplo, a matriz de comparação H_d da versão D do TA Generalizado do Exemplo 3.2, é dada por:

$$H_d = \begin{bmatrix} \widehat{l_{11}} & \widehat{l_{12}} & \widehat{l_{13}} w_{11} & \widehat{l_{13}} w_{21} \\ \widehat{l_{21}} & \widehat{l_{22}} & \widehat{l_{23}} w_{12} & \widehat{l_{23}} w_{22} \\ l_{311} & l_{312} & l_{313} w_{11} & l_{313} w_{21} \\ l_{321} & l_{322} & l_{323} w_{12} & l_{323} w_{22} \end{bmatrix} \tag{3.75}$$

Calculada a matriz de comparação H_d, que é a de menor dimensão de entre todas as estudadas até aqui, e verificado que conforme à equação (3.71) $G_{ta}(W_d) \subset D_w$, podemos escrever o seguinte corolário derivado do Teorema 2.1:

COROLÁRIO 3.6 *A versão assíncrona do Team Algorithm Generalizado, com Administrador distribuído implicitamente (versão D), dado por (3.71), converge para a única solução x^* em D, sob as hipóteses 3.1 a 3.4, se $\rho(H_d) < 1$, sendo a matriz de comparação H_d correspondente, dada por (3.74).*

Enfatiza-se que, utilizando a metodologia desta seção, é possível derivar uma condição suficiente de convergência, semelhante àquelas dadas nos corolários 3.4 a 3.6, para cada possível implementação de um TA Generalizado. Os casos estudados foram escolhidos por serem os mais intuitivos, mas lembramos que outras alternativas foram discutidas no Exemplo 3.3.

Nota 3.8 Como o bloco TA estudado no Capítulo 2 é um caso particular do Team Algorithm Generalizado, podemos verificar que as matrices de comparação H das diferentes versões estudadas, dadas por (3.62), (3.67) e (3.74), coincidem com a correspondente matriz de comparação derivada no Capítulo 2, quando $\Psi = I$.

3.4 Comparação das diferentes versões em termos de convergência: um caso típico

Para finalizar este capítulo, apresenta-se um exemplo no qual aplicamos as equações gerais derivadas na seção anterior, com o objetivo de comparar os raios espectrais das diferentes versões, e tentar assim ter uma idéia de qual seria a implementação mais eficiente do ponto de vista da convergência.

Exemplo 3.4 Resolver o problema $\Phi(x) = 0$ dado pelas equações (2.1) a (2.6), utilizando uma implementação assíncrona, para o caso em que $m = 3$ e

$$\Phi(x) = \begin{bmatrix} \Phi_1(x_1, x_2) \\ \Phi_2(x_1, x_2) \\ \Phi_3(x_1, x_2, x_3) \end{bmatrix}$$

utizando dois algoritmos, representados por G_1 e G_2; sabendo que são satisfeitas as hipóteses 3.1 a 3.4. Cada operador G_i considerado tem uma correspondente matriz de comparação H_i dada por:

$$H_i = \begin{bmatrix} l_{1i1} & l_{1i2} & 0 \\ l_{2i1} & l_{2i2} & 0 \\ l_{3i1} & l_{3i2} & l_{3i3} \end{bmatrix}, \qquad i \in \{1,2\} \tag{3.76}$$

com constantes bloco-Lipschitz que satisfazem as seguintes relações para $j \in \{1,2\}$:
$(l_{11j} \leq l_{12j})$ . . . . que favorece a utilização de G_1 para resolver $\Phi_1(x) = 0$.
$(l_{21j} \geq l_{22j})$ . . . . que favorece a utilização de G_2 para resolver $\Phi_2(x) = 0$.
$\begin{pmatrix} l_{31j} \geq l_{32j} \\ \text{e} \\ l_{313} < l_{323} \end{pmatrix}$ o que não permite uma escolha evidente para resolver $\Phi_3(x) = 0$.

Efetivamente, o raio espectral $\rho(H)$ de uma matriz de comparação $H = (h_{ij})$ é, em geral, proporcional a seus elementos h_{ij}. Sendo assim, para assegurar que $\rho(H) < 1$, devemos tentar minimizar os h_{ij} na hora de escolher os algoritmos bloco iterativos que constituem um Team Algorithm. Portanto, considerando que a matriz de comparação do TA é formado com a correspondente linha da matriz H_i (ver Seção 2.3), é natural escolher para cada bloco o algoritmo com menores h_{ij}.

Conseqüentemente, pelas características próprias do problema, é recomendável a escolha de um TA que combine os algoritmos representados por G_1 e G_2, dado que G_1 tem constantes bloco-Lipschitz menores na hora de resolver o subproblema $\Phi_1(x) = 0$, enquanto G_2 minimiza os h_{2j} correspondentes ao subproblema $\Phi_2(x) = 0$. Porém, não resulta evidente a escolha do algoritmo para resolver $\Phi_3(x) = 0$, que pode ser resolvido utilizando um único algoritmo, sem ser até aqui evidente qual dos dois, ou uma combinação de ambos algoritmos utilizando 'overlapping' parcial.

CASO	TEAM ALGORITHM	Parâmetros de Ψ		CONDIÇÃO DE CONVERGÊNCIA
1	sem Administrador	$\psi_{31} = 1,$	$\psi_{32} = 0$	Teorema 2.1
2	sem Administrador	$\psi_{31} = 0,$	$\psi_{32} = 1$	Teorema 2.1
3	Versão A	$\psi_{31} = 1,$	$\psi_{32} = 1$	Corolário 3.4
4	Versão C	$\psi_{31} = 1,$	$\psi_{32} = 1$	Corolário 3.5
5	Versão D	$\psi_{31} = 1,$	$\psi_{32} = 1$	Corolário 3.6

Tabela 3.1: `Alternativas de solução do Exemplo 3.4.`

Analisamos a seguir 5 alternativas de solução, utilizando Team Algorithms, ilustradas na Tabela 3.1, denotando como Ψ_k, $(k = 1, \ldots, 5)$ à matriz de atribuição do caso k que pode ser escrita, em geral, como:

$$\Psi_k = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ \psi_{31} & \psi_{32} \end{bmatrix}, \qquad \forall k \in \{1, \ldots, 5\} \tag{3.77}$$

onde os valores de ψ_{31} e ψ_{32} estão dados na Tabela 3.1.

A seguir estudamos os operadores, matrices de comparação e raios espectrais correspondentes a cada caso da Tabela 3.1, com o objetivo de ordenar os raios espectrais e assim escolher os casos que resultem mais eficientes.

Caso 1: bloco TA sem 'overlapping', com G_1 resolvendo $\Phi_3(x) = 0$.
Este bloco TA pode ser descrito por:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \\ x_3(k+1) \end{bmatrix} = \begin{bmatrix} G_{11}(\, x^1(k)\,) \\ G_{22}(\, x^2(k)\,) \\ G_{31}(\, x^1(k)\,) \end{bmatrix} \tag{3.78}$$

com uma matriz de comparação H_1 dada por:

$$H_1 = \begin{bmatrix} l_{111} & l_{112} & 0 \\ l_{221} & l_{222} & 0 \\ l_{311} & l_{312} & l_{313} \end{bmatrix} \tag{3.79}$$

onde $h_{13} = h_{23} = 0$ pois nem Φ_1 nem Φ_2 são funções de x_3.

Para o cálculo de $\rho(H_1)$ utilizamos o seguinte lema (apresentado em Horn e Johnson, 1988; pp. 62):

LEMA 3.1 (RAIO ESPECTRAL DE UMA MATRIZ BLOCO TRIANGULAR)
Seja a matriz bloco triangular

$$H = \begin{bmatrix} H_{11} & 0 \\ H_{21} & H_{22} \end{bmatrix} \in \mathbb{R}^{n \times n} \qquad \textit{onde} \quad H_{ii} \in \mathbb{R}^{n_i \times n_i} \quad \textit{e} \quad n_1 + n_2 = n,$$

então: $\rho(H) = \max_{i \in \{1,2\}} \rho(H_{ii})$.

Podemos notar assim que o Lema 3.1 pode ser aplicado à matriz de comparação H_1 dada por (3.79) fazendo:

$$H_{11} = \begin{bmatrix} l_{111} & l_{112} \\ l_{221} & l_{222} \end{bmatrix}, \qquad \text{e} \quad H_{22} = [l_{313}]$$

Sendo assim, podemos escrever $\rho(H_1)$ como:

$$\rho(H_1) \;=\; \max\left\{ \sigma;\; l_{313} \right\} \tag{3.80}$$

onde

$$\sigma \;\stackrel{\text{def}}{=}\; \rho(\, H_{11}\,) \tag{3.81}$$

Caso 2: bloco TA sem 'overlapping', com G_2 resolvendo $\Phi_3(x) = 0$.
A semelhança do caso anterior, este bloco TA pode ser descrito por:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \\ x_3(k+1) \end{bmatrix} = \begin{bmatrix} G_{11}(\, x^1(k)\,) \\ G_{22}(\, x^2(k)\,) \\ G_{32}(\, x^2(k)\,) \end{bmatrix} \tag{3.82}$$

com uma matriz de comparação H_2 dada por:

$$H_2 = \begin{bmatrix} l_{111} & l_{112} & 0 \\ l_{221} & l_{222} & 0 \\ l_{321} & l_{322} & l_{323} \end{bmatrix} \tag{3.83}$$

e conseqüentemente,

$$\rho(H_2) \;=\; \max\left\{ \sigma;\; l_{323} \right\} \tag{3.84}$$

onde σ foi definido em (3.81). Comparando os valores de $\rho(H_1)$ e $\rho(H_2)$; e considerando que em nosso exemplo $l_{313} < l_{323}$, concluimos que:

$$\rho(H_1) \;\leq \rho(H_2) \tag{3.85}$$

o que sugere que o TA, do caso 1, converge em menor número de iterações que o TA do caso 2. Ademais, existem problemas para os quais somente o TA do caso 1 consegue assegurar a convergência conforme o Teorema 2.1.

Analisa-se a seguir os TA com Administrador, utilizando 'overlapping' parcial, formalizados na Seção 3.3. Para isto, enfatizamos que as diferentes versões do TA utilizando 'overlapping' parcial, do Exemplo 3.4, tem associadas a mesma matriz de atribuição (3.77), que a sua vez é idéntica à matriz de atribuição do Exemplo 3.2. Na análise que segue somente se consideram pesos constantes não negativos.

Caso 3: Versão A do TA com Administrador centralizado em processador dedicado.
Este TA pode ser representado por (3.60):

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \\ x_3(k+1) \\ \chi_{31}(k+1) \\ \chi_{32}(k+1) \end{bmatrix} = \begin{bmatrix} G_{11}(\, x^1(k)\,) \\ G_{22}(\, x^2(k)\,) \\ cx_3(k) + w_1\chi_{31}(\, d_1^3(k)\,) + w_2\chi_{32}(\, d_2^3(k)\,) \\ G_{31}(\, x^1(k)\,) \\ G_{32}(\, x^2(k)\,) \end{bmatrix} \tag{3.86}$$

que tem um vetor de estado expandido dado por (3.59). Portanto, a matriz de comparação H_3 é dada por (3.62):

$$H_3 = \begin{bmatrix} l_{111} & l_{112} & 0 & 0 & 0 \\ l_{221} & l_{222} & 0 & 0 & 0 \\ 0 & 0 & |c| & w_1 & w_2 \\ l_{311} & l_{312} & l_{313} & 0 & 0 \\ l_{321} & l_{322} & l_{323} & 0 & 0 \end{bmatrix} \tag{3.87}$$

Para simplificar o cálculo de $\rho(H_3)$ assumimos que os pesos são escolhidos de forma que $w_1 + w_2 = 1$, o que implica $c = 0$. Desta forma, aplicando o Lema 3.1 e realizando alguns cálculos obtemos:

$$\rho(H_3) \;=\; \max\left\{ \sigma;\; \sqrt{w_1 l_{313} \;+\; w_2 l_{323}} \right\} \tag{3.88}$$

onde σ foi definido em (3.81).

Caso 4: Versão C do TA com Administrador distribuído e replicado.
Este TA pode ser representado por (3.66):

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \\ x_{31}(k+1) \\ x_{32}(k+1) \\ \chi_{31}(k+1) \\ \chi_{32}(k+1) \end{bmatrix} = \begin{bmatrix} G_{11}(\, x^1(k)\,) \\ G_{22}(\, x^2(k)\,) \\ c_1 x_{31}(k) + w_{11}\chi_{31}(k) + w_{21}\chi_{32}(\, d_2^1(k)\,) \\ c_2 x_{32}(k) + w_{12}\chi_{31}(\, d_1^2(k)\,) + w_{22}\chi_{32}(k) \\ G_{31}(\, x^1(k)\,) \\ G_{32}(\, x^2(k)\,) \end{bmatrix} \tag{3.89}$$

onde

$$x^1(k) \;=\; \begin{bmatrix} x_1(k) \\ x_2(\, d_2^1(k)\,) \\ x_{31}(k) \end{bmatrix}, \qquad \text{e} \qquad x^2(k) \;=\; \begin{bmatrix} x_1(\, d_1^2(k)\,) \\ x_2(k) \\ x_{32}(k) \end{bmatrix}$$

assim, a matriz de comparação H_4 é dada por (3.67):

$$H_4 = \begin{bmatrix} l_{111} & l_{112} & 0 & 0 & 0 & 0 \\ l_{221} & l_{222} & 0 & 0 & 0 & 0 \\ 0 & 0 & |c_1| & 0 & w_{11} & w_{21} \\ 0 & 0 & 0 & |c_2| & w_{12} & w_{22} \\ l_{311} & l_{312} & l_{313} & 0 & 0 & 0 \\ l_{321} & l_{322} & 0 & l_{323} & 0 & 0 \end{bmatrix} \tag{3.90}$$

que tem um vetor de estado expandido dado por (3.65).

Para simplificar o cálculo de $\rho(H_4)$, assumimos a mesma escolha de pesos feita no caso 3, em todos os processadores, i.e.,

$$w_{11} = w_{12} = w_1 \quad \text{e} \quad w_{21} = w_{22} = w_2 \qquad \text{onde} \quad w_1 + w_2 = 1 \tag{3.91}$$

Desta forma, aplicando o Lema 3.1 e realizando alguns cálculos obtemos:

$$\rho(H_4) \;=\; \max\left\{\sigma;\;\; \sqrt{w_1 l_{313} \;+\; w_2 l_{323}}\right\} \tag{3.92}$$

Caso 5: Versão D do TA com Administrador distribuído implícito.
Este TA tem um vetor de estado expandido dado por (3.70) e pode ser representado por (3.71):

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \\ x_{31}(k+1) \\ x_{32}(k+1) \end{bmatrix} \;=\; \begin{bmatrix} G_{11}(\,\hat{x}^1(k)\,) \\ G_{22}(\,\hat{x}^2(k)\,) \\ G_{31}(\,\hat{x}^1(k)\,) \\ G_{32}(\,\hat{x}^2(k)\,) \end{bmatrix} \tag{3.93}$$

onde

$$\hat{x}^1(k) \;=\; \begin{bmatrix} x_1(k) \\ x_2(\,d_2^1(k)\,) \\ w_{11}\,x_{31}(k) \;+\; w_{21}\,x_{32}(\,d_2^1(k)\,) \end{bmatrix},$$

e

$$\hat{x}^2(k) \;=\; \begin{bmatrix} x_1(\,d_1^2(k)\,) \\ x_2(k) \\ w_{12}\,x_{31}(\,d_1^2(k)\,) \;+\; w_{22}\,x_{32}(k) \end{bmatrix};$$

portanto, a matriz de comparação H_5 é dada por (3.74):

$$H_5 = \begin{bmatrix} l_{111} & l_{112} & 0 & 0 \\ l_{221} & l_{222} & 0 & 0 \\ l_{311} & l_{312} & w_{11}l_{313} & w_{21}l_{313} \\ l_{321} & l_{322} & w_{12}l_{323} & w_{22}l_{323} \end{bmatrix} \tag{3.94}$$

Para simplificar o cálculo de $\rho(H_5)$, assumimos a mesma escolha de pesos (3.91) feita para o caso 4; assim, aplicando o Lema 3.1 e realizando alguns cálculos obtemos:

$$\rho(H_5) \;=\; \max\left\{\sigma;\quad (w_1 l_{313} \;+\; w_2 l_{323})\right\} \tag{3.95}$$

Para comparar os raios espectráis até aqui calculados, consideramos que o bloco TA do caso 1 tem $\rho(H_1) < 1$, enquanto $\rho(H_2) \geq 1$, e que os pesos w_1 e w_2 são apropriadamente escolhidos, de forma que os correspondentes TA Generalizados (com 'overlapping' parcial) tenham um raio espectral menor que um. Sendo assim, podemos afirmar que:

$$\rho(H_1) \;\leq\; \rho(H_5) \;\leq\; \rho(H_3) \;=\; \rho(H_4) \;<\; 1 \;\leq\; \rho(H_2) \tag{3.96}$$

Conclusões da Análise do Exemplo 3.4

- De todos os casos estudados, o bloco TA do caso 1, que resolve $\Phi_3(x) = 0$ utilizando G_1 (sem 'overlapping'), é o de menor raio espectral e por tanto uma opção recomendável. Porém, se o algoritmo G_1 é mais lento que o G_2 (por exemplo, por utilizar maior número de operações por iteração), qualquer TA Generalizado (casos 3,4 e 5), utilizando 'overlapping' parcial, consegue combinar ambos algoritmos de forma a obter um raio espectral menor que 1, para uma escolha apropriada dos pesos.

- A utilização de um Administrador implícito permite melhorar o raio espectral com respeito aos TA com Administradores explícitos, por eliminar da matriz de comparação as linhas correspondentes aos Administradores (linha com os pesos), que caracterizam-se por ter $\sum_j h_{ij} \geq 1$. Assim, no Exemplo 3.4, $\rho(H_4) = \sqrt{\rho(H_5)} < 1$, de onde $\rho(H_5) < \rho(H_4)$.

Nota 3.9 Enfatizamos que o raio espectral somente dá uma idéia do número de iterações até a convergência. A recomendação final para a utilização de um dado algoritmo deve considerar também o número de operações por iteração.

Sumário do Capítulo 3

Neste capítulo foram formalizados, no contexto assíncrono dos sistemas distribuídos, os *Team Algorithms Generalizados* que incluem como casos particulares aos bloco TA estudados no Capítulo 2, a proposta de Talukdar et al. (1983) que considera o uso de um Administrador centralizado operando sobre todo o vetor x, assim como o conceito de 'overlapping' parcial proposto em Ikeda e Šiljak (1980).

Primeiramente, foi analisada a proposta de Talukdar et al. (1983), considerando várias políticas para a escolha dos pesos utilizados no Administrador; comprovando-se experimentalmente que a escolha de pesos constantes se mostra satisfatoria em geral, quando os pesos são escolhidos corretamente. Porém, a política de minimização do resíduo em norma infinito pode ser recomendada quando não é possível fazer uma escolha 'a priori' dos pesos constantes.

Os TA Generalizados foram introduzidos na Seção 3.2 como uma formulação que engloba os diferentes métodos citados acima, sendo que essa generalização permite obter as seguintes vantagens:

- Resolver sistemas de equações algébricas que os algoritmos sendo combinados não resolvem isoladamente.

VERSÃO	TIPO DE ADMINISTRADOR		
	implícito / explícito	centralizado / distribuído	Processador
A	explícito	centralizado	dedicado
B	explícito	centralizado	1
C	explícito	distribuído	replicado
D	implícito	distribuído	replicado
E	implícito	centralizado	1

Tabela 3.2: `Versões assíncronas dos TA utilizando 'overlapping'.`

- Viabilizar a resolução eficiênte de sistemas de equações algébricas de grande porte em sistemas computacionais de memória distribuída, caracterizados por seu baixo custo por instrução.

- Permitir um considerável 'Speedup', quando problemas de grande porte são apropriadamente particionados em subproblemas menores que podem ser resolvidos em paralelo. O balanceamento de carga computacional não se mostra tão crítico nas versões assíncronas (comparadas com a correspondente versão síncrona).

- Possibilitar a utilização de algoritmos rápidos que nem sempre convergem no domínio de interesse. Para isto, um TA pode combinar o algoritmo rápido com um outro de boa convergência no domínio considerado, mesmo que mais lento, e aproveitar assim a velocidade do primeiro.

- Evitar a divergência prematura dos processos encarregados do cálculo de alguns subvetores. Isto é possível como conseqüência de uma apropriada escolha de pesos no Administrador.

- Facilitar a resolução de sistemas de equações com *blocos críticos*, fortemente acoplados a outros blocos, como conseqüência direta da utilização de 'overlapping' parcial.

Foram estudadas várias versões de TA Generalizados utilizando 'overlapping' parcial, que resumimos na Tabela 3.2. A título de exemplo, três versões foram formalizadas para o caso geral, num contexto assíncrono, derivando-se para cada uma destas versões, uma condição suficiente de convergência.

Vários exemplos foram apresentados, enfatizando-se as características dos TA Generalizados. Assim, foram comparadas as diversas versões de TA Generalizados com 'overlapping' parcial, que apresentaram em geral propriedades similares. Porém, as versões com Administradores implícitos, resultaram em matrizes de comparação com os menores raios espectrais.

Capítulo 4

Problemas Exemplos

A utilização dos *Team Algorithms* num contexto assíncrono, foi até aqui postulada e discutida dando exemplos de dimensões relativamente reduzidas, apessar de termos afirmado que os TA apresentam as maiores vantagens na resolução de sistemas de grande porte. Para embasar experimentalmente esta afirmação, escolhemos um problema exemplo de porte médio (616 equações), de solução seqüencial bem conhecida: o problema de Fluxo de Potência Elétrica (Stott, 1974; Monticelli, 1983). A escolha deste problema exemplo foi feita com base em razões históricas e práticas:

razões histórias: a primeira proposta do que poderia ser considerado um bloco TA, para resolver um problema de engenharía, é o trabalho de Dusonchet et al. (1971) que propõe justamente resolver o problema do fluxo de potência elétrica. Também a primeira idéia de um TA com Administrador foi publicado num relatório da área elétrica ('EPRI Report', Talukdar et al., 1983);

razões práticas: pelo interesse que a solução deste problema apresenta no contexto da análise de sistemas elétricos, bem como pela disponibilidade de dados reais conhecidos de sistemas de médio e grande porte.

Escolhido o problema exemplo, este é descrito na seção 4.1 juntamente com os principais algoritmos hoje utilizados na sua solução em máquinas seqüenciais. Estes algoritmos são assim formalizados num contexto bloco assíncrono, derivando-se também condições suficientes de convergência nesse contexto. Uma proposta para utilizar um Team Algorithm que combine os métodos bloco Jacobi e Desacoplado Rápido é analisada, no contexto assíncrono, derivando-se uma condição suficiente de convergência para o mesmo.

Dado nosso interesse num contexto paralelo assíncrono, a seção 4.2 descreve o ambiente computacional utilizado em nossa implementação, incluindo detalhes do hardware e do software. Para enfatizar as condições sob as quais um TA apresenta as maiores vantagens, discute-se dois casos diferentes. O primeiro, é resolvido por um bloco TA sem 'overlapping', cujos detalhes de implementação são descritos na seção 4.3. O

segundo caso, por ser um caso de convergência mais problemática, não pode ser resolvido por um bloco TA e requer da utilização de um TA com Administrador para encontrar a solução. As diferentes versões de um *Team Algorithm Generalizado* utilizando 'overlapping' parcial são apresentadas, para este caso, na seção 4.4. Os resultados experimentais dessas implementações são dados no Capítulo 5.

4.1 Fluxo de Potência Elétrica

O problema do Fluxo de Potência Elétrica (Stott, 1974) é também conhecido como o problema do Cálculo do Fluxo de Carga (Monticelli, 1983). Basicamente, o problema consiste na determinação de tensões, em magnitude e fase, dos barramentos de uma rede elétrica, para uma dada condição de geração e cargas especificadas. A partir destas tensões, é possível determinar o carregamento das linhas e outras grandezas de interesse. O problema utiliza uma modelagem estática da rede o que permite a sua representação por um conjunto de equações algébricas.

O problema do fluxo de potência elétrica é resolvido em várias áreas do setor elétrico, como o planejamento, a operação e o controle do sistema. Por sua importância, métodos cada vez mais sofisticados foram criados para resolvé-lo. No periodo de 1930 a 1956 utilizaram-se *analisadores de circuitos* (ou 'Network Analysers') para resolver o citado problema. Estes eram basicamente modelos reduzidos da rede em estudo.

As primeiras tentativas de resolver o problema do fluxo de potência elétrica utilizando computadores seqüenciais, foram realizadas na década de 50; porem, com sucesso limitado. O primeiro programa computacional bem sucedido, foi desenvolvido por Ward e Hale (1956), implementando um método iterativo derivado do método de Newton. Os programas que se seguiram, continuaram utilizando métodos iterativos, popularizando-se na década de 50 o método da matriz Y (detalhes em Stott, 1974).

A década de 60 trouxe grandes avanços nas *técnicas de esparsidade* (Duff, 1981; Morozowski, 1981) que foram especialmente bem aproveitados pelo método de Newton-Raphson proposto em Tinney e Hart (1967), que passou a ser o mais bem sucedido da década. A incorporação de características próprias da rede no método de Newton levou a postular o método desacoplado por Stott (1972). Aperfeiçoamentos deste método permitiram desenvolver o método Desacoplado Rápido apresentado em Stott e Alsaç (1974) que domina até hoje as utilizações comerciais.

Centenas de trabalhos foram escritos estudando o problema do fluxo de carga (veja referências em Stott, 1974) durante estas décadas, o que levou a um amadurecimento dos métodos de solução no contexto seqüencial. Contudo, postula-se nesta seção, para um contexto paralelo assíncrono, a utilização de um *Team Algorithm* que combina a versão bloco Jacobi do método da matriz Y com o método Desacoplado Rápido, para problemas em que nenhum dos métodos sendo combinados consegue isoladamente resolver com bom desempenho todas as partes do sistema.

Considerando que para resolver o problema do Fluxo de Potência Elétrica para sistemas de grande porte hoje são utilizados 'mainframes', a principal motivação desta seção é a formalização e análise dos diversos métodos de resolução do problema do Fluxo de Potência Elétrica, no contexto dos algoritmos bloco iterativos assíncronos, descrito na Capítulo 2, de forma a viabilizar a solução deste tipo de problema em maquinas paralelas de custo reduzido.

A proposta de utilizar um Team Algorithm para resolver o problema do fluxo de potência elétrica não tem por objetivo formular um método que apresente um desempenho superior em todos os problemas práticos; mas simplesmente, demonstrar que existem casos para os quais um TA permite resolver problemas em que os métodos envolvidos não conseguem resolver isoladamente ou o fazem com um desempenho notavelmente inferior.

4.1.1 Formulação Matemática do Problema

O problema do Fluxo de Potência Elétrica pode ser formulado como um sistema quase-linear de equações do tipo $Ax = F(x)$ (Bhaya et al., 1991) onde A está dada pela *matriz de Admitância:*

$$Y = \{y_{ki}\} \in \mathbf{C}^{n \times n}, \qquad \text{com} \quad y_{ki} = G_{ki} + jB_{ki} \in \mathbf{C}; \tag{4.1}$$

onde $x \in \mathbf{C}^n$ representa o vetor de voltagens, e $F(x)$ representa o vetor de corrente $I \in \mathbf{C}^n$. Sendo assim, o problema pode ser formulado matemáticamente como:

$$Y\,x \;=\; I(x) \tag{4.2}$$

Note que a tensão x_k do nó k pode ser dada em coordenadas polares, por sua magnitude V_k e fase θ_k, i.e.

$$x_k \;=\; V_k\, e^{j\theta_k} \quad \in \mathbf{C}, \qquad\qquad \forall k \in \{1, \ldots, n\} \tag{4.3}$$

A situação de regime em um sistema elétrico pode ser representada pela equação (4.2) juntamente com um conjunto de restrições operacionais sobre as potências e voltagens dos nós (ou barras) da rede elétrica. Na resolução do problema, distingue-se três tipos de barras segum sejam as magnitudes especificadas para essa barra (Stott, 1974). Assim, uma dada barra k pode ser:

barra PQ ou *barra de carga*, quando a potência complexa injetada $S_k = P_k + jQ_k$ é especificada;

barra PV ou *barra de tensão controlada*, quando a potência ativa P_k e a magnitude da tensão V_k são especificadas para essa barra;

barra slack ou *barra de referência*, quando a própria tensão x_k da barra (ou nó) é especificada.

A *barra de referência*, é um conceito fictício criado com dupla função: fornecer a referência angular do sistema, sendo a própria terra a referência de magnitude; além de permitir fechar o balanço de potência do sistema, levando em conta as perdas de transmissão não conhecidas antes de se ter a solução final; daí a necessidade de se dispor de uma barra na qual não é especificada a potência ativa.

Dado que a corrente $I_k(x_k)$ em uma barra k não é especificada, ela pode ser calculada como abaixo:

$$I_k(x_k) = \left(\frac{S_k}{x_k}\right)^* = \frac{P_k - jQ_k}{V_k\, e^{-j\theta_k}}, \qquad \forall k \in \{1,\ldots,n\} \tag{4.4}$$

onde $*$ representa o complexo conjugado.

Para vários métodos de solução aplicados a problemas com barras de tensão controlada, a formulação do problema em coordenadas cartesianas, segundo (4.2), não se mostra conveniente pois a magnitude da tensão a ser controlada não aparece em forma explícita. Estes métodos formulam o problema do fluxo de potência como a seguir (veja Monticelli, 1983):

$$\begin{cases} P_k &= V_k \sum_{i\in\mathbf{K}} V_i \left(G_{ki} \cos\theta_{ki} + B_{ki} \sin\theta_{ki} \right) \\ Q_k &= V_k \sum_{i\in\mathbf{K}} V_i \left(G_{ki} \sin\theta_{ki} - B_{ki} \cos\theta_{ki} \right) \end{cases} \qquad \forall k \in \{1,\ldots,n\} \tag{4.5}$$

onde $\mathbf{K}$ é o conjunto de todas as barras adjacentes à barra k, incluindo a própria barra k; e θ_{ki} representa a diferença de fase entre as barras k e i, i.e. $\theta_{ki} = \theta_k - \theta_i$.

O sistema não linear de equações dado por (4.2), ou de forma equivalente por (4.5), pode ser resolvido utilizando diferentes métodos iterativos descritos na literatura (ver Stott, 1974), destacando-se o método da matriz Y, o método da matriz Z e as variações do método de Newton. Dentre estes métodos, estamos interessados em dois que serão combinados para formar um *Team Algorithm*, utilizando a filosofia do Capítulo 3, ou seja:

o Desacoplado Rápido é o mais utilizado comercialmente por suas excelentes características de convergência na maioria das condições normais de operação. Este método é uma variante aperfeiçoada do método de Newton em coordenadas polares.

o método da matriz Y (numa variante bloco Jacobi (BJ)), por permitir uma paralelização simples do problema considerado e permitir a solução de alguns problemas para os quais o Desacoplado Rápido não consegue convergir adequadamente.

Como pretendemos utilizar métodos iterativos, não podemos deixar de discutir o critério de parada a ser utilizado. Neste sentido, o critério mais utilizado é a partir do *desajuste de potência* (ou 'power mismatch') que deve ser menor que uma dada tolerância ε, onde o desajuste de potência ΔS é definido como:

$$\Delta S \stackrel{\text{def}}{=} \max_i \left\{ |\Delta P_i|;\ |\Delta Q_i| \right\} \quad \text{onde} \quad \Delta P_i + j\Delta Q_i = S_i - x_i I_i^* \tag{4.6}$$

Uma característica importante do problema de fluxo de carga é o alto grau de esparsidade da matriz Y, o que permite utilizar técnicas específicas para obter algoritmos eficientes. Felizmente, estas técnicas estão hoje bem desenvolvidas e existe uma boa literatura disponível (Duff, 1981; Morozowski, 1981).

Como estamos analisando métodos iterativos para resolver o problema de fluxo de carga, devemos lembrar a necessidade de conhecer a região D na qual pretendemos encontrar a solução de interesse, conhecida tecnicamente como *ponto de operação* . Pelo fato do problema em questão ser não linear, a estimação desta região D é de fundamental importância para obter o ponto de operação desejado e excluir soluções falsas ou estranhas (Johnson, 1977); ou mesmo soluções estáveis fora da região de interesse (Korsak, 1972). Adicionalmente, é necessario estimar uma condição inicial $x(0)$ e poder controlar que o algoritmo termine mesmo que ele saia da região desejada. Além disso, se pretendemos utilizar os teoremas e corolários demonstrados para os TA dos capítulos 2 e 3, é fundamental satisfazer a Hipótese 2.1. Afortunadamente, para a grande maioria dos problemas práticos de interesse, esta região D está dada por (veja Wu, 1977):

$$\forall k \in \{1,\ldots,n\} \qquad \begin{cases} \mid \theta_k \mid \leq \theta_{max} \\ \mid V_k - 1 \mid \leq \epsilon \end{cases} \tag{4.7}$$

onde geralmente θ_{max} é de uns dez graus e ϵ é menor que a unidade. A existência deste domínio popularizou a condição inicial $x(0)$ conhecida como 'flat start' que consiste em fazer todos os voltagens desconhecidos iguais a 1 p.u., e as fases iguais a 0.

Considerando nosso interesse em implementações paralelas num contexto assíncrono, se apresenta a seguir o problema do fluxo de potência elétrica dado por (4.2), em uma representação por blocos. Para isto enfatizamos que por simplicidade, no Capítulo 2 consideramos a representação dos sistemas no conjunto dos reais $\mathbb{R}$, todavia todos os conceitos, equações e resultados são facilmente estendidos ao conjunto $\mathbf{C}$ dos complexos.

Formulação por blocos do problema do fluxo de potência elétrica

Na linha da formulação do Capítulo 2, o problema do fluxo de carga dado por (4.2) pode ser escrito como:

$$\Phi(x) = 0, \qquad x \in \mathbf{C}^n, \qquad \Phi : \mathbf{C}^n \to \mathbf{C}^n \tag{4.8}$$

definido num domínio $D \subset \mathbf{C}^n$.

Como o objetivo é resolver este sistema de equações em blocos, no mesmo esquema apresentado na seção 2.1, o sistema é particionado de forma que cada processador *resolva* somente uma parte do sistema completo, comunicando os resultados parciais. Seja então a decomposição cartesiana:

$$\mathbf{C}^n = \mathbf{C}^{n_1} \times \cdots \times \mathbf{C}^{n_m}, \qquad n_1 + \cdots + n_m = n. \tag{4.9}$$

de forma que o domínio D satisfaça

$$D = D_1 \times \cdots \times D_m, \qquad D_i \subset \mathbf{C}^{n_i}, \qquad \forall i \in \{1,\ldots,m\} \tag{4.10}$$

e um vetor $x \in D$ satisfaza (2.4); sendo assim, $\Phi(x)$ é decomposto de acordo com:

$$\Phi(x) = \begin{bmatrix} \Phi_1(x) \\ \vdots \\ \Phi_m(x) \end{bmatrix}, \qquad \Phi_i : D \subset \mathbf{C}^n \rightarrow \mathbf{C}^{n_i} \tag{4.11}$$

Conseqüentemente, o subproblema a ser resolvido em cada processador i é dado por:

$$\Phi_i(x) = 0, \qquad \forall i \in \{1, \ldots, m\} \tag{4.12}$$

que para o problema do fluxo de carga corresponde a:

$$\Phi_i(x) \; = \; \sum_{j=1}^{m} Y_{ij}\, x_j - I_i(x_i) \tag{4.13}$$

onde, $Y_{ij} \in \mathbf{C}^{n_i \times n_j}$ e $I_i : D_i \rightarrow \mathbf{C}^{n_i}$.

Sendo o problema em questão quase-linear, podemos resolver (4.12) utilizando em cada processador i um método representado por seu operador G_i, que pode estar dado por qualquer dos algoritmos estudados na seção 2.2 (ou combinação dos mesmos).

4.1.2 Variante bloco Jacobi do método da matriz Y

O método da matriz Y, analisado por Stott (1974), foi intensamente utilizado nas primeiras implementações computacionais por sua simplicidade e também por exigir mínimos requisitos de memória. Porém, sabe-se que a convergência do método não é boa em sistemas com barras de tensão controlada (Stott, 1974).

A idéia do método é muito simples na versão seqüencial: resolver iterativamente a equação (4.2); para tanto, é necessário primeiramente calcular a corrente I usando a relação (4.4). Para o cálculo da corrente I_k utiliza-se a potência complexa S_k, que é dada nas barras PQ; porém, nas barras PV, Q_k não é conhecida e deve ser estimada, o que dificulta a convergência dos sistemas com barras de tensão controlada. Vários métodos existem na literatura para tratar o caso em que estas barras estão presentes. Contudo, o resultado é sempre o mesmo: o método não converge bem para sistemas com barras PV (Stott, 1974).

Por outro lado, o método é facilmente paralelizável e tem uma convergência aceitável em sistemas sem barras de tensão controlada (resultados experimentais em Bhaya et al., 1991). Para estes problemas sem barras PV, podemos escrever uma versão bloco Jacobi do método da matriz Y, utilizando o algoritmo BJ descrito por (2.38). A esta variante do método da matriz Y chamaremos daqui em diante simplesmente como método bloco Jacobi (ou BJ), pois pode ser formulado fazendo $A = Y$, e particionando o problema conforme (2.32) a (2.34). Assim, escolhendo $M_i = Y_{ii}$ em (2.38), obtemos o seguinte operador G_i a ser utilizado pelo método:

$$G_i(x) \; = \; Y_{ii}^{-1}\, I_i(x_i) \; - \sum_{j=1, j\neq i}^{m} Y_{ii}^{-1}\, Y_{ij}\, x_j \qquad \forall i \in \{1, \ldots m\} \tag{4.14}$$

lembrando que este operador G_i é bloco-Lipschitz contínuo com constantes dadas por (2.40). Para o caso particular (4.14), as constantes bloco-Lipschitz são dadas por:

$$l_{ij} = \begin{cases} l_{F_i}\|Y_{ii}^{-1}\| & \text{se } j = i \\ \\ \|Y_{ii}^{-1}Y_{ij}\| & \text{se } j \neq i \end{cases} \tag{4.15}$$

onde a constante de Lipschitz l_{F_i} da corrente $I_i(x_i)$ pode ser estimada conforme a análise realisada em Bhaya et al. (1991).

Conseqüentemente, o Teorema 2.1 (ou de forma equivalente, o Corolário 2.1) estabelece uma condição suficiente de convergência para uma implementação assíncrona da versão bloco Jacobi do método da matriz Y (ou simplesmente, método BJ).

4.1.3 Método Desacoplado Rápido

O método Desacoplado Rápido (ou DR), originalmente proposto em Stott e Alsaç (1974) com o nome de 'Fast Decoupled Load Flow', é hoje o método mais difundido comercialmente para resolver problemas de fluxo de carga. O método foi sendo aperfeiçoado por décadas e alcançou um nível significativo de eficiência para a grande maioria dos sistemas de energia elétrica.

Como existe uma vasta literatura explicando os detalhes do método, nos limitaremos a apresentar apenas as fórmulas, seguindo a linha de Monticelli (1983), que deriva o método DR a partir do método de Newton em coordenadas polares, fazendo aproximações sobre a matriz Jacobiana. Assim, o método DR consiste em melhorar, a cada iteração, a estimativa das tensões em magnitude e fase, calculando as correções ΔV e $\Delta\theta$ de acordo com:

$$\left[\frac{\Delta P}{V}\right] = [B'][\Delta\theta] \tag{4.16}$$

$$\left[\frac{\Delta Q}{V}\right] = [B"][\Delta V] \tag{4.17}$$

onde B' e $B"$ são matrizes constantes calculadas a partir da matriz de admitância Y; e os desbalanceamentos de potência ΔP e ΔQ são calculados segundo a equação (4.6).

Para justificar as vantagens do método DR, lembramos que o método de Newton, do qual deriva o Desacoplado Rápido, apresenta uma convergência quadrática numa região próxima da solução e portanto, em geral, é o método que converge no menor número de iterações para o problema considerado (Stott, 1974). Afortunadamente, para a grande maioria dos problemas do fluxo de carga, o método DR conserva as excelentes características de convergência do método de Newton, sem a necessidade do cálculo do Jacobiano e de sua inversão a cada iteração. Conseqüentemente, para muitos problemas de interesse, o método DR converge em aproximadamente o mesmo número de iterações que o método de Newton; porém, utilizando um número bem menor de operações por iteração, pois as matrizes B' e $B"$ poden ser invertidas uma única vez no inicio do processo, utilizando técnicas de esparsidade.

Outra importante vantagem do método Desacoplado Rápido é o baixo custo computacional para verificar a terminação do algoritmo, atendendo ao critério do 'power mismatch' apresentado na subseção 4.1.1. Isto porque ΔP e ΔQ já são calculados como parte do algoritmo, enquanto outros métodos, como o método da matriz Y ou o método da matriz Z, necessitam do cálculo correspondente quando é necessário verificar se já foi alcançada uma solução aceitável.

Enfatizamos que, diferentemente do método da matriz Y, o vetor de estado não é dado pelas n tensões complexas representadas pelo vetor $x \in \mathbf{C}^n$, mas por um vetor $\xi \in \mathbb{R}^{n+q}$, onde q representa o número de barras PQ, definido como:

$$\xi \stackrel{\text{def}}{=} \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \\ V_1 \\ \vdots \\ V_q \end{bmatrix} \tag{4.18}$$

o que facilita a solução de sistemas com barras de tensão controlada, pois as magnitudes das tensões aparecem explicitamente na formulação do problema dado em (4.5), que serve de base para a derivação do método. Sendo assim, o método converge sem problemas para a grande maioria dos problemas práticos, especialmente em sistemas com barras PV. Ainda mais, o método consegue resolver problemas que nem mesmo o método de Newton resolve (e vice versa).

Contudo, existem situações nas quais o método não tem uma boa convergência (vide discussão em Wu, 1977) e que basicamente podem ser reduzidas a dois casos:

- ramos com valores elevados da relação R/X (resistência/reatância);
- elevadas diferenças de fases θ_{ki} entre barras adjacentes.

Afortunadamente, no primeiro dos casos, o problema pode ser muitas vezes superado utilizando nós fictícios (conforme recomendações de Dy Liacco T.E., dadas em Wu, 1977), quando são conhecidas 'a priori' os ramos com elevada relação R/X.

A convergência da versão seqüencial do método DR foi estudada por Wu (1977). Nesse trabalho demonstra-se que, se representamos o método por um operador ϕ, i.e.

$$\xi(k+1) = \phi\big(\xi(k)\big) \qquad \xi \in D_\xi \tag{4.19}$$

onde D_ξ está dado por (4.7); então, o operador ϕ é Lipschitz contínuo na norma$-\infty$ em D_ξ, com constante C_w; ou seja:

$$\forall \xi_1, \xi_2 \in D_\xi, \qquad \| \phi(\xi_1) - \phi(\xi_2) \|_\infty \leq C_w \| \xi_1 - \xi_2 \|_\infty \tag{4.20}$$

onde a constante C_w será doravante denominada *constante de Wu*. C_w é função dos dados do problema (B', B'', q, P e Q) e os parâmetros ϵ e θ_{max} do domínio D_ξ. Note que, se $C_w < 1$, o operador ϕ é contrativo e a convergência do método está assegurada pelo Teorema 2.3. Tal fato leva a postular o seguinte lema, demonstrado em Wu (1977):

LEMA 4.1 (CONVERGÊNCIA DO MÉTODO DESACOPLADO RÁPIDO)
Se para um dado problema do fluxo de potência, com condição inicial de 'flat start', se tem uma Constante de Wu menor que 1, e ademais os $\Delta\theta(0)$ *e* $\Delta V(0)$ *satisfazem a relação:*

$$\max_k \left\{ \frac{\epsilon\,|\Delta\theta_k(0)|}{\theta_{max}};\quad |\Delta V_k(0)| \right\} < (1 - C_w)\,\epsilon$$

então, o problema do fluxo de potência elétrica tem uma única solução em D_ξ *e o método Desacoplado Rápido converge geometricamente para esta solução.*

A discussão do método DR terminaria aqui a não ser pelo nosso interesse nas implementações paralelas, e mais especificamente, na combinação bloco assíncrona de algoritmos diferentes. Assim, se queremos combinar um método que utiliza um vetor de estado x para representar as tensões (como o método da matriz Y) com o método DR que utiliza o vetor ξ, devemos conhecer a relação que existe entre ambos. Derivamos neste contexto um lema que permite relacionar estes dois vetores de estado:

LEMA 4.2 *seja* $D_x \subset \mathbf{C}$ *o conjunto de tensões complexas* $x_k = V_k\, e^{j\theta_k}$ *que satisfazem (4.7), e seja* $D_\xi \subset \mathbb{R}^2$ *o conjunto correspondente de vetores* $\xi_k = [V_k,\, \theta_k]^T$ *que também satisfazem a mesma condição; então, existem constantes* C_ξ *e* C_x *que satisfazem as seguintes relações:*

$$\forall x_1, x_2 \in D_x, \qquad |x_1 - x_2| \le C_\xi \|\xi_1 - \xi_2\|_\infty \tag{4.21}$$

$$\forall \xi_1, \xi_2 \in D_\xi, \qquad \|\xi_1 - \xi_2\|_\infty \le C_x |x_1 - x_2| \tag{4.22}$$

onde $|\cdot|$ *representa o módulo de um número complexo (i.e.,* $|x_k| = V_k$*). Uma estimativa destas constantes é dada por:*

$$C_x = \frac{1}{(1 - \epsilon)} \left(\frac{\theta_{max}}{\sin\theta_{max}} \right) \qquad e \qquad C_\xi = \sqrt{2 + 2\epsilon + \epsilon^2} \tag{4.23}$$

Prova.
Seja $x_1, x_2 \in D_x$ e $\xi_1, \xi_2 \in D_\xi$, então:

$$|\Delta x|^2 = |x_1 - x_2|^2 = V_1^2 + V_2^2 - 2V_1V_2\cos\Delta\theta = \Delta V^2 + 4V_1V_2\sin^2\left(\frac{\Delta\theta}{2}\right)$$

Parte 1: demonstração para C_ξ
por ser $\sin\beta \le \beta$ $(\beta \in \mathbb{R})$ *temos:*

$$|\Delta x|^2 \le \Delta V^2 + V_1V_2\Delta\theta^2 \le \Delta V^2 + (1 + \epsilon)^2\Delta\theta^2 \le [1 + (1 + \epsilon)^2]\max(\Delta V^2;\, \Delta\theta^2)$$

conseqüentemente, $|\Delta x| \leq C_\xi \|\Delta\xi\|_\infty$

Parte 2: demonstração para C_x
Para verificar que $\|\Delta\xi\|_\infty \leq C_x|\Delta x|$ *temos que satisfazer as seguintes desigualdades:*

$$\begin{cases} (a) & \Delta V^2 \leq C_x^2 \, |\Delta x|^2 \\ (b) & \Delta\theta^2 \leq C_x^2 \, |\Delta x|^2 \end{cases}$$

parte (a) $\quad \Delta V^2 \leq \Delta V^2 + 4V_1V_2 \sin^2\left(\frac{\Delta\theta}{2}\right) = |\Delta x|^2 \leq C_x^2|\Delta x|^2$

parte (*b*) *por ser* $\frac{\Delta\theta}{2} \leq \theta_{max} < \frac{\pi}{2}$ *temos que* $\frac{\sin\theta_{max}}{\theta_{max}} \leq \frac{\sin(\Delta\theta/2)}{(\Delta\theta/2)}$, *logo:*

$$\Delta\theta^2 \leq \frac{4\theta_{max}^2}{\sin^2\theta_{max}} \sin^2\left(\frac{\Delta\theta}{2}\right) \leq \frac{4\theta_{max}^2}{\sin^2\theta_{max}} \frac{V_1V_2}{(1-\epsilon)^2} \sin^2\left(\frac{\Delta\theta}{2}\right)$$

conseqüentemente,

$$\Delta\theta^2 \leq \frac{\theta_{max}^2}{\sin^2\theta_{max}} \frac{1}{(1-\epsilon)^2} \left[\Delta V^2 + 4V_1V_2 \sin^2\left(\frac{\Delta\theta}{2}\right)\right] = C_x^2 \, |\Delta x|^2$$

■

Como existe uma relação bi-unívoca entre x e ξ para um dado problema de fluxo de potência elétrica, podemos pensar no método DR não somente como um operador ϕ sobre ξ, mas também como um operador G_{DR} sobre x. Neste caso, o método DR na versão seqüencial poderia ser representado por:

$$x(k+1) = G_{DR}\Big(x(k)\Big) \tag{4.24}$$

o que leva a demonstrar o seguinte lema:

LEMA 4.3 (MÉTODO DR VISTO COMO UM OPERADOR CONTRATIVO)
O operador $G_{DR}(x)$ *que representa o método Desacoplado Rápido operando em coordenadas cartesianas é um operador contrativo no domínio* D *definido conforme (4.7) em norma–*∞*, se*

$$C_{DR} \stackrel{\text{def}}{=} C_\xi \, C_w \, C_x < 1$$

Prova.

$\forall x, y \in D,$

$$\begin{aligned} \|G_{DR}(x) - G_{DR}(y)\|_\infty &\leq C_\xi \, \|\phi(\xi_x) - \phi(\xi_y)\|_\infty \\ &\leq C_\xi \, C_w \, \|\xi_x - \xi_y\|_\infty \\ &\leq C_\xi \, C_w \, C_x \, \|x - y\|_\infty \end{aligned}$$

■

Nota 4.1 Para os valores de C_x e C_ξ dados por (4.23), tem-se que $C_{DR} > C_w$ e portanto o lema 4.3 é bem mais conservativo que o Lema 4.1. Porém, o Lema 4.3 será útil para se derivar condições suficientes de convergência de algoritmos combinados para o problema em questão, num contexto assíncrono.

Enfatizamos aqui que o lema 4.3 permite analisar o método DR como um operador contrativo, quando aplicado ao problema dado por (4.2). Analogamente, se consideramos a decomposição do problema, dada por (4.11) a (4.13), podemos aplicar o método DR a cada subproblema $\Phi_i(x) = 0$ dado por (4.12), considerando as tensões não atualizadas em cada processador i como constantes a cada iteração, dando a estas barras o mesmo tratamento que à barra slack (tensão conhecida). Desta forma, o Lema 4.3 aplicado a cada bloco, permite calcular os módulos (ou constantes contrativas) dos operadores aplicados a cada subvetor. Assim, se cada processador i aplica um operador contrativo G_{DRi} de módulo C_{DRi} a seu respectivo subproblema $\Phi_i(x) = 0$, no contexto assíncrono, estamos em presença do método das *Iterações Contrativas*, discutido na subseção 2.2.4, justamente para sistemas quase-lineares, como é o caso do problema do fluxo de carga.

Conseqüentemente, o Corolário 2.4 estabelece uma condição suficiente de convergência para a versão bloco assíncrona do mêtodo DR, que no caso específico do nosso problema exemplo tem a forma abaixo:

COROLÁRIO 4.1 *A versão bloco assíncrona do método Desacoplado Rápido, converge para a única solução x^* em D, assumindo verdadeiras as Hipóteses 2.1, 2.2, 2.6 e 2.7 (válidas para $x \in \mathbf{C}$), se*

$$\rho(H_{DR}) < 1$$

onde,

$$H_{DR} = (h_{ij}), \qquad h_{ij} = \begin{cases} \alpha_i & se\ j = i \\ (1+\alpha_i)\ \dfrac{\|Y_{ii}^{-1}Y_{ij}\|_\infty}{1 - l_{F_i}\|Y_{ii}^{-1}\|_\infty} & se\ j \neq i \end{cases} \tag{4.25}$$

onde l_{F_i} é a mesma constante de Lipschitz de $I_i(x_i)$ dada em (4.15), e os valores de $\alpha_i = C_{DRi}$ podem ser calculados para cada bloco a partir do Lema 4.3.

Nota 4.2 O método da solução por componente, discutido na subseção 2.2.3, poderia ser utilizado fazendo com que cada processador i aplique sucessivamente o operador G_{DRi}, até alcançar a convergência local, e somente então realize o intercambio de informações correspondente com os outros processadores; essa seria uma possibilidade de implementação a ser estudada.

Para finalizar esta análise do método Desacoplado Rápido, enfatizamos a importância dos parâmetros ϵ e θ_{max} que delimitam a região de interesse D, de acordo

com (4.7), para determinar a convergência do método. Sendo assim, em alguns casos mais críticos, pode ser estimada uma condição inicial $x(0) = V(0)\, e^{j\theta(0)}$ melhor que o simples 'flat start' ($x(0) = 1$) e assim o domínio D dado por (4.7) pode ser redefinido (Wu, 1977) como o conjunto de tensões $x = V\, e^{j\theta}$ que satisfazem:

$$\forall k \in \{1, \ldots, n\} \qquad \begin{cases} |\, \theta_k - \theta_k(0)\, | & \leq & \theta_{max} \\ |\, V_k - V_k(0)\, | & \leq & \epsilon \end{cases} \tag{4.26}$$

4.1.4 Team Algorithms

Considerando todas as vantagens dos *Team Algorithms* no contexto bloco assíncrono, analisadas no Capítulo 3, postulamos a seguir a utilização dos mesmos na resolução do problema do fluxo de potência elétrica, em sistemas computacionais de memória distribuída.

Em geral, dado um problema com sua correspondente partição em blocos, a escolha do algoritmo apropriado a cada subproblema deve ser realizada atendendo às características particulares do subproblema considerado. Sendo assim, a conformação do Team Algorithm resultante da escolha do algoritmo apropriado a cada subproblema, depende do problema específico que está sendo considerado. Conseqüentemente, consideramos um exemplo específico de forma a ilustrar a conformação para posterior análise de um Team Algorithm.

Exemplo 4.1 Seja um sistema elétrico formado por m subsistemas tais que:

- todas as barras PV do sistema formam parte do subsistema 1;
- os subsistemas 2 a m caracterizam-se por ter ramos com valores elevados da relação R/X, assim como nós adjacentes com elevadas diferenças de fase.

Considerando que temos disponíveis os métodos DR e BJ, analisados nas subseções anteriores, para resolver o problema do Exemplo 4.1, torna-se evidente das características do problema proposto, a seguinte escolha de algoritmos para conformar um TA adequado à solução do problema considerado:

SUBSISTEMA 1:	método Desacoplado Rápido (G_1);
SUBSISTEMAS 2 a m:	método bloco Jacobi ($G_i, \quad i = 2, \ldots, m$).

Se os subsistemas estão bem desacoplados, podemos pensar em implementar um bloco TA sem 'overlapping', utilizando m processadores. Para isto, denotamos como G_{DR1} ao operador G_1 correspondente ao método DR aplicado ao subvetor 1, enquanto mantemos a notação G_i para o operador do método BJ aplicado ao subvetor i, $(i = 2 \cdots m)$. Desta forma a matriz de atribuição associada a este bloco TA é dada pela mesma matriz identidade (veja característica 4 da matriz de atribuição, na

subseção 3.2.1). O bloco TA assim constituído pode ser representado, num contexto assíncrono, por:

PROCESSADOR 1:

$$x_1(k+1) = G_{DR1}\big(x^1(k)\big) \qquad \text{onde } G_{DR1} \text{ foi definido na subseção 4.1.3;}$$

PROCESSADOR i, $\quad i = 2, \ldots, m$:

$$x_i(k+1) = G_i\big(x^i(k)\big) \qquad \text{onde } G_i \text{ é definido conforme (4.14).} \tag{4.27}$$

Do exposto na seção 2.3, a matriz de comparação $H = (h_{ij})$ do bloco TA dado por (4.27) pode ser facilmente derivada a partir das matrices de comparação dos métodos sendo combinados. Assim, podemos escrever:

$$H = \big(h_{ij}\big), \qquad \text{onde} \quad h_{ij} = \begin{cases} h_{1j} & \text{dado por (4.25)} \quad \text{se } i = 1 \quad \text{(método DR)} \\ h_{ij} & \text{dado por (4.15)} \quad \text{se } i > 1 \quad \text{(método BJ)} \end{cases} \tag{4.28}$$

Uma vez conhecida a matriz de comparação, a convergência deste bloco TA pode ser assegurada da mesma forma que na seção 2.3, utilizando o Teorema 2.1, adequadamente re-escrito para considerar agora x como pertencente ao conjunto dos números complexos $\mathbf{C}$ (ao invés do conjunto $\mathbb{R}$).

Contudo, pode acontecer que o bloco TA dado por (4.27) não resolva adequadamente o problema considerado, e a utilização de 'overlapping' parcial, com o correspondente Administrador, tenha que ser considerada. O seguinte exemplo enfatiza este caso:

Exemplo 4.2 Seja um sistema elétrico formado por m subsistemas com as mesmas características do Exemplo 4.1, no qual o subsistema m é um 'bloco crítico' fortemente ligado aos subsistemas 1 e $(m-1)$.

Devido à semelhança com o Exemplo 4.1, podemos escolher os mesmos algoritmos para resolver cada subproblema; porém, o bloco m deve ser resolvido no mesmo processador que os blocos 1 e $(m-1)$ ao mesmo tempo, para não 'cortar' as ligações fortes, o que somente é possível utilizando 'overlapping' parcial. A Figura 4.1 ilustra um sistema elétrico destas características, com $m = 5$ subsistemas interligados. Ligações fortes entre subsistemas são indicadas com linhas duplas; enquanto a parte do sistema elétrico , tratada por cada processador $(p = 4)$ é indicada por uma curva tracejada. Note a utilização de 'overlapping' parcial entre os blocos 1 e 4, pois ambos resolvem o subsistema 5.

Conseqüentemente, se propõe a utilização de um TA Generalizado, com 'overlapping' parcial, que utilize $p = m - 1$ processadores, já que o subsistema m será replicado e resolvido em conjunto com os subsistemas 1 e $(m-1)$ respectivamente.

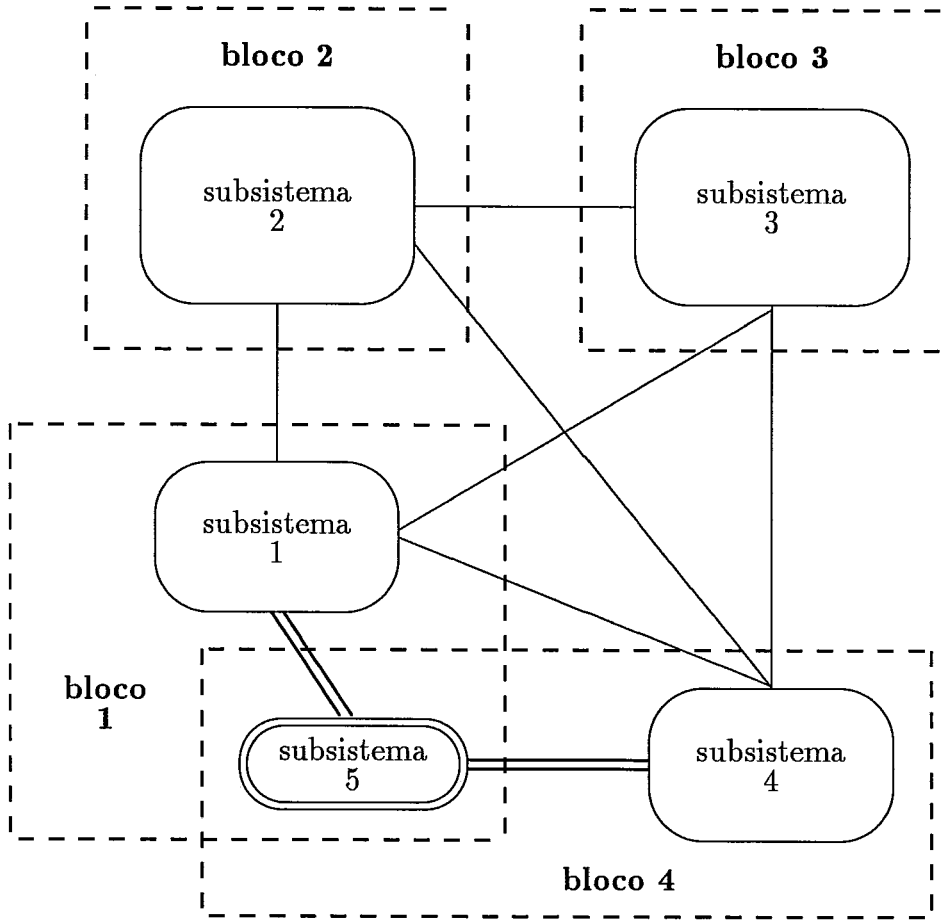

Figura 4.1: Sistema elétrico do Exemplo 4.2 composto por 5 subsistemas elétricos interligados (linhas duplas indicam ligações fortes). Note a utilização de 'overlapping' parcial, pois tanto o processador 1 quanto o processador 4 resolvem o subsistema 5.

O TA Generalizado correspondente tem associado uma matriz de atribuição de dimensão $m \times p = m \times (m-1)$ dada por:

$$\Psi = \left(\psi_{ij}\right) = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ 1 & 0 & \cdots & 1 \end{bmatrix} \tag{4.29}$$

$$\text{i.e.,} \qquad \psi_{ij} = \begin{cases} 1 & \text{para } i = j \\ 0 & \text{para } i < m, \ j \neq i \\ 1 & \text{para } i = m, \ j = 1 \\ 0 & \text{para } i = m, \ 1 < j < p \\ 1 & \text{para } i = m, \ j = p \end{cases}$$

Claramente,num contexto assíncrono, existem várias versões do TA Generalizado que tem associado a matriz de atribuição (4.29), conforme discussão do Capítulo 3. Como exemplo, apresentamos a seguir um TA Generalizado com Administrador implícito e centralizado, sem processador exclusivo; no qual, a função do Administrador é incluida no processador 1, de forma similar ao caso 4 do Exemplo 3.3. Esta versão do TA Generalizado, chamada na Tabela 3.2 de *Versão E*, pode ser representada, para o caso particular do Exemplo 4.1, pelas equações que seguem:

PROCESSADOR 1:

$$\begin{bmatrix} x_1 \\ x_m \end{bmatrix} \leftarrow G_{DR1}\left(\hat{x}^1\right) \qquad \text{com} \quad \hat{x}^1 = [\, x_1^T, \ldots, x_p^T, (w_1 x_m + w_2 \chi_{mp})^T \,]^T$$

PROCESSADOR i, $\quad i = 2, \ldots, (p-1)$

$$x_i \leftarrow \widehat{G}_i\left(x^i\right) \qquad \text{com} \quad x^i = [\, x_1^T, \ldots, x_m^T \,]^T \tag{4.30}$$

PROCESSADOR p

$$\begin{bmatrix} x_p \\ \chi_{mp} \end{bmatrix} \leftarrow \begin{bmatrix} \widehat{G}_p\left(x^p\right) \\ G_{mp}\left(x^p\right) \end{bmatrix} \qquad \text{com} \quad x^p = [\, x_1^T, \ldots, x_m^T \,]^T$$

onde $G_{DR1}(\cdot)$ foi definido na subseção 4.1.3 e $\widehat{G}_i(\cdot)$ representa o método BJ utilizado no processador i e dado por (4.14). A derivação de uma condição suficiente de convergência para esta, e as outras versões estudadas no Capítulo 3, do TA Generalizado que tem associado a matriz de atribuição (4.29), fica a cargo do leitor interessado. Para isto, é suficiente aplicar o Teorema 2.1 ao algoritmo considerado, seguindo a mesma metodologia da seção 3.3.

Nota 4.3 Os exemplos 4.1 e 4.2 foram implementados no sistema distribuído que se descreve a seguir, para um sistema elétrico de 616 barras. Os detalhes do problema e da implementação são descritos nas seções que seguem; enquanto, os resultados experimentais são apresentados e discutidos no Capítulo 5.

4.2 Ambiente Computacional Utilizado

Para a implementação dos programas de fluxo de carga que serão descritos nas próximas seções, foi utilizado o supercomputador iPSC/860 da INTEL, disponível no Laboratório de Computação Paralela da Universidade Federal de Rio de Janeiro. O mesmo dispõe de um hipercubo de 8 nós (que pode ser expandido até 128 nós), um computador pessoal tipo 386 utilizado como hospedeiro local e dois hospedeiros remotos, sendo um deles uma 'workstation' (Sun 3) ao qual estão conectados mais dois computadores pessoais, como terminais.

Os trabalhos foram realizados a partir da 'workstation' usada como hospedeiro remoto, unida via Ethernet ao hospedeiro local, conhecido como SRM (System Resource Manager). Os programas paralelos implementados consistem de um programa de controle de nós, executado no SRM (ou simplesmente hospedeiro) e um programa para cada nó utilizado. [1]

Cada um dos 8 nós do sistema consiste num microprocessador i860 com 8 Mbytes de memória, capaz de comunicar-se em forma direta com qualquer dos outros nós do sistema ou com o hospedeiro, utilizando um módulo DCM (Direct-Connect Module) com 8 canais. Cada microprocessador i860 tem uma arquitetura de 64 bits e utiliza um 'clock rate' de 40 MHz, obtendo um desempenho pico de 32 MIPS (para inteiros) ou 80 MFLOPS em punto flutuante de 32 bits, caindo a 60 MFLOPS com ponto flutuante de 64 bits (Heath et al., 1991). Maiores detalhes podem ser consultados nos manuais do sistema ou nos trabalhos que publicam diversos testes e aplicações já realizadas (como em Dunigan, 1991; ou Heath et al., 1991).

O ambiente de software do sistema baseia-se num sistema operacional tipo Unix residente no hospedeiro. Este sistema operacional é uma versão estendida do Unix System V, que suporta TCP/IP e NFS (Network File System) via Ethernet. Por sua parte, os nós utilizam um sistema operacional (kernel) relativamente simples, conhecido como NX, com facilidades para executar um único processo por nó.

A utilização de um ambiente Unix no SRM, permite ao hospedeiro executar concorrentemente vários processos, que podem pertencer a usuarios diferentes. Cada um destes processos pode utilizar (ou não), em forma simultánea, um grupo dos 8 nós disponíveis. Como o sistema é multi-usuario, não foi possível assegurar as mesmas condições de utilização do sistema em execuções sucessivas de um mesmo programa. A conseqüencia disto, foi o aumento do tempo de resposta na entrada/saída de dados, assim como variações significativas nos atrasos de comunicação entre processadores, o que afeta especialmente o desempenho das implementações assíncronas.

Como os programas são executados fundamentalmente nos nós, é importante ressaltar as capacidades de Entrada/Saída de cada nó. Neste sentido é de grande utilidade a capacidade de todos os nós de escrever na tela, accessar o disco do SRM ou

[1] O número de nós a ser utilizado deve ser uma potência de 2, mesmo que alguns não realizem trabalho útil ('dummy').

utilizar o CFS (Concurrent File System) que apresenta um excelente desempenho.

A linguagem utilizada em nossa implementação foi o Fortran iPSC/860, uma extensão do Fortran 77, especialmente desenvolvida para permitir o máximo aproveitamento do sistema, com instruções específicas para a programação paralela. A escolha do Fortran foi motivada pela disponibilidade de varios programas seqüencias para a solução do problema do fluxo de potência elétrica que foram reutilizados no contexto paralelo. Para obter um máximo de reutilização dos programas disponíveis, manteve-se quase sem alteração os módulos seqüenciais de entrada e saída de dados.

Para o aperfeiçoamento dos programas (debugging), foi utilizado o IPD (Interactive Parallel Debuger), que porém não resultou muito útil para a análise dos programas assíncronos, pois o 'debugger' não permite reproducir totalmente o ambiente assíncrono no qual os programas são executados.

Para finalizar esta breve exposição do sistema computacional do iPSC/860, descrevemos o sistema de 'buffers' (memória intermediária) utilizado na comunicação entre os nós, para enfatizar as vantagens de uma implementação na qual um dado nó solicita uma mensagem ao sistema de 'buffers' mesmo antes que ela chegue proveniente de algum outro processador. Efetivamente, cada nó dispõe de dois buffers:

Buffer do Sistema: no qual são depositadas as mensagens que chegam ao nó mas que ainda não foram requisitadas;

Buffer da Aplicação: no qual se depositam as solicitações de mensagens (a serem recebidas 'a posteriori'), até que elas sejam atendidas;

sendo assim, uma mensagem que chega depois da solicitação, não é escrita no Buffer do Sistema e vai diretamente a atender à solicitação que estava no Buffer da Aplicação. Caso contrário, uma mensagem recém chegada é escrita no Buffer do Sistema até ser requisitado. Quando isto acontece, uma segunda escrita é necessária, desta vez no Buffer da Aplicação. Conseqüentemente, duas *escritas* da mensagem são necessarias, uma para cada Buffer. Enfatizamos aqui a capacidade do sistema de realizar comunicações assíncronas eficientes que permitem transmitir uma mensagem àqueles nós que ainda não solicitaram essa mensagem, ou solicitar por um tipo particular de mensagem, sem ficar bloqueado caso a mensagem não tenha chegado, o que favorece as implementações assíncronas.

Finalmente, ressaltamos a grande dificuldade encontrada para controlar o uso dos buffers no contexto assíncrono devido à ausência da forma de se conhecer o estado dos mesmos. Efetivamente, se um nó começa a receber um número grande de mensagens sem liberar o buffer correspondente, o sistema de buffers fica cheio e deixa de receber novas mensagens sem comunicar ao usuário. O resultado é que o nó descarta, em muitos casos, a mensagem de terminação e com isto fica trabalhando *ininterruptamente*. A única solução por nós encontrada para recuperar estes nós 'presos', foi dar um novo 'boot' em todo o sistema.

4.3 Implementação de um bloco TA: Problema Exemplo 1

Nesta seção, discute-se um problema de fluxo de carga que será denominado *problema exemplo 1*. Este problema não pode ser resolvido em forma aceitável por nenhum dos algoritmos estudados neste capítulo, quando aplicados individualmente. Porém, quando um bloco TA é formado, escolhendo o algoritmo apropriado para cada bloco, o problema pode ser facilmente resolvido. O pseudocódigo da implementação realizada será também apresentado, deixando-se para o capítulo final a apresentação e análise dos resultados experimentais.

Como dispunhamos de um supercomputador de 8 nós ($p = 8$), o *problema exemplo 1* foi construido interconectando 8 subsistemas ($m = 8$) de 77 barras cada um ($n_i = 77, \forall i \in \{1, \ldots, 8\}$), de forma que o sistema completo tenha 616 barras, das quais 18 são barras PV.

O primeiro subsistema, que denominaremos *subsistema 1*, contem todas as barras PV e foi inspirado no problema tipo da IEEE para 57 barras que é facilmente resolvido pelo método Desacoplado Rápido. Porém, o método da matriz Y não consegue resolvê-lo devido à presença das barras PV.

Os outros 7 subsistemas foram inspirados em sistemas de plataformas submarinas para extração de petróleo, que não utilizam barras de tensão controlada e interconectam muitas de suas barras utilizando cabos submarinos que se caracterizam pelo elevado valor da relação R/X (resistência/reatância). Estes subsistemas tem também vários nós adjacentes com elevadas diferenças de fase. O resultado das características destes 7 subsistemas é que o método DR tem dificuldades para convergir corretamente enquanto o método BJ consegue convergir razoavelmente bem.

Os 8 subsistemas se interconectam ums com os outros utilizando ramos com elevada relação R/X (cabos submarinos). Sendo assim, o *problema exemplo 1* poderia representar um sistema real no qual o sistema elétrico de uma região é interconectado a 7 plataformas submarinas. O balanceamento de carga computacional do sistema não foi otimizado, pois o número de ramos internos a cada subsistema não é constante, variando também o número de ramos que interconectam os diferentes subsistemas. A Figura 4.2 representa o número de ramos entre um subsistema e outro, sendo que cada eixo representa o índice de um subsistema. Assim por exemplo, o elemento da fila 6, coluna 7 é 9; o que indica que existem 9 ramos ligando os subsistemas 6 e 7. O número total de ramos do sistema é de 945 e pode ser obtido somando todos os ramos que aparecem na Figura 4.2.

Nota 4.4 O Problema Exemplo 1 é um caso particular do Exemplo 4.1, analisado na subseção 4.1.4.

8 ↑	105							
7	10	92						
6	6	9	102					
5	6	4	7	94				
4	6	4	4	8	99			
3	8	4	4	4	5	92		
2	7	4	4	4	4	7	103	
1	5	4	5	4	4	5	5	107
	8	7	6	5	4	3	2	1 →

Figura 4.2: Número de ramos entre subsistemas do *problema exemplo 1*.

Das características do problema proposto, não é de estranhar que o método da matriz Y não consegue resolver o problema todo nem mesmo num contexto seqüencial, devido à presença de barras PV. Todavia, o método Desacoplado Rápido seqüencial consegue resolver o problema proposto quando a relação R/X não é muito grande (mas falha em convergir, caso contrário). Sendo assim, o bloco TA representado por (4.27) se mostra muito mais eficiente, pois resolve cada subproblema com o método mais adequado a esse subproblema. Por exemplo, o método BJ não é utilizado com o subsistema que tem barras PV, e portanto converge bem em outros subproblemas, onde o método DR falha. Efetivamente, foi possível verificar experimentalmente que o bloco TA representado por (4.27) consegue resolver o problema proposto, mesmo quando o método Desacoplado Rápido seqüencial não converge.

Para o caso particular do *Problema Exemplo 1*, utilizando 8 processadores, o bloco TA dado por (4.27) pode ser representado por:

PROCESSADOR 1:

$$x_1 \leftarrow G_{DR1}(x) \qquad \text{onde } G_{DR1} \text{ foi definido na subseção 4.1.3;}$$

PROCESSADOR i, $\quad i = 2, \ldots, 8$:

$$x_i \leftarrow G_i(x) \qquad \text{onde } G_i \text{ está definido segum (4.14).} \tag{4.31}$$

A condição suficiente de convergência para o bloco TA representado por (4.31), foi discutida na subseção 4.1.4. Contudo, ela se mostra muito conservativa para o problema em questão, como é o caso da maioria das condições suficientes derivadas num contexto assíncrono, ainda mais se considaramos que mesmo a constante de Wu, derivada num contexto seqüencial, já é muito conservativa. Contudo, estudos de convergência tem um interesse teórico, e no caso particular do bloco TA considerado, a matriz de comparação H dada por (4.28) tem no mínimo a utilidade de permitir analisar em que medida, cada parâmetro do sistema e do algoritmo afetam a convergência.

Para finalizar esta seção, apresentamos o pseudo-código da implementação paralela assíncrona realizada no sistema computacional distribuído descrito na seção ante-

rior (o iPSC/860 da INTEL). O pseudo-código que se apresenta nas Figuras 4.3 e 4.4, utiliza uma pseudo-linguagem tipo Fortran estruturado, por compatibilidade com o código fonte. Dois parâmetros de comunicação foram incluidos na implementação: *rep* e *nrec*. Isto foi necessário para evitar o problema de ultrapassar a capacidade do sistema de buffers, explicado na seção anterior. O parâmetro *rep* permite aumentar a granularidade do problema, aumentando o tempo de cálculo antes de qualquer tentativa de comunicação (recepção / transmissão). Por sua vez, o parâmetro *nrec* determina o número de recepções a cada transmissão, lembrando que tanto as recepções como as transmissões se realizam somente se possível, não havendo nenhum prejuizo importante se não forem efetuadas.

Para o caso específico do nó 1, utilizamos um parâmetro *maxit* que determina o número máximo de iterações que serão tentadas no processador 1 antes de interromper o processamento. Desta forma, assegura-se que o sistema termine mesmo que o algoritmo divirja. Também pode-se notar no processador 1 uma variável lógica *fim.local* que toma o valor TRUE se uma solução local foi alcançada pelo método DR, ou FALSE em caso contrário. Lembramos que o custo computacional para estebelecer o valor desta variável *fim.local* é mínimo quando o operador G_{DR1} é aplicado.

A medição dos tempos de cada execução é explicitado no pseudo-código para enfatizar que somente é considerado o tempo de cálculo, para não ter que considerar o tempo de entrada/saída de dados, altamente dependente do estado de utilização do sistema. É importante notar que a medição é realizada no processador 1 por ser este o processador que determina quando uma solução é encontrada. Enfatizamos a utilização de uma barreira de sincronização entre todos os nós, antes de iniciar a contagem de tempo. Isto foi necessário para evitar a computação do tempo decorrido, mesmo antes que os programas correspondentes estejam carregados em todos os nós.

4.4 Implementação de um TA Generalizado: Problema Exemplo 2

Nesta seção apresenta-se uma variante do problema de fluxo de carga da seção anterior, que chamamos *problama exemplo 2*. Este problema tem características similares ao Problema Exemplo 1 da seção anterior, que como já fora discutido, não permite obter uma boa convergência com nenhum dos métodos estudados, quando aplicados individualmente. Porém, resulta eficiente a utilização de um *Team Algorithm* que utiliza o método adequado a cada bloco. Contudo, o *problema exemplo 2* não é, em geral, de solução simples por dois motivos: não existe uma boa partição em subproblemas menores, e a condição inicial de 'flat start' não é boa o suficiente para evitar uma *tendência* do sistema, a divergir nas primeiras iterações, para algumas barras.

A solução encontrada para resolver o Problema Exemplo 2, foi a utilização de um TA Generalizado, que utiliza uma partição adequada com 'overlapping' parcial; enquanto que a divergência prematura de algumas tensões, é resolvida com a utilização

Leitura de dados.
Transmissão de dados aos outros nós.
continua ← TRUE
iter ← 0
Sincronização entre os nós.
Inicio da contagem de tempo.
Prefatoração de B' e B''.
DO WHILE *continua*
    DO $n = 1$, *nrec*
        DO $i = 2$, p
            Recepção de todo x_i que tenha chegado.
        ENDDO
        DO $i = 1$, *rep*
            iter ← *iter* + 1
            IF (*iter* > *maxit*) **Terminação com erro.**
            $x_1 \leftarrow G_{DR1}(x)$ (método DR)
            IF (solução local) THEN *fim.local* ← TRUE
                ELSE *fim.local* ← FALSE.
            ENDIF
        ENDDO
        IF (*fim.local*) THEN
            Verificação da solução global.
            IF (solução global) THEN
                Terminação da contagem de tempo.
                Saída dos resultados.
                Terminação geral.
            ENDIF
        ENDIF
    ENDDO
    IF (terminou transmissão anterior) Transmitir x_1 a todos.
ENDDO

Figura 4.3: Pseudo-código do bloco TA para o processador 1.

```
Recepção dos dados.
continua ← TRUE
Sincronização entre os nós.
Prefatoração de Y_kk.
DO WHILE continua
    DO n = 1, nrec
        DO i = 1, p
            IF (i = k) THEN Receber mensagem de terminação.
                       ELSE Recepção de todo x_i que tenha chegado.
        ENDDO
        DO i = 1, rep
            x_k ← G_k(x)        (método BJ)
        ENDDO
    ENDDO
    IF ( terminou transmissão anterior ) THEN Transmitir x_k a todos.
ENDDO
```

Figura 4.4: `Pseudo-código para um processador` k; $\quad k = 2 \cdots 8$.

de um Administrador que limita as atualizações de voltagem a um percentual do valor calculado, no caso em que este ficaria fora do domínio desejado D, se utilizado diretamente. O TA assim constituído, consegue resolver problemas que nem o bloco TA, nem os métodos sendo combinados, conseguem resolver.

O problema do fluxo de potência elétrica que denominamos *problema exemplo 2* foi gerado a partir do problema exemplo 1 da seção anterior, modificando os parâmetros da rede. Contudo, manteve-se a dimensão de 616 barras para o problema, das quais 18 continuam sendo barras PV. Também foi mantida a estrutura do problema em um *subsistema 1* com todas as barras PV e com os demais subsistemas caracterizados pelo elevado valor da relação R/X dos ramos e das elevadas diferenças de fases θ_{ki} entre algumas barras adjacentes. O número de subsistemas com estas características passa neste caso a ser três. A distribuição de barras por subsistema é a seguinte:

$$n_1 = 70; \qquad n_2 = 188; \qquad n_3 = 188 \qquad \text{e} \qquad n_4 = 170.$$

O sistema tem 945 ramos distribuídos entre os quatro subsistemas da forma ilustrada na Figura 4.5. O acoplamento entre subsistemas é mais forte que no problema exemplo 1, especialmente entre os subsistemas 1 e 4. O resultado deste acoplamiento (elevado valor de $\|Y_{14}\|$ e $\|Y_{41}\|$) gera uma grande dificuldade para a convergência do sistema assim particionado. Isto pode ser explicado a partir da matriz de comparação $H = (h_{ij})$, uma vez que:

$$h_{14} \propto \|Y_{11}^{-1}\, Y_{14}\| \qquad \text{e} \qquad h_{41} \propto \|Y_{44}^{-1}\, Y_{41}\|;$$

4	207			
3	25	203		
2	22	20	196	
1	20	17	20	215
	4	3	2	1

Figura 4.5: **Número de ramos entre subsistemas do** *problema exemplo 2.*

são valores elevados e sendo assim, nenhum dos algoritmos sem 'overlapping', apresentados neste capítulo, consegue uma boa convergência para esta partição.

Relembrando o Exemplo 3.3, no qual tinhamos um problema similar de forte acoplamento entre blocos, podemos optar pela mesma solução: *'overlapping' parcial.* A idéia é simplesmente subdividir o *subsistema 1* original em dois outros subsistemas: um com as 10 barras *críticas*, a serem replicadas por estar fortemente ligadas ao subsistema 4, que denominaremos *subsistema 5*, e outro com as 60 barras restantes, que continuaremos chamando de *subsistema 1.* Sendo assim, o *subsistema 5* pode ser resolvido tanto no processador 1, como no processador 4.

O sistema assim constituído, é ilustrado na Figura 4.1, e pode ser resolvido por o TA Generalizado do Exemplo 4.2, dado que a formulação do problena exemplo 2 desta seção é um caso particular do Exemplo 4.2.

Sendo assim, consideramos o TA Generalizado que utiliza o método DR para resolver, no processador 1, os subsistemas 1 e 5; enquanto os outros processadores implementam o método BJ para resolver seus respectivos subsistemas, enfatizando-se que o processador 4 resolve os subsistemas 4 e 5. O TA assim definido tem uma matriz de atribuição Ψ dada por (4.29), que repetimos a seguir para o caso particular do problema exemplo 2 $(m = 5,\ p = 4)$:

$$\Psi = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \tag{4.32}$$

$$\text{onde} \quad \begin{cases} G_1 : & \text{método DR} \\ G_2 \cdots G_4 : & \text{método BJ} \end{cases} \qquad \text{e} \quad \begin{cases} n_1 = 60 \\ n_2 = 188 \\ n_3 = 188 \\ n_4 = 170 \\ n_5 = 10 \end{cases}$$

Como existem várias possíveis versões para um TA Generalizado que tem associado a matriz de atribuição (4.32), escolhemos 3 destas de forma a poder compará-las:

Versão A: com Administrador centralizado em processador dedicado, descrito pelas equações (3.58) e (3.60) e convergência assegurada pelo Corolário 3.4. Neste caso, o Administrador é implementado num quinto processador.

Versão D: com Administrador implícito distribuído e replicado, descrito pelas equações (3.69) e (3.71) e convergência assegurada pelo Corolário 3.6. Neste caso, o Administrador é replicado em cada um dos processadores. Porém, na implementação por nós realizada, os processadores $j = 2$ e 3 não implementam um Administrador, pois eles não calculam χ_{5j} e podem simplesmente receber o valor do subvetor 5 (x_5) utilizado no processador 1 para verificar se a solução global foi alcançada.

Versão E: com Administrador centralizado implícito (no processador 1), descrito na subseção 4.1.4 por (4.30). Neste caso, a função do Administrador é implementada no mesmo processador 1, o que facilita a verificação de que as novas tensões estejam no domínio desejado D, dado que o método DR utiliza os valores explícitos de magnitude e fase das tensões.

Verifica-se experimentalmente que as três implementações do Administrador acima citadas, tem um desempenho similar para o problema exemplo 2, como mostram os resultados que serão apresentados no próximo capítulo. Portanto, se precisamos escolher alguma das implementações, poderiamos nos inclinar pela versão E que utiliza um Administrador centralizado implícito no processador 1, por não precisar de um quinto processador e por a simplicidade que implica não ter que replicar o Administrador.

Para exemplificar esta escolha para o problema exemplo 2, re-escrevemos a seguir a versão E do TA Generalizado que tem associado a matriz de atribuição (4.32), conforme (4.30):

PROCESSADOR 1:

$$\begin{bmatrix} x_1 \\ x_5 \end{bmatrix} \leftarrow G_{DR1}(\hat{x}^1) \qquad \text{onde} \quad \hat{x}^1 = [x_1^T, \ldots, x_4^T, (w_1 x_5 + w_2 \chi_{54})^T]^T$$

PROCESSADOR i, $\quad i = 2, 3$

$$x_i \leftarrow \widehat{G}_i(x^i) \qquad \text{onde} \quad x^i = [x_1^T, \ldots, x_5^T]^T \tag{4.33}$$

PROCESSADOR 4

$$\begin{bmatrix} x_4 \\ \chi_{54} \end{bmatrix} \leftarrow G_4(x^4) = \begin{bmatrix} \widehat{G}_4(x^4) \\ G_{54}(x^4) \end{bmatrix} \qquad \text{onde} \quad x^4 = [x_1^T, \ldots, x_5^T]^T$$

onde $G_{DR1}(\cdot)$ foi definido na subseção 4.1.3 e $G_i(\cdot)$ representa o método BJ utilizado no processador i e dado por (4.14).

O pseudo-código da implementação assíncrona do TA generalizado dado por (4.30) é similar ao dado nas Figuras 4.3 e 4.4 para o bloco TA. Mesmo assim, apresenta-se na Figura 4.6 o pseudo-código para o processador 1, de forma a enfatizar as diferenças entre as duas implementações. O pseudo-código correspondente aos demais processadores não é apresentado, pois praticamente não difere do apresentado na Figura 4.4.

Por brevidade também não são apresentados os detalhes das outras versões assíncronas do TA Generalizado que resolve o *problema exemplo2*. Porém, foram implementados no sistema computacional descrito na seção 4.2, as três versões discutidas nesta seção, um bloco TA, assim como versões síncronas de cada uma destas versões. Vários métodos seqüencias foram também implementados, de forma a medir o 'speedup' obtido; dentre estos métodos destacamos o Desacoplado Rápido, o método da matriz Y e até uma versão seqüencial do Team Algorithm considerado (com e sem 'overlapping'). Os resultados experimentais destas implementações são apresentados a seguir, no Capítulo 5.

Para encerrar o estudo de convergência realizado neste capítulo, considerando o problema do fluxo de potência elétrica, devemos mencionar que as condições suficientes de convergência derivadas a partir do Teorema 2.1 para cada um dos TA Generalizados que foram implementados, se mostram muito conservativas nos exemplos experimentados, como é o caso na maioria dos estudos de convergência num contexto assíncrono. Ainda mais, nem mesmo no contexto seqüencial foi possível assegurar a convergência dos problemas exemplos 1 e 2 utilizando a condição suficiente de convergência dada pelo Lema 4.1.

Como exemplo, a constante de Wu para o *problema exemplo 1* (o mais simples, apresentado na Seção 4.3) é dada por:

$$C_w \approx 310 \gg 1.$$

Conseqüentemente, não é de estranhar que quando consideramos o TA que utiliza o método DR no bloco 1 se tenha:

$$\rho(H_{TA}) \geq h_{11} = C_{DR_1} \approx 5.1 > 1$$

i.e., o TA que utiliza o método DR no processador 1 tem um raio espectral maior do que 1 e portanto, o Teorema 2.1 não assegura a convergência do TA implementado. Porém, enfatizamos o valor teórico dos estudos de convergência.

Sumário do Capítulo 4

Neste capítulo foi apresentado o problema do *fluxo de potência elétrica* como problema exemplo de forma a verificar experimentalmente, em sistemas de porte considerável, as características dos Team Algorithms estudadas nos capítulos anteriores.

Leitura de dados.
Transmissão de dados aos outros nós.
continua $\leftarrow$ TRUE
iter $\leftarrow$ 0
Sincronização entre os nós.
Inicio da contagem de tempo.
Prefatoração de B' e B''.
DO WHILE *continua*
DO $n = 1,\ nrec$
DO $i = 2,\ p$
Recepção de todo x_i e χ_{ji} que tenha chegado.
ENDDO
DO $i = 1,\ rep$
iter $\leftarrow$ *iter* + 1
IF (*iter* > *maxit*) **Terminação com erro.**
Escolha dos pesos w_1 e w_2 (se verifica que $x_5 \in D$).
$x_5 \leftarrow w_1 x_5 + w_2 \chi_{54}$
$\begin{bmatrix} x_1 \\ x_5 \end{bmatrix} \leftarrow G_{DR1}(x)$ (método DR)
IF (solução local) THEN *fim.local* $\leftarrow$ TRUE
ELSE *fim.local* $\leftarrow$ FALSE.
ENDIF
ENDDO
IF (*fim.local*) THEN
Verificação da solução global.
IF (solução global) THEN
Terminação da contagem de tempo.
Saída dos resultados.
Terminação geral.
ENDIF
ENDIF
ENDDO
IF (terminou transmissão anterior) Transmitir $\begin{bmatrix} x_1 \\ x_5 \end{bmatrix}$ a todos.
ENDDO

Figura 4.6: `Pseudo-código da versão E de um TA para o processador 1.`

Após uma introdução histórica do problema do fluxo de potência elétrica, o mesmo foi formalizado matematicamente. Métodos tradicionais para resolver o problema, como a versão bloco Jacobi do método da matriz Y e o método Desacoplado Rápido, foram estudados, primeiramente no contexto seqüencial em que são normalmente utilizados, para logo se propor uma versão bloco assíncrona dos mesmos, que viabilize a solução deste tipo de problemas nos sistemas computacionais de memória distribuída, mais baratos que os 'mainframes' hoje utilizados para resolver sistemas de grande porte. Estudos de convergência foram incluidos para cada proposta.

A existência de problemas nos quais nenhum dos algoritmos considerados consegue resolver isoladamente de forma satisfatória, foi levantado. Por outro lado, foram discutidos exemplos nos quais, um *Team Algorithm* que combina estes métodos, consegue resolve-los de forma adequada. Várias versões de Team Algorithms foram analisadas para estes exemplos, derivando-se uma condição suficiente de convergência para alguns casos e enfatizando o método geral para os outros casos não analisados em detalhes. Foi verificado que as condições suficientes de convergência derivadas do Teorema 2.1 se mostram muito conservativas para os problemas práticos estudados; porém, foi enfatizado o valor teórico destes resultados de modo a conhecer em que forma cada parâmetro afeta a convergência.

Com o objetivo de verificar experimentalmente as propriedades estudadas, foram realizadas implementações dos algoritmos num sistema computacional distribuído, descrito na Seção 4.2. Dois problemas exemplos foram assim implementados: o *problema exemplo 1* que pode ser resolvido utilizando um bloco TA sem 'overlapping', e o *problema exemplo 2* que somente pode ser resolvido utilizando um TA Generalizado com 'overlapping'. Os detalhes das implementações realizadas para resolver estes problemas exemplos foram incluidos, deixando-se para o próximo capítulo a apresentação e discussão dos resultados experimentais.

Capítulo 5

Resultados Experimentais

Neste capítulo apresentamos os principais resultados experimentais obtidos no hipercubo iPSC/860, descrito na Seção 4.2, para diversas implementações realizadas com o objetivo de resolver os *Problemas Exemplos 1 e 2*, de fluxo de potência elétrica, apresentados nas seções 4.3 e 4.4 respectivamente. Discussões dos resultados experimentais são incluidas, de forma a enfatizar as características mais importantes das implementações assíncronas e, especialmente, os *Team Algorithms Generalizados.*

Para facilitar a apresentação e comparação dos resultados experimentais, são primeiramente definidos na Seção 5.1, os parâmetros de comparação a serem utilizados no contexto distribuído deste capítulo, ou seja: o 'speedup', a aceleração devida ao assincronismo, as acelerações relativa e seqüencial de um TA, assim como as eficiências correspondentes.

Na Seção 5.2 são apresentados os resultados experimentais do bloco TA, sem 'overlapping', cujo pseudo-código, ilustrado nas figuras 4.3 e 4.4, foi explicado na Seção 4.3. O referido bloco TA é utilizado para resolver o problema do fluxo de potência elétrica, dado pelo Problema Exemplo 1. Resultados são apresentados para vários métodos de solução do problema considerado, ou seja: o método da matriz Y (versão bloco Jacobi), o método Desacoplado Rápido e o bloco TA, num contexto seqüencial; assim como versões paralelas síncronas e assíncronas do bloco TA, para 2, 4 e 8 processadores. A influência da condição inicial e da tolerância na resolução do problema é também analisada. Os principais resultados são discutidos de forma a enfatizar as vantagens dos bloco TA implementados num contexto assíncrono.

A resolução do Problema Exemplo 2 é analisada na Seção 5.3. É verificada experimentalmente a impossibilidade de se resolver satisfatoriamente este problema, utilizando qualquer dos métodos acima citados, sem usar 'overlapping' parcial. Sendo assim, um TA Generalizado utilizando 'overlapping' parcial é usado para resolver o problema exemplo 2, sem maiores dificuldades. São apresentados resultados experimentais correspondentes a versões seqüenciais bem como paralelas síncronas e assíncronas do referido TA Generalizado, enfatizando-se as principais características dos algoritmos que utilizam 'overlapping' parcial.

Várias versões assíncronas do TA Generalizado utilizando 'overlapping' parcial são comparadas, verificando-se experimentalmente um melhor desempenho, em termos de 'speedup', da versão E do TA com Administrador implícito e centralizado no processador 1 (sem processador dedicado), em relação à maioria dos casos experimentados (não todos). Finalmente, é apresentado um *Mapa de Iterações* típico, de forma a analisar a sensibilidade das implementações em relação à escolha dos pesos feita pelo Administrador.

5.1 Figuras de comparação no contexto dos TA

Para facilitar a apresentação dos resultados experimentais correspondentes aos TA Generalizados, assim como a comparação entre as diferentes versões ou com outros possíveis métodos de solução, definimos a seguir alguns parâmetros de comparação usados no contexto deste trabalho.

Uma das figuras de mérito mais utilizada para comparar implementações paralelas e seqüenciais é a *aceleração* (ou 'speedup' S_p) que pode ser definida de diferentes maneiras. Neste trabalho utilizamos a definição de Stone (1987):

DEFINIÇÃO 5.1 (ACELERAÇÃO OU 'SPEEDUP': S_p)
A aceleração ou 'speedup' S_p é dada pela relação entre o menor tempo seqüencial dentre todos os métodos existentes, e o tempo de execução do algoritmo considerado num sistema distribuído.

$$S_p \stackrel{\text{def}}{=} \frac{\textit{melhor tempo seqüencial dentre todos os métodos existentes}}{\textit{tempo da execução paralela considerada}}$$

A definição 5.1 se mostra ambígua para fins práticos (Stone, 1987), pois não é possível implementar todos os métodos seqüencias que existem para conhecer 'o melhor tempo seqüencial'. Contudo, será utilizada no sentido de considerar o melhor tempo dentre todas as implementações por nós realizadas no contexto seqüencial, i.e. o menor tempo dentre os métodos da matriz Y, o método Desacoplado Rápido e diferentes versões do TA Generalizado. Porém, enfatizamos que outros métodos, como o de Newton-Raphson e o método da matriz Z, não foram considerados nos cálculos de S_p nas seções que seguem.

Nota 5.1 Lembramos que os 'tempos' considerados neste trabalho não incluem a entrada e saída de dados, como fora indicado nos pseudo-códigos do Capítulo 4.

O conceito de aceleração ou 'speedup' é utilizado para representar o ganho obtido como conseqüência da utilização de vários processadores. Porém, muitas vezes se prefere trabalhar com o conceito de *eficiência* (η) que representa a percentagem de tempo que cada processador está fazendo trabalho útil. A eficiência, pode ser calculada a partir da aceleração (Stone, 1987) conforme:

DEFINIÇÃO 5.2 (EFICIÊNCIA: η)

$$\eta \stackrel{\text{def}}{=} 100 \frac{S_p}{p} \qquad (em\ \%)$$

onde S_p é o 'speedup' e p representa o número de processadores.

A implementação paralela de um algoritmo pode ser feita num contexto síncrono ou assíncrono. Em geral, foi verificado que as versões síncronas são menos eficientes que as correspondentes versões assíncronas, devido às perdas de tempo nas barreiras de sincronização e ao inevitável desbalanceamento de carga computacional. Para representar o ganho introduzido pela utilização de uma versão assíncrona em relação à versão síncrona do mesmo algoritmo, utilizamos o conceito de *aceleração devida ao assincronismo* A_a, definido a seguir:

DEFINIÇÃO 5.3 (ACELERAÇÃO DEVIDA AO ASSINCRONISMO: A_a)
A aceleração devida ao assincronismo A_a é dada pela relação entre os tempos de execução das versões síncrona e assíncrona do algoritmo considerado, num mesmo sistema distribuído.

$$A_a \stackrel{\text{def}}{=} \frac{\textit{tempo de execução da versão síncrona}}{\textit{tempo de execução da versão assíncrona}}$$

Considerando a Definição 5.1 de aceleração é possível utilizar um tempo seqüencial bem específico, em lugar do ambíguo 'melhor tempo seqüencial'. É assim que em muitos casos o 'speedup' é definido como a relação de tempos entre as execuções seqüencial e paralela do mesmo algoritmo em um mesmo processador. Para o caso específico dos Team Algorithms que combinam métodos diferentes, podemos definir a *aceleração relativa de um TA* (A_r) considerando como 'melhor tempo seqüencial' ao menor tempo dentre todos os algoritmos sendo combinados, quando utilizados individualmente para resolver o problema global, num contexto seqüencial.

DEFINIÇÃO 5.4 (ACELERAÇÃO RELATIVA DE UM TA: A_r)
A aceleração relativa A_r de um Team Algorithm que combina diferentes algoritmos é dada pela relação entre o melhor tempo seqüencial dentre todos os métodos combinados, quando utilizados individualmente para resolver o problema global, e o tempo de execução do Team Algorithm considerado, num sistema distribuído, i.e.

$$A_r \stackrel{\text{def}}{=} \frac{\textit{melhor tempo seqüencial dentre os métodos combinados por um TA}}{\textit{tempo da execução paralela do TA considerado}}$$

A aceleração relativa de um TA, acima definida, se mostra uma figura de mérito apropriada para representar o ganho de se utilizar um Team Algorithm num contexto distribuído. De forma análoga à Definição 5.2, podemos definir a *eficiência relativa de um TA* (η_r) , correspondente a A_r, como:

DEFINIÇÃO 5.5 (EFICIÊNCIA RELATIVA DE UM TA: η_r)

$$\eta_r \stackrel{\text{def}}{=} 100\,\frac{A_r}{p} \qquad (em\ \%)$$

onde A_r é a aceleração relativa do TA considerado e p representa o número de processadores.

Uma outra figura de mérito de interesse para o caso específico dos Team Algorithms é a aceleração (ou ganho) devido à combinação de diferentes algoritmos, mesmo num contexto seqüencial. Esclarecemos este conceito com o Exemplo 3.1, no qual cada um dos algoritmos sendo combinados pelo TA ou não converge ou converge em mais de duzentas iterações, enquanto o TA resolve o problema em 3 iterações. Para estes casos é útil definir a *aceleração seqüencial de um TA* (A_s) que permite representar o ganho de se usar um TA em relação à utilização dos algoritmos que o compõem, mesmo no contexto seqüencial:

DEFINIÇÃO 5.6 (ACELERAÇÃO SEQÜENCIAL DE UM TA: A_s)
A aceleração seqüencial A_s de um Team Algorithm que combina diferentes algoritmos é dada pela relação entre o melhor tempo seqüencial dentre todos os métodos combinados, quando utilizados individualmente para resolver o problema global, e o tempo seqüencial do Team Algorithm considerado, i.e.

$$A_s \stackrel{\text{def}}{=} \frac{\textit{melhor tempo seqüencial dentre os métodos combinados por um TA}}{\textit{tempo da execução seqüencial do TA considerado}}$$

A aceleração seqüencial de um TA, se mostra numa figura de mérito apropriada para representar o ganho em se utilizar um Team Algorithm independente do ganho devido ao paralelismo.

Nota 5.2 Enfatiza-se que quando as definições acima se referem ao *melhor tempo seqüencial,* consideram ambas as implementações: com e sem 'overlapping'. Portanto, se a utilização de 'overlapping' parcial melhora o tempo de execução seqüencial, o *melhor tempo* se refere à resolução do sistema expandido; caso contrário, se refere à resolução do sistema original (sem 'overlapping').

Finalizamos esta seção considerando o caso particular para o qual 'o melhor tempo seqüencial' é obtido da execução seqüencial de um Team Algorithm e o melhor tempo paralelo é obtido da versão assíncrona do TA considerado; sendo assim, a aceleração relativa máxima A_r, correspondente à versão assíncrona, pode ser relacionada com as outras acelerações acima definidas:

$$A_r = A_s \times S_p \times A_a \tag{5.1}$$

onde S_p representa a aceleração da versão síncrona do TA. Da relação 5.1 resulta evidente que a aceleração relativa máxima de um TA, para o caso considerado, é constituida por três *fatores de ganho:*

1. a própria utilização do TA que mesmo no contexto seqüencial melhora o desempenho em relação aos algoritmos que o compõem de um fator A_s;

2. o ganho correspondente ao paralelismo, considerando um contexto síncrono, dado por S_p; e finalmente,

3. o ganho devido ao assincronismo, dado por A_a.

5.2 Implementação de um bloco TA sem 'overlapping'

O objetivo desta seção é apresentar e discutir os resultados experimentais da implementação de um bloco TA sem 'overlapping' realizada no hipercubo iPSC/860 da Intel, com o objetivo de resolver o problema do fluxo de potência elétrica dado pelo Problema Exemplo 1, apresentado na Seção 4.3.

Para enfatizar as vantagens de utilizar um bloco TA na resolução do Problema Exemplo 1, foram implementadas também versões seqüencias dos algoritmos individuais, que são:

- o método da matriz Y (versão bloco Jacobi); e
- o método Desacoplado Rápido;

de forma a se calcular os parâmetros de comparação definidos na seção 5.1. O bloco TA proposto na seção 4.3 foi implementado em versões seqüenciais e paralelas síncronas e assíncronas, utilizando no caso paralelo, 2, 4 e 8 processadores.

Um caso típico é ilustrado na Tabela 5.1 que apresenta os melhores resultados experimentais obtidos, assim como as correspondentes figuras de comparação, para cada uma das versões acima citadas. Foram estudados diversas variantes do caso ilustrado na Tabela 5.1, considerando distintas condições inicias $x(0)$ assim como tolerâncias (ε) diferentes. As tabelas 5.2 a 5.4 ilustram alguns destes casos.

A seguir, discutimos os resultados experimentais apresentados nas tabelas 5.1 a 5.4:

- os melhores tempos de execução nem sempre são obtidos para uma carga computacional bem balanceada, como ilustra a Tabela 5.5;
- o bloco TA sem 'overlapping' é claramente superior a qualquer um dos algoritmos que o compõem, nas quatro tabelas apresentadas. Ainda mais, existem casos, como os ilustrados nas tabelas 5.2 e 5.3, nos quais somente o Team Algorithm consegue resolver o problema considerado;
- dos resultados apresentados nas tabelas 5.2 e 5.3, concluimos que a combinação de dois algoritmos que individualmenta não conseguem resolver um dado problema,

Problema Exemplo 1								
$\varepsilon = 0.001$ e $x(0)$ = 'flat start'								
ALGORITMO	p	*iter*	tempo [seg]	A_r	η_r (%)	S_p	η (%)	A_a
Matriz Y (Jacobi)	1	∞	∞	-	-	-	-	-
Desacoplado Rápido	1	131	9.91836	**1**	**100**	0.12	12	-
TA seqüencial	1	33	1.16475	8.52	852	**1**	**100**	-
TA síncrono	2	30	1.01661	9.8	488	1.15	57	1
	4	30	0.80767	12.3	307	1.4	36	1
	8	31	0.49818	19.9	249	2.34	29	1
TA assíncrono	2	65	0.64777	15.3	766	1.8	90	1.56
	4	43	0.48398	20.5	512	2.4	60	1.67
	8	32	0.38236	25.9	324	3.1	38	1.30

Tabela 5.1: Melhores tempos experimentais medidos no iPSC/860.

Problema Exemplo 1					
$\varepsilon = 0.0005$ e $x(0)$ = 'flat start'					
ALGORITMO	p	*iter*	tempo [seg]	S_p	η
Matriz Y (Jacobi)	1	∞	∞	-	-
Desacoplado Rápido	1	∞	∞	-	-
TA seqüencial	1	30	1.36279	1	100 %
TA síncrono	8	34	0.55399	2.5	31 %
TA assíncrono	8	34	0.41019	3.3	42 %

Tabela 5.2: Melhores tempos experimentais medidos no iPSC/860.

Problema Exemplo 1					
$\varepsilon = 0.0001$ e $x(0)$ = 'flat start'					
ALGORITMO	p	*iter*	tempo [seg]	S_p	η
Matriz Y (Jacobi)	1	∞	∞	-	-
Desacoplado Rápido	1	∞	∞	-	-
TA seqüencial	1	42	1.79387	1	100 %
TA síncrono	8	65	0.96642	1.9	23 %
TA assíncrono	8	34	0.50638	3.5	44 %

Tabela 5.3: Melhores tempos experimentais medidos no iPSC/860.

Problema Exemplo 1								
$\varepsilon = 0.001$ e $x(0) \approx x^*$ com $\Delta S = 0.05$								
ALGORITMO	p	*iter*	tempo [seg]	A_r	η_r	S_p	η	A_a
Matriz Y (Jacobi)	1	∞	∞	-	-	-	-	-
Desacoplado Rápido	1	36	3.17046	**1**	**100 %**	0.5	48 %	-
TA seqüencial	1	22	1.53266	2.1	207 %	**1**	**100 %**	-
TA síncrono	8	19	0.33722	9.4	118 %	4.5	57 %	1
TA assíncrono	8	19	0.23535	13.5	168 %	6.5	81 %	1.43

Tabela 5.4: Melhores tempos experimentais medidos no iPSC/860.

Problema Exemplo 1		
$\varepsilon = 0.001$ e $x(0)$ = 'flat start'		
ALGORITMO	p	PARTIÇÃO
Matriz Y (bloco-Jacobi)	1	616
Desacoplado Rápido	1	616
TA seqüencial	1	77; 539
TA síncrono	2	77; 539
	4	154; $\cdots$; 154
	8	77; $\cdots$; 77
TA assíncrono	2	77; 539
	4	77; 154; 154; 231
	8	77; $\cdots$; 77

Tabela 5.5: Partições utilizadas nas medições de tempo da Tabela 5.2.

BLOCO TA ASSÍNCRONO $p = 8$ e $x(0)$ = 'flat start'			
ε	tempo [seg]	S_p	η
0.001	0.38236	3.1	38 %
0.0005	0.41019	3.3	42 %
0.0001	0.50638	3.5	44 %

Tabela 5.6: Variação de S_p com a tolerância para o Problema Exemplo 1.

pode resultar em um novo algoritmo combinado que resolve sem dificuldades o problema considerado, o que poderíamos chamar de *efeito sinergético*;

- em todas as tabelas apresentadas, 'o melhor tempo seqüencial' é aquele obtido pela versão seqüencial do bloco TA e o melhor tempo paralelo pela versão assíncrona; conseqüentemente, a relação 5.1 pode ser utilizada. Como exemplo, da Tabela 5.1, obtemos uma aceleração relativa ao bloco TA assíncrono que utiliza 8 processadores de $A_r = 25.9$; enquanto $S_p = 2.34$ para a correspondente versão síncrona; sendo ademais $A_s = 8.52$ e $A_a = 1.30$, i.e. a aceleração relativa máxima do TA A_r pode ser calculada como $A_s \times S_p \times A_a$. Enfatizamos que A_r é muito maior que o número de processadores $(p = 8)$, não pelo uso do paralelismo (de fato $S_p \times A_a < p$), mas pela aceleração seqüencial do bloco TA considerado;

- para o problema de 616 barras considerado, a aceleração aumenta com o número (p) de processadores, enquanto a eficiência diminui com p; como ilustra a Tabela 5.1;

- as versões síncronas são menos eficientes que as correspondentes versões assíncronas, devido ao inevitável desbalanceamento de carga computacional para uma partição dada. As tabelas 5.1 a 5.4 ilustram esta clara superioridade das versões assíncronas, em termos de aceleração e eficiência;

- A Tabela 5.6 enfatiza que a eficiência das versões assíncronas aumenta quando a tolerância se torna mais estrita (o valor de ε diminui).

- a excelente velocidade de convergência que apresentam as versões assíncronas quando estão operando numa região de 'boa convergência', pode ser apreciada comparando-se a aceleração e eficiência obtidas com diferentes condições inicias. A Tabela 5.7 compara as melhores acelerações das tabelas 5.1 e 5.4, enfatizando que tanto S_p quanto η aumentam se a condição inicial é escolhida mais próxima da solução x^*.

Para concluir esta seção, mencionamos que não foi utilizada nenhuma versão com 'overlapping' para resolver o Problema Exemplo 1, dado que o bloco TA sem 'overlapping' resolve sem dificuldades o problema considerado e a utilização de 'overlapping'

BLOCO TA ASSÍNCRONO $p = 8$ e $\varepsilon = 0.001$				
condição inicial	*iter*	tempo [seg]	S_p	η
$x(0)$ = 'flat start'	32	0.38236	3.1	38 %
$x(0) \approx x^*$ com $\Delta S = 0.05$	19	0.23535	6.5	81 %

Tabela 5.7: Variação de S_p com $x(0)$ para o Problema Exemplo 1.

implica num custo computacional adicional que se mostra desnecessário para este exemplo. Porém, na seção que segue considera-se um problema que não pode ser facilmente resolvido sem usar 'overlapping'; portanto, para esse tipo de problemas mais complexos, um TA Generalizado se mostra atraente apesar do sobrecusto computacional.

5.3 Implementação de um TA Generalizado

Nesta seção, apresenta-se os resultados experimentais da implementação de um Team Algorithn Generalizado que utiliza o conceito de 'overlapping' parcial, realizada no hipercubo iPSC/860 da Intel, com o objetivo de resolver o problema do fluxo de potência elétrica, dado pelo Problema Exemplo 2 apresentado na Seção 4.4. Uma discassão dos resultados é incluida para analisar as vantagens de se utilizar um TA Generalizado com 'overlapping' e comparar experimentalmente algumas das versões assíncronas dos TA estudadas no Capítulo 3.

Analogamente ao estudo experimental dos bloco TA sem 'overlapping', foram implementadas também as versões seqüenciais dos algoritmos 'individuais', assim como do TA Generalizado, com e sem 'overlapping'.

O TA Generalizado proposto na seção 4.4, assim como o bloco TA da seção 4.3, foram implementados em versões paralelas síncronas e assíncronas, utilizando 2 e 4 processadores.

Um caso típico para algumas versões citadas na seção 4.4[1] é ilustrado na Tabela 5.8 que apresenta os melhores resultados experimentais obtidos, assim como as correspondentes figuras de comparação.

Foram também estudadas diversas variantes do caso ilustrado na Tabela 5.8, considerando distintas condições inicias e tolerâncias. As tabelas 5.9 a 5.11 ilustram alguns destes casos.

[1]As versões consideradas são: a versão A com Administrador em processador dedicado, a versão D com Administrador distribuido implícito e a versão E com Administrador implícito no processador 1.

Problema Exemplo 2 $\varepsilon = 0.001$ e $x(0)$ = 'flat start'						
ALGORITMO	'overlap'	p	*iter*	tempo [seg]	S_p	η (%)
Matriz Y (BJ)			-		-	-
método DR	não	1	-	não converge	-	-
TA seqüencial			-		-	-
TA seqüencial	sim	1	53	2.33379 a 2.33396	**1**	**100**
TA	não	2	-	não	-	-
síncrono		4	-	converge	-	-
TA	não	2	-	convergência	-	-
assíncrono		4	-	ocasional	-	-
TA	sim	2	64	1.82564 a 1.82789	1.3	64
síncrono		4	84	1.32385 a 1.32602	1.8	44
	versão A		138 a 184	1.73598 a 2.30847	1.3	45
TA	versão D	2	146 a 189	1.57115 a 2.03278	1.5	74
assíncrono	versão E		172 a 194	1.76724 a 2.10294	1.3	66
com	versão A		62 a 93	1.11098 a 1.64973	2.1	42
'overlapping'	versão D	4	83 a 410	0.97077 a 1.26550	2.4	60
	versão E		68 a 79	0.79059 a 0.89672	3	74

Tabela 5.8: `Melhores tempos experimentais medidos no iPSC/860.`

Problema Exemplo 2 $\varepsilon = 0.0005$ e $x(0)$ = 'flat start'						
ALGORITMO	'overlap'	p	*iter*	tempo [seg]	S_p	η (%)
Matriz Y (BJ)			-		-	-
método DR	não	1	-	não converge	-	-
TA seqüencial			-		-	-
TA seqüencial	sim	1	71	3.00175 a 3.00176	**1**	**100**
TA síncrono	não		-	não converge	-	-
TA assíncrono	não	4	-	convergência ocasional	-	-
TA síncrono	sim		98	1.58010 a 1.58039	1.9	47
TA	versão A		74 a 117	1.30439 a 2.06648	2.3	46
assíncrono	versão D	4	96 a 102	1.11423 a 1.18662	2.7	67
com 'overlapping'	versão E		76 a 82	0.88137 a 0.95324	3.4	85

Tabela 5.9: `Tempos experimentais típicos, medidos no iPSC/860.`

Problema Exemplo 2						
$\varepsilon = 0.0001$ e $x(0)$ = 'flat start'						
ALGORITMO	'overlap'	p	*iter*	tempo [seg]	S_p	η (%)
Matriz Y (BJ)			-		-	-
método DR	não	1	-	não converge	-	-
TA seqüencial			-		-	-
TA seqüencial	sim	1	91	3.76373 a 3.76389	**1**	**100**
TA síncrono	não		-	não converge	-	-
TA assíncrono	não	4	-	convergência ocasional	-	-
TA síncrono	sim		127	2.03969 a 2.04029	1.8	46
TA	versão A		120 a 159	2.12954 a 2.81327	1.8	35
assíncrono	versão D	4	126 a 134	1.48544 a 1.59473	2.5	63
com 'overlapping'	versão E		92 a 106	1.07906 a 1.24149	3.5	87

Tabela 5.10: Tempos experimentais típicos, medidos no iPSC/860.

Problema Exemplo 2						
$\varepsilon = 0.001$ e $x(0) \approx x^*$ com $\Delta S = 0.05$						
ALGORITMO	'overlap'	p	*iter*	tempo [seg]	S_p	η (%)
Matriz Y (BJ)			-	não converge	-	-
método DR	não	1	5	1.05268 a 1.05281	**1**	**100**
TA seqüencial			-	não converge	-	-
TA seqüencial	sim	1	31	1.49757 a 1.49788	0.7	70
TA síncrono	não		-	não converge	-	-
TA assíncrono	não	4	-	convergência ocasional	-	-
TA síncrono	sim		28	0.63664 a 0.63966	1.7	41
TA	versão A		28 a 44	0.50505 a 0.76776	2.1	42
assíncrono	versão D	4	42 a 59	0.49685 a 0.69122	2.1	53
com 'overlapping'	versão E		36 a 68	0.42965 a 0.80798	2.5	61

Tabela 5.11: Tempos experimentais típicos, medidos no iPSC/860.

Observa-se que os resultados e as conclusões (da análise) obtidas em relação aos bloco TA sem 'overlapping' no caso do Problema Exemplo 1 podem ser replicados quase que sem alteração no caso do Problema Exemplo 2. Por este motivo, nesta seção nos limitamos a enfatizar aquelas características que são próprias do uso de Administradores com 'overlapping', ou específicas do Problema Exemplo 2. Desta forma, discutimos a seguir os resultados experimentais apresentados nas tabelas 5.8 a 5.11:

- as tabelas 5.8 a 5.11 apresentam os tempos mínimos e máximos medidos em execuções sucessivas da mesma implementação, assim como as variações no número de iterações. Pode-se notar a grande variação tanto em número de iterações como nos tempos, para as versões assíncronas;

- os métodos sem 'overlapping' têm dificuldades para resolver o Problema Exemplo 2 e, em geral, não convergem quando a condição inicial $x(0)$ é dada pelo correspondente 'flat start' definido na subseção 4.1.1. A única exceção é a versão assíncrona do TA sem 'overlapping' que ocasionalmente, dependendo de como se dá o assíncronismo, pode convergir;

- a vantagem de se utilizar um TA Generalizado (com 'overlapping') se mostra assim evidente, por ser o único algoritmo capaz de resolver o problema, com qualquer uma das versões consideradas: seqüencial ou paralela, síncrona ou assíncrona;

- o melhor desempenho das implementações assíncronas, com respeito à sua versão síncrona, pode ser verificado nas tabelas 5.8 a 5.11;

- devido ao fato de que o possível assíncronismo com apenas 2 processadores não é muito interessante, as tabelas 5.9 a 5.11 somente apresentam resultados para $p = 4$ processadores. Sendo assim, e considerando a eficiência η, podemos comparar as diferentes versões assíncronas e concluir que:

    - nas tabelas 5.8 a 5.11 notamos que as versões com Administrador implícito são melhores que as que utilizam um Administrador explícito em processador dedicado, conforme foi concluido da discussão da seção 3.4; portanto, a versão A é a menos eficiente das três apresentadas nas tabelas 5.8 a 5.11;

    - a replicação do Administrador introduz una pequena ineficiência, portanto, a versão E do TA Generalizado é, em geral, melhor que a versão D;

    - concluimos que em geral, das três versões implementadas, a versão E do TA Generalizado é a mais recomendável, seguida pela versão D e finalmente pela versão A;

- a ordenação das versões acima realizadas é uma simples recomendação; portanto, existem exceções, como a mostrada na Tabela 5.8 para o caso de dois processadores, onde a eficiência da versão D é maior que para a versão E. Esclarecemos

VERSÃO E DO TA GENERALIZADO $p = 4$ e $x(0)$ = 'flat start'			
ε	tempo [seg]	S_p	η
0.001	0.79059	3	74 %
0.0005	0.88137	3.4	85 %
0.0001	1.07906	3.5	87 %

Tabela 5.12: Variação de S_p com a tolerância para o Problema Exemplo 2.

neste sentido que as figuras de comparação, foram calculadas a partir dos melhores tempos; porém, se consideramos os piores tempos, continua sendo válido a recomendação de utilizar a versão E, mesmo para este caso excepcional;

- a partir dos resultados das tabelas 5.8 a 5.10, continua visível a existência de um *efeito sinergético*, entendido como a possibilidade do TA Generalizado resolver problemas que nenhum dos algoritmos que o compõem consegue resolver individualmente;

- o fato de que a eficiência das versões assíncronas aumenta quando a tolerância diminui continua existindo, como fica mostrado na Tabela 5.12;

- Da Tabela 5.11 fica evidente o bom desempenho do método Desacoplado Rápido (converge em somente 5 iterações) para uma condição inicial bem próxima da solução. Observa-se que a versão E do TA Generalizado consegue, por outro lado, melhorar esse desempenho com uma aceleração de 2.5, utilizando 4 processadores, o que representa uma eficiência de 61%.

Finalizamos esta seção lembrando que, para se obter os excelentes resultados experimentais ilustrados nas tabelas 5.8 a 5.11 que estavamos discutindo para os Team Algorithms Generalizados, foi necessário realizar uma escolha adequada dos pesos utilizados pelo Administrador. Esta escolha foi realizada com base experimental, resolvendo repetidamente o mesmo problema com diferentes pesos.

Mapas de Iterações, como o ilustrado na Figura 5.1, foram levantados para a maioria dos casos experimentados. Desta forma, foi possível verificar, experimentalmente, que para a maioria dos problemas a seguinte escolha dos pesos (w_1, w_2) se mostra satisfatória se:

$$w_1 + w_2 = 1 + \delta \tag{5.2}$$

onde $0 < \delta < 1$. Em outras palavras, os pesos podem ser escolhidos com um certo grau de *sobre-relaxação combinada*, conforme discussão do caso 2 da subseção 3.1.1.

Afortunadamente, a região do Mapa de iterações onde o TA Generalizado tem uma boa convergência é bem ampla, como ilustra a Figura 5.1. Portanto, sempre que a escolha de pesos seja razoavelmente boa, o fato do algoritmo convergir (ou divergir), não

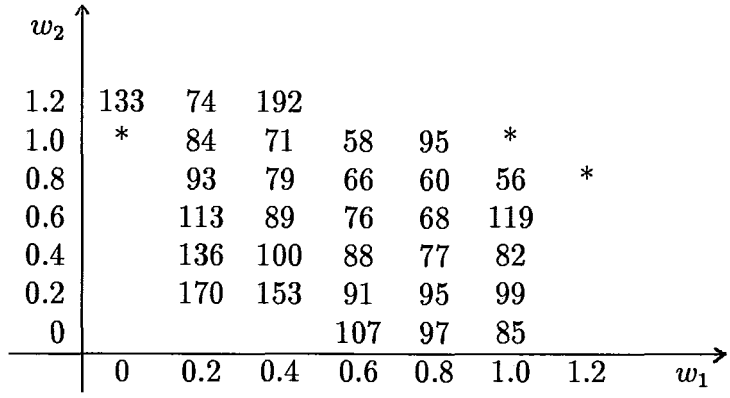

Figura 5.1: Mapa de Iterações do TA seqüencial com 'overlapping'.

é muito sensível à escolha dos pesos. Contudo, resulta evidente que uma boa escolha de pesos ajuda a obter melhores figuras de mérito (como eficiência e 'speedup').

Nas implementações realizadas para resolver o Problema Exemplo 2, não foram estudados os Team Algorithms Generalizados que utilizam pesos variáveis a cada iteração. Efetivamente, os pesos foram mantidos constantes, com a única exceção do caso em que o valor de uma voltagem saia da região de interesse, o que era resolvido fazendo correções menores que as calculadas (modificação de pesos). Conseqüentemente, a experimentação com políticas que permitam escolher os pesos, como as discutidas na subseção 3.1.2, é um ponto a ser ainda pesquisado.

As implementações dos Team Algorithms, realizadas para resolver o problema do fluxo de potência elétrica no contexto assíncrono do iPSC/860, permitiu uma enriquecedoura experiência trabalhando com assíncronismo, uma área relativamente nova com escassos resultados experimentais. Tentar resumir no papel o resultado destes anos de experimentação com assíncronismo não é tarefa fácil. Porém, enfatizamos a seguir as características mais sobresalentes:

- por ser uma área relativamente nova de trabalho, problemas não documentados podem acontecer. O mais crítico em nosso trabalho, foi a impossibilidade de se controlar efetivamente o sistema de 'buffers', o que levou à necessidade de reinicializar o sistema computacional para recuperar alguns nós que ficaram com os 'buffers' cheios, como foi discutido na seção 4.2;

- quando desejamos encontrar uma partição adequada para um dado problema, devemos buscar uma partição que permita constantes bloco-Lipschitz pequenas mesmo que isto implique em um desbalanceamento de carga, caso ambas condições não possam ser simultaneamente obtidas. Assim, muitos sistemas bem balanceados divergem; enquanto boas partições que asseguram a convergência

Problema Exemplo 1			
p	tempo total [seg]	tempo de sincronização + comunicação	Percentagem
2	1.92	0.07	4 %
	2.01	0.10	5 %
	2.10	0.13	6 %
4	0.81	0.07	9 %
	0.90	0.11	12 %
	0.99	0.17	17 %
8	2.56	2.13	83 %
	2.57	2.17	84 %
	2.60	2.20	85 %

Tabela 5.13: `Tempos típicos de sincronização e comunicação.`

conseguem resolver o problema, mesmo que lentamente. Ainda mais, o assincronismo é uma técnica apropriada para lidar com estes desbalanceamentos de carga;

- a linguagem de programação utilizada na codificação permite realizar comunicações síncronas de forma simples e segura, porém pouco eficientes se consideramos a versão assíncrona. A Tabela 5.13 ilustra a alta percentagem de tempo utilizada na comunicação e sincronização, nas implementações síncronas do bloco TA. Note que a ineficiência aumenta com o número de processadores, o que leva a pensar que, em sistemas de paralelismo massivo, as implementações assíncronas são as mais promissoras;

- a execução de um mesmo programa, quando utiliza comunicação assíncrona, pode diferir consideravelmente de uma execução para outra, dependendo da carga computacional do sistema, e portanto do trabalho realizado pelos outros usuarios do sistema; assim, variações na ordem de 15% a 20% resultaram normais em nossas implementações;

- para certos problemas, o *'caminho'* seguido pelo algoritmo assíncrono, até a solução, depende das condições de carga computacional; o que leva o algoritmo a convergir num número de iterações (ou tempo) bem diferente, dependendo do *'caminho'* seguido. Todavia, em condições de boa convergência, os tempos não variam muito de uma execução para outra.

Finalizamos este capítulo enfatizando as duas características mais importantes verificadas a partir das implementações computacionais:

1. o desempenho das implementações assíncronas pode ser bem superior à correspondente versão síncrona, especialmente em presença de desbalanceamento de carga e um número elevado de processadores;

2. um Team Algorithm pode resolver problemas que nenhum dos algoritmos que o compõem consegue resolver individualmente. Todavia, mesmo que em alguns casos, algum destes algoritmos resolva individualmente o problema considerado, o TA pode se mostrar mais eficiente.

Sumário do Capítulo 5

Neste capítulo foram apresentados os principais resultados experimentais e conclusões das implementações realizadas no hipercubo iPSC/860, para resolver os problemas de fluxo de potência elétrica chamados neste trabalho de Problemas Exemplos 1 e 2.

Inicialmente, foram definidas figuras de comparação para o contexto distribuído dos Team Algorithms. Assim foram definidos os conceitos de:

- aceleração e eficiência;
- aceleração devida ao assincronismo;
- aceleração e eficiência relativa de um TA;
- aceleração seqüencial de um TA;

Em seguida foram apresentados os resultados experimentais das implementações realizadas para resolver o Problema Exemplo 1 utilizando um bloco TA sem 'overlapping'. Estes resultados foram analisados destacando-se a capacidade do bloco TA de resolver problemas para os quais os algoritmos sendo combinados ou não convergem ou o resolvem com um desempenho muito pobre, quando comparados com o TA. Foi também enfatizada a superioridade das implementações assíncronas sobre as suas versões síncronas, quando existe um inevitável desbalanceamento de carga ou cresce o número de processadores do sistema. Também foi verificado que a eficiência η das implementações assíncronas cresce em regiões de boa convergência; como exemplo, a eficiência aumenta quando a tolerância é mais estrita.

Por sua complexidade e características próprias, o Problema Exemplo 2 somente foi possível resolver a partir de 'flat start' utilizando um Team Algorithm com 'overlapping' parcial. Dentre todas as versões implementadas, resultou evidente o melhor desempenho das implementações assíncronas, e dentre estas, da versão E com Administrador implícito e centralizado sem processador dedicado. Um *Mapa de Iterações* típico

foi apresentado exemplificando a região do plano definido pelos pesos, onde geralmente tem-se uma escolha eficiente dos mesmos.

Finalmente, foi destacada uma série de experiências obtidas a partir do trabalho realizado com assincronismo, uma área de pesquisa ainda nova com poucos resultados experimentais, mas promissora para sistemas computacionais de paralelismo massivo, como demonstraram os resultados experimentais obtidos.

Capítulo 6

Conclusões

A idéia de se combinar métodos diferentes para resolver um sistema de equações algébricas utilizando um sistema computacional, foi postulado há várias décadas atrás. O interesse nestas idéias foi crescendo consideravelmente a partir da década de 80, como conseqüência direta da disponibilidade de sistemas computacionais de memória distribuída caracterizados por um baixo custo por instrução quando comparados com os sistemas chamados 'mainframes'. Contudo, a idéia não foi formalizada nem mesmo no contexto seqüencial. Conseqüentemente, as principais contribuições deste trabalho foram: a formalização e a conseqüente análise dos *Team Algorithms* no contexto assíncrono.

A resolução de sistemas de equações de grande porte em sistemas computacionais distribuídos requer a partição do problema considerado em subproblemas menores, que possam ser resolvidos em cada um dos processadores pelo método mais adequado. O *Team Algorithm* se mostra assim como a solução mais apropriada; especialmente quando consideramos a capacidade dos mesmos de trabalhar no contexto assíncrono, dado que desbalanceamentos de carga são em geral inevitáveis, o que reduz a eficiência das implementações síncronas.

As resenhas históricas sobre os *Team Algorithms* e o contexto assíncrono, realizadas no Capítulo 1, permitiu apresentar o contexto geral e as motivações que deram origem a este trabalho, enfatizando-se que a combinação de diferentes métodos num contexto assíncrono é uma área nova de trabalho, e em certo sentido, ainda inexplorada, considerando-se que os primeiros trabalhos apareceram somente na década de 90. Sendo assim, é fácil entender a ausência de resultados tanto teóricos como experimentais no contexto assíncrono dos sistemas computacionais distribuídos; conseqüentemente, outra contribuição deste trabalho é apresentar os primeiros resultados experimentais efetivos, na resolução de sistemas de equações algébricas de porte médio no contexto paralelo assíncrono.

Considerando a complexidade da formulação matemática dos sistemas assíncronos, o desafio inicial deste trabalho foi a derivação de uma metodologia geral que permi-

tisse obter condições de convergência no contexto geral dos algoritmos bloco iterativos assíncronos, que pudesse ser também utilizada no caso dos *Team Algorithms*. Afortunadamente, a relativa maturidade dos estudos teóricos sobre assíncronismo do grupo de computação paralela da COPPE permitiu derivar o Teorema 2.1 e conseqüentemente toda a metodologia do Capítulo 2. Este fato permitiu derivar condições suficientes de convergência para diferentes versões dos algoritmos bloco iterativos assíncronos, tais como: o bloco Jacobi, o método geral das cordas paralelas, do qual o Newton simplificado é um caso particular, o método da solução por componente e os métodos contrativos, e como conseqüência natural, para os *Team Algorithms*. Da análise das condições suficientes de convergência, foi possível postular a utilização do raio espectral da matriz de comparação como *Indicador de Mérito*, de forma a possibilitar uma comparação 'a priori' entre as diferentes alternativas de solução disponíveis no contexto dos algoritmos bloco iterativos assíncronos.

No Capítulo 3, os TA foram formalizados e analisados para o caso geral, no contexto assíncrono dos sistemas distribuídos. Nessa oportunidade, utilizamos o nome específico de *Team Algorithms Generalizados* para enfatizar que os TA incluem como casos particulares :

- os bloco TA estudados no Capítulo 2 (sem 'overlapping');
- o conceito de 'partial overlapping' proposto em Ikeda e Šiljak (1980) ('overlapping' parcial); assim como
- a proposta de Talukdar et al. (1983) que considera o uso de um Administrador centralizado operando sobre o problema todo ('overlapping' total);

enfatizou-se também a capacidade do *Team Algorithm* de combinar as principais vantagens de todas estas propostas. Note que ademais, o *A-team* proposto em Talukdar et al. (1992) pode também ser considerado como um TA Generalizado com 'overlapping' total que utiliza um Administrador distribuído *'inteligente'* (de pesos variáveis).

Para o caso geral acima, foi analisada a utilização de um Administrador encarregado de *administrar* as diferentes versões de uma mesma variável gerada pela utilização de 'overlapping'. Várias políticas para a escolha dos pesos utilizados no Administrador foram estudadas; comprovando-se experimentalmente que a escolha de pesos constantes se mostra satisfatoria em geral quando comparada ao caso de pesos variáveis.

Também foram estudadas várias versões de TA Generalizados utilizando 'overlapping' parcial, conforme seja o tipo de Administrador utilizado, derivando-se para cada uma destas versões, uma condição suficiente de convergência. Comparações teóricas utilizando o indicador de mérito proposto permitiram recomendar as versões com Administrador implícito, o que foi comprovado experimentalmente. Como exemplo, nos sistemas implementados, a versão E, aquela com Administrador implícito e centralizado no processador 1 (não dedicado), resultou a mais eficiente nos problemas apresentados no Capítulo 4.

Conseqüentemente, a utilização de um *Team Algorithm* permite:

- Resolver sistemas de equações algébricas que os algoritmos 'individuais' não resolvem isoladamente (efeito sinergético).
- Viabilizar a resolução eficiente de sistemas de equações algébricas de grande porte em sistemas computacionais de memória distribuída, caracterizados por seu baixo custo por instrução.
- Obter um considerável 'Speedup', quando problemas de grande porte são particionados em subproblemas menores que podem ser resolvidos em paralelo. O balanceamento de carga computacional não se mostra tão crítico nas versões assíncronas (comparadas com a correspondente versão síncrona).
- Aproveitar a velocidade de algoritmos rápidos que nem sempre convergem no domínio de interesse. Para isto, o TA combina o algoritmo rápido com um outro método capaz de assegurar a convergência no domínio considerado.
- Evitar a divergência prematura de alguns blocos; a partir de uma apropriada escolha de pesos no Administrador.
- Acelerar a convergência bem como viabilizar a resolução de sistemas de equações na presença de *blocos críticos* fortemente acoplados a outros blocos, com a utilização de 'overlapping' parcial.

Considerando que as diversas áreas da engenharia são as primeiras nas quais a metodologia formalizada no presente trabalho pode ser utilizada, foi escolhido o problema do *fluxo de potência elétrica* como problema exemplo, dado que existem problemas de grande porte que atualmente requerem a utilização de 'mainframes' de elevado custo. A análise de vários métodos de solução no contexto seqüencial, como o método da matriz Y (versão bloco Jacobi) e o método Desacoplado Rápido, permitiu postular versões bloco iterativas assíncronas destes métodos e dos TA obtidos a partir da combinação dos mesmos, capazes de viabilizar a solução deste tipo de problemas em sistemas computacionais distribuídos de menor custo que os 'mainframes' hoje utilizados. Análises de convergência foram incluidas, para cada caso, exemplificando a metodologia geral apresentada nos Capítulos 2 e 3.

Tomando por base o problema do fluxo de potência elétrica, dois problemas exemplos foram postulados e resolvidos experimentalmente, possibilitando desta forma a verificação das características dos *Team Algorithms* em sistemas reais de porte médio. As implementações realizadas permitiram verificar que:

- o *Team Algorithm* consegue resolver problemas que nenhum dos algoritmos que o compõem consegue resolver individualmente. Ainda mais, mesmo no caso em que um deles resolve o problema completo, resultados experimentais mostram que o TA pode apresentar desempenhos bem superiores;
- existem problemas que nenhum algoritmo sem 'overlapping' consegue resolver. A utilização de um *Team Algorithm* com 'overlapping' parcial permite resolver o problema considerado sem maiores dificuldades;

- as implementações síncronas tem um desempenho fraco quando existe um inevitável desbalanceamento de carga e cresce com o número de processadores. Porém, a versão assíncrona correspondente não é tão sensível a este tipo de problema permitindo um excelente desempenho. Concluimos assim, que o uso de um paralelismo massivo no futuro, poderá fazer com que as implementações assíncronas sejam cada vez mais utilizadas;

- a eficiência (η) dos algoritmos assíncronos cresce nas regiões de 'boa convergência'.

Para concluir, enfatizamos a potencialidade dos *Team Algorithms* na medida que sistemas computacionais distribuídos, e computadores paralelos em particular, estão se tornando cada dia mais acessíveis para organizações comerciais e científicas do mundo todo; sendo assim, este trabalho pretende chamar a atenção para essa importante classe de algoritmos, de forma que as pesquisas possam continuar no futuro. Para isto, encerramos o presente trabalho sugerindo algumas linhas de pesquisa que permitirão dar continuidade ao trabalho até aqui realizado:

- Estudos mais detalhados de Administradores que utilizam pesos variáveis poderiam melhorar os resultados até aqui obtidos.

- Estudo de taxas de convergência.

- Aplicações dos Team Algorithms a outros problemas da engenharia, como a solução de problemas de grande porte, provenientes da utilização de elementos finitos nas áreas de engenharia civil e mecânica.

- Melhorar os métodos automáticos de partição para problemas de grande porte. Isto poderia ser realizado incluindo a possibilidade de obter partições com 'overlapping' parcial, e utilizando como parâmetro, por exemplo, o indicador de mérito dado pela condição suficiente de convergência.

- A partir da formulação dada neste trabalho, avaliar a possibilidade de utilizar versões bloco dos *A-teams* (utilizando 'inteligência'), viabilizando desta forma outros métodos para resolver sistemas de grande porte em computadores de memória distribuída.

- Construir Team Algorithms combinando os algoritmos de otimização propostos na área de inteligência artificial, como os algoritmos genéticos e a técnica de 'Simulated Annealing', com algoritmos numéricos do tipo 'hill climbing'. Alguns trabalhos neste sentido estão sendo realizados com os algoritmos genéticos (veja Souza e Talukdar, 1991), mas não temos conhecimento até hoje de combinações assíncronas utilizando 'Simulated Annealing'.

Apêndice A

Convergência de Métodos Bloco Iterativos Assíncronos

Resume-se a seguir a demonstração do Teorema de convergência para métodos paralelos bloco iterativos, no contexto parcialmente assíncrono da seção 2.1 acima, originalmente derivado em Kaszkurewicz, Bhaya e Šiljak (1990).

Consideramos o método bloco iterativo assíncrono (variante no tempo) dado por:

$$x_i(k+1) \;=\; \sum_{j=1}^{m} \; H_{ij}\Big(x^i(k), k\Big) \;\; x_j(d_j^i(k)) \qquad\qquad \forall i \in \{1,\ldots,m\} \tag{A.1}$$

que é denotado genericamente, em Bhaya et al. (1991), como

$$x(k+1) \;=\; \mathcal{H}\Big(x(k),\, k\Big)\; x(k) \tag{A.2}$$

e que pode ser escrito como

$$x_i(k+1) \;=\; \sum_{j=1}^{m} \; H_{ij}\Big(x_1(d_1^i(k)), \ldots, x_m(d_m^i(k)),\, k\Big) \;\; x_j(d_j^i(k)) \tag{A.3}$$

$\forall i \in \{1,\ldots,m\}$, onde

$$x_j(d_j^i(k)) \;\in\; \Big\{\, x_j(k),\; x_j(k-1),\; \ldots,\; x_j(k-d) \,\Big\} \tag{A.4}$$

para enfatizar a possível dependência de $H_{ij}(\cdot)$ com versões atrasadas até d iterações, dos subvetores x_j.

A convergência do método bloco iterativo assíncrono descrito por (A.3) pode ser analisada num *espaço de estado*, considerando o vetor de estado $X \in \mathbb{R}^{n(d+1)}$ definido como:

$$X(k) \;=\; \begin{bmatrix} X_1(k) \\ \vdots \\ X_m(k) \end{bmatrix} \qquad \text{onde} \quad X_j(k) \;\stackrel{\text{def}}{=}\; \begin{bmatrix} x_j(k) \\ x_j(k-1) \\ \vdots \\ x_j(k-d) \end{bmatrix} \;\in\; \mathbb{R}^{n_j(d+1)} \tag{A.5}$$

e introduzindo a *função de chaveamento* $\Upsilon_{ij}(x_{j(k),k})$ com a propriedade de tomar um único valor do subvetor x_j, atrasado até d iterações, para cada tríplice (i,j,k), i.e.

$$\Upsilon_{ij}(x_{j(k),k}) \in \Big\{ x_j(k),\, x_j(k-1),\, \ldots,\, x_j(k-d) \Big\}, \qquad \forall i,j \in \{1,\ldots,m\} \quad \text{(A.6)}$$

Desta forma, a equação (A.3) pode ser escrita na forma de espaço de estado:

$$X_i(k+1) = \begin{bmatrix} x_j(k+1) \\ x_j(k) \\ \vdots \\ x_j(k-d+1) \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{m} \overline{H}_{ij}(k)\, \Upsilon_{ij}(x_{j(k),k}) \\ x_j(k) \\ \vdots \\ x_j(k-d) \end{bmatrix} \qquad \text{(A.7)}$$

$\forall i \in \{1,\ldots,m\}$, onde

$$\overline{H}_{ij}(k) \stackrel{\text{def}}{=} H_{ij}\Big(x_1(d_1^i(k)),\ldots,\, x_m(d_m^i(k)),\; d_1^i(k),\ldots,\, d_m^i(k)\Big) \in \mathbb{R}^{n_i \times n_j} \qquad \text{(A.8)}$$

A seguir, definimos a matriz agregada $H \in \mathbb{R}^{m\times m}$ como:

$$H = (h_{ij}) \qquad \text{com} \quad h_{ij} \stackrel{\text{def}}{=} \sup_{X,\,k} \|\overline{H}_{ij}(k)\|_\infty \qquad \text{(A.9)}$$

onde h_{ij} é dado pelo valor supremo da norma-∞ de $\overline{H}_{ij}(k)$ definido em (A.8), calculado dentre todos os vetores de estado X e iterações k. Desta forma, podemos estabelecer a seguir o principal resultado de Kaszkurewicz, Bhaya e Šiljak (1990):

TEOREMA A.1
Se em um domínio fechado $D = D_1 \times \cdots \times D_m \subset \mathbb{R}^n$ *temos*

$$\mathcal{H}(D) \subset D$$

e também existem m *números reais positivos* u_i, $i = 1,\ldots,m$, *tal que*

$$\max_{1\le i\le m} \Big\{ u_i^{-1} \sum_{j=1}^{m} u_j\, h_{ij} \Big\} < 1 \qquad \text{(A.10)}$$

onde H *é a matriz agregada do algoritmo bloco iterativo assíncrono representado por (A.3), i.e.*

$$H = (h_{ij}) \qquad \textit{com} \quad h_{ij} \stackrel{\text{def}}{=} \sup_{X,\,k} \|\overline{H}_{ij}(k)\|_\infty$$

então, a solução nula de (A.3) é localmente, exponencialmente, estável no domínio D.

Prova.
Para simplificar a notação da prova, introduzimos algumas notações a seguir:

$$\mathbf{m} \stackrel{\text{def}}{=} \{1,\ldots,m\} \qquad \text{(A.11)}$$

$$\mathbf{d} \stackrel{\text{def}}{=} \{0, 1, \ldots, d\} \tag{A.12}$$

$$(\mathbf{d-1}) \stackrel{\text{def}}{=} \{0, 1, \ldots, d-1\} \tag{A.13}$$

Consideremos a função de Liapunov

$$V(k) \stackrel{\text{def}}{=} \max_{i \in \mathbf{m},\, l \in \mathbf{d}} \left\{ u_i^{-1} \, \|x_i(k)\|_\infty , \quad u_i^{-1} \, \|x_i(k-l)\|_\infty \right\} \tag{A.14}$$

Da equação (A.7) temos que,

$$\begin{aligned}
V(k+1) &= \max_{i \in \mathbf{m},\, l \in (\mathbf{d-1})} \left\{ u_i^{-1} \left\| \sum_{j=1}^{m} \overline{H}_{ij}(k) \, \Upsilon_{ij}(x_{j(k),k}) \right\|_\infty , \quad u_i^{-1} \, \|x_i(k-l)\|_\infty \right\} \\
&\leq \max_{i \in \mathbf{m},\, l \in \mathbf{d}} \left\{ \sum_{j=1}^{m} u_j \, u_i^{-1} \left\| \overline{H}_{ij}(k) \right\|_\infty u_j^{-1} \, \| \Upsilon_{ij}(x_{j(k),k}) \|_\infty , \right. \\
&\qquad \left. u_i^{-1} \, \|x_i(k-l)\|_\infty \right\} \\
&\leq \max_{i \in \mathbf{m},\, l \in \mathbf{d}} \left\{ \max_{i \in \mathbf{m}} \left\{ u_i^{-1} \sum_{j=1}^{m} u_j \left\| \overline{H}_{ij}(k) \right\|_\infty \right\} \right. \\
&\qquad \left. \times \max_{j \in \mathbf{m}} \left\{ u_j^{-1} \, \| \Upsilon_{ij}(x_{j(k),k}) \|_\infty \right\}, \quad u_i^{-1} \, \|x_i(k-l)\|_\infty \right\} \\
&\leq \max_{i \in \mathbf{m},\, l \in \mathbf{d}} \left\{ \max_{j \in \mathbf{m}} \left\{ u_j^{-1} \, \| \Upsilon_{ij}(x_{j(k),k}) \|_\infty \right\}, \quad u_i^{-1} \, \|x_i(k-l)\|_\infty \right\} \\
&\leq \max_{i \in \mathbf{m},\, l \in \mathbf{d}} \left\{ u_i^{-1} \, \| x_i(k) \|_\infty , \quad u_i^{-1} \, \|x_i(k-l)\|_\infty \right\} \quad = \quad V(k),
\end{aligned}$$

i.e.,

$$V(k+1) \leq V(k). \tag{A.15}$$

Da equação de estado (A.7) e a Hipótese 2.2 que estabelece o contexto parcialmente assíncrono, se conclui que a função de Liapunov $V(k)$ permanece constante no máximo d unidades de tempo e posteriormente deve decrescer, por (A.10).

Desta forma, a função de Liapunov $V(k)$ satisfaz as condições de:

- ser uma função não crescente; e
- necessariamente decrescer a cada d iterações;

consequentemente, o teorema de Liapunov ('Proposition 1.1' Bertsekas e Tsitsiklis, 1989b) estabelece a convergência do método bloco iterativo assíncrono (A.3) para a solução nula ($x^* = 0$). ■

Para facilitar a utilização do Teorema A.1, se apresenta a seguir um Lema demonstrado em Kaszkurewicz, Bhaya e Šiljak (1990), que fornece várias condições equivalentes a (A.10).

LEMA A2
Dada uma matriz não negativa $H = (h_{ij})$, *as afirmações que seguem são equivalentes:*

1. *Existe uma matriz diagonal e positiva* U *tal que* $\|U^{-1}HU\|_\infty < 1$.
2. *O raio espectral* $\rho(H)$, *é estritamente menor que 1.*
3. $I - H$ *é quase dominante.*
4. *Existe uma matriz diagonal e positiva* P *tal que* $(H^TPH - P)$ *é negativa definida.*
5. $I - H$ *é uma 'M−matrix' não singular.*

■

Apêndice B

Lema Assíncrono de Comparação

Demonstra-se a seguir uma versão variante no tempo do conhecido Lema de Bellman que segue a versão invariante no tempo de Grujić e Šiljak (1973), originalmente derivado em Bhaya et al. (1991).

LEMA B1 (LEMA DE COMPARAÇÃO)
Seja $z(k;\, k_0, z_0) \in \mathbb{R}^b$, $\forall k$ *uma solução da inequação vetorial a diferenças:*

$$z(k+1) \;\leq\; H(k)\, z(k), \tag{B.1}$$

onde $\forall k, \quad H(k) \in \mathbb{R}^{b\times b}$ *e a condição inicial é dada por* $\quad z_0 \;=\; z(k_0;\, k_0, z_0)$.

Seja $y(k;\, k_0, y_0)$ *uma solução da equação vetorial de comparação a diferenças associada*

$$y(k+1) \;=\; H(k)\, y(k) \tag{B.2}$$

com condição inicial $\quad y_0 \;=\; y(k_0;\, k_0, y_0)$.

Se $z_0 = y_0 \quad e \quad \forall k \geq k_0, \quad H(k) = \Big(h_{ij}(k)\Big) \;\geq\; 0, \quad i.e.$

$$\forall k \geq k_0, \qquad h_{ij}(k) \;\geq\; 0, \tag{B.3}$$

então,

$$\forall k \geq k_0, \qquad z(k;\, k_0, z_0) \;\leq\; y(k;\, k_0, z_0). \tag{B.4}$$

Prova por indução.
Supondo:

$$\forall k \in \{k_0, k_0+1, \ldots, m\}, \qquad z(k;\, k_0, z_0) \;\leq\; y(k;\, k_0, z_0) \tag{B.5}$$

e multiplicando os dois membros da inequação (B.5) pela matriz não negativa $H(k)$ se tem, por (B.1) e (B.2):

$$z(k+1;\, k_0, z_0) \;\geq\; y(k+1;\, k_0, z_0). \tag{B.6}$$

■

A prova acima se baseia em que, se $v^T = (v_1, \ldots, v_b)$ e $w^T = (w_1, \ldots, w_b)$ são dois vetores em $\mathbb{R}^b$ que satisfazem a relação

$$\forall i \in \{1, \ldots, b\}, \qquad v_i \leq w_i, \tag{B.7}$$

que denotamos como $v \leq w$, e se $A \geq 0$ é uma matriz não negativa, então

$$Av \leq Aw. \tag{B.8}$$

Para concluir, derivamos o Lema 2.2, no contexto da Seção 2.1, que estabelece quanto segue:

LEMA B2 (LEMA ASSÍNCRONO DE COMPARAÇÃO)
Sob as hipóteses 2.1 e 2.2, dada uma matriz não negativa $H = (h_{ij}) \in \mathbb{R}^{m \times m}$ $(h_{ij} \geq 0)$ *e um vetor de dimensão m variante no tempo e não negativo* $z(k) \geq 0$, *satisfazendo a inequação:*

$$z_i(k+1) \quad \leq \quad \sum_{j=1}^{m} h_{ij} \; z_j\Big(d_j^i(k)\Big), \qquad \forall\, i \in \{1, \ldots, m\} \tag{B.9}$$

então, $\rho(H) < 1$ *é uma condição suficiente para que* $z(k)$ *tenda exponencialmente a zero quando* $k \to \infty$, *i.e.*

$$\lim_{k \to \infty} z(k) = 0 \qquad se \qquad \rho(H) < 1$$

Prova.

Pela Hipótese 2.2 (assincronismo parcial) temos que

$$z_j(d_j^i(k)) \; \in \; \Big\{ z_j(k),\, z_j(k-1),\, \ldots,\, z_j(k-d) \Big\}, \qquad \forall i, j \in \{1, \ldots, m\} \tag{B.10}$$

portanto, seguindo a metodologia do Apêndice A, definimos o vetor de estado $Z \in \mathbb{R}^{m(d+1)}$ como:

$$Z(k) \; = \; \begin{bmatrix} Z_1(k) \\ \vdots \\ Z_m(k) \end{bmatrix} \qquad \text{onde} \quad Z_j(k) \; \stackrel{\text{def}}{=} \; \begin{bmatrix} z_j(k) \\ z_j(k-1) \\ \vdots \\ z_j(k-d) \end{bmatrix} \in \mathbb{R}^{d+1} \tag{B.11}$$

desta forma, por (B.9), temos que

$$Z(k+1) \quad \leq \quad \mathcal{H}(k)\; Z(k) \qquad \forall k \in \mathbb{N} \tag{B.12}$$

onde $\mathcal{H}(k) \in \mathbb{R}^{(d+1)\times(d+1)}$ é uma matriz não negativa, variante no tempo, cujos elementos são: os elementos não negativos de H, uns e zeros, conforme discussão do Apêndice A.

Seja $Y(k)$ uma solução da equação vetorial de comparação a diferenças associada, i.e.

$$Y(k+1) \quad = \quad \mathcal{H}(k)\ Y(k) \qquad\qquad \forall k \in \mathbb{N} \tag{B.13}$$

então, pelo Teorema A1 (originalmente derivado em Kaszkurewicz et al., 1990) e sob a Hipótese 2.1, $\rho(H) < 1$ é uma condição suficiente para determinar a convergência exponencial de $Y(k)$ para a solução nula, i.e.

$$\lim_{k\to\infty} Y(k) \quad = \quad 0. \tag{B.14}$$

Finalmente, considerando que $\mathcal{H}(k)$ é não negativa e que $Z(0) = Y(0)$, (mesma condição inicial), o Lema B1 estebelece que:

$$\lim_{k\to\infty} z(k) = 0 \qquad \text{se} \qquad \rho(H) < 1$$

■

Referências Bibliográficas

[1] Aboytes F. e Sasson A.M. (1971). A Power System Decomposition Algorithm. *Proc. of the IEEE Power Industry Computer Application Conference,* pp. 448-452.

[2] Aragon C.R., Johnson D.S. e McGeoch L.A. (1984). Optimization by simulated annealing: an experimental evaluation. *Workshop on Statistical Physics in Engineering and Biology.*

[3] Arjomandi E., Fischer M.J. e Lynch N.A. (1983). Efficiency of Synchronous Versus Asynchronous Distributed Systems. *ACM Journal,* Vol. 30, No. 3, pp. 449-456.

[4] Asarin E.A., Krasnoselskii M.A., Kozyakin V.S. e Kuznetsov N.A. (1990). Stability Analysis of Desynchronized Systems. *Proc. 11th IFAC Conference,* Vol. 2.

[5] Barán B. e Corrêa R.C. (1990). Algoritmo Tipo Gradiente: uma proposta para sistemas computacionais paralelos parcialmente assíncronos. *XVI Conferência Latinoamericana de Informática.* Assunção, Paraguai.

[6] Barán B. (1991). Algoritmos Combinados: Uma aplicação para Fluxo de Potência Elétrica. Relatório Técnico da Universidade Federal de Rio de Janeiro. Brasil.

[7] Barán B., Kaszkurewicz E. e Bhaya A. (1993). Distributed Asynchronous Team Algorithms: Application to the Load Flow Problem. *XIX Conferência Latinoamericana de Informática.* Bs. Aires, Argentina.

[8] Barlow R.H. e Evans D.J. (1982). Parallel Algorithms for the Iterative Solution to Linear Systems. *The Computer Journal,* Vol. 25, No. 1, pp. 56-60.

[9] Baudet G.M. (1978). Asynchronous iterative methods for multiprocessors, *J. ACM, 25* (2), pp. 226-244.

[10] Beidas B.F. e Papavassilopoulos G.P. (1991). Convergence Analysis of Asynchronous Iterations with Stochastic Delays. *Proc. of the 30th Conference on Decision and Control.* Brighton, Inglaterra.

[11] Bertsekas D.P. (1983). Distributed Asynchronous Computation of Fixed Points. *Mathematical Programming 27, North-Holland,* pp. 107-120.

[12] Bertsekas D.P. e Tsitsiklis J.N. (1988). A survey of some aspects of parallel and distributed iterative algorithms. *Technical Report LIDS-P-1835.* Laboratory for Information and Decision Systems, MIT. Cambridge, MA.

[13] Bertsekas D.P. e Tsitsiklis J.N. (1989a). Convergence Rate and Termination of Asynchronous Iterative Algorithms. *Proceding of the International Confernce on Supercomputing.* Creta, Grecia.

[14] Bertsekas D.P. e Tsitsiklis J.N. (1989b). *Parallel and distributed computation. Numerical Methods.* Prentice-Hall.

[15] Bhaya A., Kaszkurewicz E. e Mota F. (1991). Asynchronous Block-Iterative Methods for Almost Linear Equations. *Linear Algebra and Its Applications,* Vol. 155, pp. 487-508.

[16] Brucoli M., La Scala M., Torelli F. e Trovato M. (1987). Overlapping decomposition for small disturbance stability analysis of interconnected power networks. *Large Scale Systems*, 13, pp. 115-129.

[17] Calvet J.L. e Titli A. (1989). Overlapping vs. Partitioning in Block-iteration Methods: Application in Large-scale System Theory. *Automatica,* Vol. 25, No. 1, pp. 137-145.

[18] Carré B.A. (1968). Solution of Load Flow by Partitioning into Trees. *IEEE Trans. on PAS*, vol. PAS-88, pp. 1931-1938.

[19] Chazan D. e Miranker W. (1969). Chaotic relaxation. *Linear Algebra Appl. 2,* pp. 199-222.

[20] Decker I.C., Falcão D.M. e Kaszkurewicz E. (1991). Parallel implementation of a power system dynamic simulation methodology using conjugate gradient method. *IEEE Power Industry Computer Application Conference.* Baltimore.

[21] Dijkstra E.W. e Sholten C.S. (1980). Termination detection for diffusing computations. *Inf. Proc. Lett.,* Vol. 11, pp. 1-4.

[22] Dijkstra E.W., Feijen W.H.J. e Van Gasteren A.J.M. (1983). Derivation of a termination algorithm for distributed computations. *Inf. Proc. Lett.,* Vol. 15, pp. 217-219.

[23] Donnelly J.D.P. (1971). Periodic Chaotic Relaxation. *Linear Algebra and its Applications,* Vol. 4, pp. 199-222.

[24] Duff I.S. (1981). *Sparse Matrices and Their Uses*, Academic Press, New York.

[25] Dunigan T.H. (1991). Performance of the Intel iPSC/860 and Ncube 6400 hypercubes. *Parallel Computing,* 17. North-Holland, pp. 1285-1302.

[26] Dusonchet Y.P., Talukdar S.N. e Sinnot H.E. (1971). Load Flows using a combination of Point Jacobi and Newton's Methods, *IEEE Transactions on Power Apparatus and Systems,* PAS-90, pp. 941-949.

[27] El Tarazi M.N. (1982). Some convergence results for asynchronous algorithms, *Numer. Math., 39,* pp. 325-340.

[28] Falcão D.M. e Barán B. (1992). Team Algorithms. Uma aplicação para Fluxo de Potência Elétrica. XVIII Conferência Latinoamericana de Informática. Las Palmas de Gran Canárias, Espanha.

[29] Fletcher R. (1987). *Practical Methods of Optimization.* John Wiley. Second Edition.

[30] Fox M.S. (1981). An Organizational View of Distributed Systems. *IEEE Trans. on Systems, Man and Cybernetics,* Vol. SMC-11, No. 1, pp. 70-80.

[31] Francez N. (1980). Distributed termination. *ACM Trans. Prog. Lang. and Sys.,* Vol. 2, pp. 42-55.

[32] Francez N. e Rodeh M. (1982). Achieving Distributed Termination without Freezing. *IEEE Trans. on Sw. Eng.,* SE-8, No. 3, pp. 287-292.

[33] Frommer A. (1991). Orders of Convergence for Superlinearly Convergent Chaotic Iterations. *Computing* 47, pp. 97-101. *Springer-Verlag.*

[34] Golub G.H. e VanLoan C.F. (1989). *Matrix Computations.* Johns Hopkins Univ. Press, Baltimore.

[35] Grujić L.T. e Šiljak D.D. (1973). On stability of discrete composite systems, *IEEE Trans. Automatic Control AC-18* (5), pp. 522-524.

[36] Hageman L.A. e Young D.M. (1981). *Applied Iterative Methods*, New York, Academic Press.

[37] Heath M.T., Geist G.A. e Drake J.B. (1991). Early Experience with the Intel iPSC/860 at Oak Ridge National Laboratory. *International Journal of Supercomputer Applications,* Vol. 5, No. 2, pp. 10-26.

[38] Hodzic M. e Šiljak D.D. (1986). Decentralized Estimation and Control with Overlapping Information Sets. *IEEE Trans. on Automatic Control,* Vol. AC-31, No. 1, pp. 83-86.

[39] Horn R.A. e Johnson C.R. (1988). *Matrix Analysis*, Cambridge University Press.

[40] IEEE Power Systems Enginering Committee Report (1992). Parallel Processing in Power Systems Computation. *Trans. on Power Systems,* Vol. 7, No. 2, pp. 629-637.

[41] Ikeda M. e Šiljak D.D. (1980). Overlapping decomposition, expansions and contractions of dynamic systems. *Large Scale System 1*, North-Holland Publishing Co., pp. 29-38.

[42] Ikeda M. e Šiljak D.D. (1981). Generalized Decompositions of Dynamic Systems and Vector Liapunov Functions. *IEEE Trans. on Automatic Control*, Vol. AC-26, No. 5, pp. 1118-1125.

[43] Ikeda M., Šiljak D.D. e White D.E. (1984). An Inclusion Principle for Dynamic Systems. *IEEE Trans. on Automatic Control*, Vol. AC-29, No. 3, pp. 244-249.

[44] Ikeda M. e Šiljak D.D. (1987). Stability of Reduced-order models via vector Liapunov functions. *1987 American Control Conference.* Jun 10-12.

[45] Ilić-Spong M., Katz N., Dai H. e Zaborszky J. (1984). Block Diagonal Dominance for Systems of Nonlinear Equations with Application to Load Flow Calculations in Power Systems. *Mathematical Modelling,* Vol. 5, pp. 275-297.

[46] Irving M.R. e Sterling M.J.H. (1990). Optimal Network Tearing Using Simulated Annealing. *IEE Proceedings,* vol. 137, no. 1, pp. 69-72.

[47] Johnson B.K. (1977). Extraneous and False Load Flow Solutions. *IEEE Trans. on PAS*, Vol. PAS-96, No. 2, pp. 524-534.

[48] Kaszkurewicz E., Bhaya A. e Šiljak D.D. (1990). On the convergence of parallel asynchronous block-iterative computations. *Linear Algebra Appl., 131,* pp. 139-160.

[49] Kirkpatrick S., Gelatt C.D. e Vecchi M.P. (1983). Optimization by simulated annealing. *Science,* 220 (4598), pp. 671-680.

[50] Kleptsyn A.F., Krasnoselskii M.A., Kuznetsov N.A. e Kozyakin V.S. (1984). Desynchronization of Linear Systems. *Math. and Computers in Simulation XXVI,* pp. 423-431. *North-Holland.*

[51] Korsak A.J. (1972). On the Question of Uniqueness of Stable Load Flow Solutions. *IEEE Trans. on PAS*, Vol. 91, pp. 1093-1100.

[52] Kung H.T. (1976). Synchronous and Asynchronous Parallel Algorithms for Multiprocessors. *Algorithm and Complexity - New Directions and Recent Results.* Ed. J.Traub, pp. 153-200. Academic Press, N.Y.

[53] Lee J.G., Vogt W.G. e Mickle M.H. (1979). Optimal Decomposition of Large Scale Networks. *IEEE Trans. on Systems, Man and Cybernetics,* vol. SMC-9, no. 7, pp. 369-375.

[54] Levenberg K. (1944). A Method for the solution of certain nonlinear problems in least squares. *Q. Appl. Math.*, Vol. 2, pp. 164-168.

[55] Marquardt D.W. (1963). An algorithm for least squares estimation of nonlinear parameters. *J. Soc. Ind. Appl. Math.*, Vol. 11, pp. 431-441.

[56] Marschak J. e Radner R. (1972). *Economic Theory of Teams.* New Haven, CT: Yale University Press.

[57] Mehrotra R. e Talukdar S.N. (1983). Task Schedulling on Multiprocessors for Power Systems Problems, *IEEE Transactions on Power Apparatus and Systems,* PAS-102, pp. 3590-3597.

[58] Mickle M.H., Vogt W.G. e Colclaser R.G. (1977). Parallel Processing and Optimal Network Decomposition Applied to Load Flow Analysis and Related Problems. *Special Report of the Electrical Power Research Institute,* EPRI EL-566-SR, pp. 171-182.

[59] Miellou J.C. (1974). Itérations chaotiques à retards, *C. R. Acad. Sc. Paris, T. 278,* Ser. A., pp. 957-960.

[60] Mitra D., Romeo F. e Sangiovanni-Vicentelli A. (1986). Convergence and finite-time behavior of simulated annealing. *Advances in Applied Probability,* 18 (3), pp. 747-771.

[61] Mitra, D. (1987). Asynchronous relaxations for the numerical solution of differential equations by parallel processors, *SIAM J. Sci. Stat. Comput., 8* (1), pp. 43-58.

[62] Monticelli A. (1983). *Fluxo de Carga em Redes de Energia Elétrica,* CEPEL. Editora Edgar Blücher Ltda.

[63] Morozowski M. (1981). *Matrizes Esparsas em Redes de Potência: Técnicas de Operação,* Livros Técnicos e Científicos Editora S.A. Eletrobrás. Rio de Janeiro.

[64] Ogbuobiri E., Tinney W.F., e Walker J.W. (1970). Sparsity Oriented Decomposition for Gaussian Elimination on Matrices. *IEEE Trans. on PAS,* vol. PAS-89, no.1, pp. 141-150.

[65] Ohta Y. e Šiljak D.D. (1985). Overlapping Block Diagonal Dominance and Existance of Liapunov Functions. *Journal of Mathematical Analysis and Applications,* 112, pp. 396-410.

[66] Ortega J.M. e Rheinboldt W.C. (1968). Local and global convergence of generalized linear iterations. Symposium on Numerical Solution of Nonlinear Problems, SIAM, Filadelfia, Pensilvania.

[67] Ortega J.M. e Rheinboldt W.C. (1970). *Iterative solution of nonlinear equations in several variables.* Academic Press, New York.

[68] Ortega J.M. (1988). *Introduction to parallel and vector solution of linear systems.* Plenum Press, New York.

[69] Penny N.H. (1986). Blackboard Systems: The blackboard model of problem solving and the evolution of blackboard architectures. *The AI Magazine.*

[70] Powell M.J.D. (1970). A hybrid method for nonlinear equations. *Numerical Methods for Nonlinear Equations,* Gordon & Breach, Londres.

[71] Ramesh V.C., Quadrel R., de Souza P. e Talukdar S.N. (1991). Asynchronous Teams. An Organizational Model for Distributed Problem Solving, *1991 Summer National Meeting, American Institute of Chemical Engineers,* Pittsburgh.

[72] Robert F. (1976). Contraction en norme vectorielle: convergence d'itérations chaotiques pour des equations non linéaires de point fixe à plusieurs variables, *Linear Algebra Appl., 13,* pp. 19-35.

[73] Rosenfeld J.L. (1967). A case study in Programming for parallel processors. IBM T.J. Watson Research Center Report No. RC-64. Publicado em *Comm. ACM,* Vol. 12, No. 12, pp. 645-655, 1969.

[74] Rosenfeld J.L. e Driscoll G.C. (1968). Solution of the Dirichlet Problem on a Simulated Parallel Processing System. *Proc. IFIP Congress 68,* pp. 499-507. North Holland. Amsterdam.

[75] Sasson A.M. (1970). Decomposition Technique Applied to the Nonlinear Programming Load Flow Method. em IEEE Trans. on PAS, vol. PAS-89, no. 1, pp. 78-82.

[76] Sezer M.E. e Šiljak D.D. (1984). Nested epsilon decompositions of complex systems. *IFAC* 9^{th} *World Congress,* Budapest, Hungria.

[77] Sezer M.E. e Šiljak D.D. (1986). Nested epsilon decompositions and clustering of complex systems. *Automatica,* vol. 22, no. 3, pp. 69-72.

[78] Sezer M.E. e Šiljak D.D. (1991). Nested epsilon decompositions of linear systems: Weakly coupled and overlapping blocks. *SIAM Journal of Matrix Analysis and Applications,* 12, pp. 521-533.

[79] Šiljak D.D. (1981). Complex Dynamic Systems: Stability, Control and Reliability. *4th. Summer Session on Control Systems Theory and Applications.* Arab School of Science and Tecnology. Bloudan - Siria.

[80] Souza P.S. e Talukdar S.N. (1991). Genetic Algorithms in Asynchronous Teams. *Proceedings of the Fouth International Conference on Genetic Algorithms,* pp. 392-397. California.

[81] Stoer J. e Witzgall C. (1962). Transformations by diagonal matrices in normed spaces. *Numerische Mathematik,* Vol. 4, pp. 158-171.

[82] Stone H.S. (1987). *High-Performance Computer Architecture.* Addison-Wesley Publishing Company.

[83] Stott B. (1972). Decoupled Newton Load Flow. *IEEE Trans. on PAS,* Vol. 91, pp. 1955-1959.

[84] Stott B. e Alsaç O. (1974). Fast Decoupled Load Flow. *IEEE Trans. on Power Apparatus and Systems,* Vol. PAS-93, pp. 859-869.

[85] Stott B. (1974). Review of load-flow calculation methods. *Proc. of the IEEE, 62,* pp. 916-929.

[86] Strang G. (1980). *Linear Algebra and its Applications,* Second Edition. Academic Press Inc. Florida.

[87] Talukdar S.N., Pyo S.S. e Giras T.C. (1982). Asynchronous Procedures for parallel Processing. Relatório Técnico DRC-18-54-82, Carnegie-Mellon University, Pittsburgh, Pennsylvania. Apresentado em PICA-83.

[88] Talukdar S.N., Pyo S.S. e Mehrotra R. (1983). Designing Algorithms and Assignments for Distributed Processing, *Electric Power Reserch Institute. Final Report.* Carnegie-Mellon University, Pittsburgh, Pennsylvania.

[89] Talukdar S.N. e Cardozo E. (1985). Artificial Intelligence Techniques for Power System Operations. Some Demonstration and an assessment of Opportunities. *EPRI Report No. 1999-7.*

[90] Talukdar S.N., Cardozo E., Leao L., Banares R, e Joobbani R. (1986). A System For Distributed Problem Solving. Relatório Técnico EDRC-05-05-86. Carnegie-Mellon University, Pittsburgh, Pennsylvania.

[91] Talukdar S.N., Ramesh V.C. e Nixon J.C. (1991). A Distributed System of Control Specialist for Real-Time Operations. *Proceeding of the* 3^{rd} *Symposium on Expert Systems Applications to Power Systems.* Japan.

[92] Talukdar S.N., Ramesh V.C., Quadrel R. e Christie R. (1992). Multiagent Organizations for Real-Time Operations. *Proc. of the IEEE,* Vol. 80, No. 5, pp. 765-778.

[93] Talukdar S.N. e Souza P.S. (1992). Scale Efficient Organizations. To be published by the Engineering Design Research Center of Carnegie Mellon University. Pittsburgh, Pennsylvania.

[94] Tinney W.F. e Hart C.E. (1967). Power Flow Solution by Newton's Method. *IEEE Trans. on PAS*, Vol. PAS-86, No. 11, pp. 1449-1460.

[95] Undrill J.M. e Harpp H.H. (1971). Automatic Sectionalization of Power System Networks for Network Solutions. *IEEE Trans. on PAS,* vol. PAS-90, no. 1, pp. 46-53.

[96] Üresin A. e Dubois M. (1989). Sufficient Conditions for the Convergence of Asynchronous Iterations. *Parallel Computing,* 10, pp. 83-92.

[97] Üresin A. e Dubois M. (1990). Parallel Asynchronous Algorithms for Discrete Data. *ACM Journal,* Vol. 37, No.3, pp. 588-606.

[98] Vale M.H.M., Falcão D.M. e Kaszkurewicz E. (1992). Electrical Power Network Decomposition for Parallel Computations. *IEEE Interanational Symposium on Circuits and Systems - ISCAS 92*, San Diego, California.

[99] Vanelli A. (1983). Solution Techniques for 0-1 Indefinite Quadratic Programming Problems with Applications to Decomposition. Ph.D. Dissertation. University of Waterloo. Canada.

[100] Varga, R. (1962). *Matrix Iterative Analysis.* Prentice Hall, Englewood Cliffs, N.J.

[101] Westerberg A.W. e Director S.W. (1978). A modified least square algorithm for solving sparse $n \times n$ sets of nonlinear equations. *Computers and Chemical Engineering,* Vol. 2, pp. 77-81.

[102] White R.E. (1986). Parallel algorithms for nonlinear problems, *SIAM J. Alg. Disc. Meth., 7* (1), pp. 137-149.

[103] Williams R.J. e Peng J. (1991). Function Optimization using Connectionist Reinforcement Learning Algorithms. *Connection Science,* Vol. 3, No. 3, pp. 241-268.

[104] Ward J.B. e Hale H.W. (1956). Digital computer solution of power-flow problem. *AIEE Trans. on PAS*, Vol. 75, pp. 398-401.

[105] Wu F.F. (1977). Theoretical Study of the Convergence of the Fast Decoupled Load Flow. em IEEE Trans. on PAS, Vol. 96, pp. 268-275.

[106] Young D.M. (1971). *Iterative Solutions of Large Linear Systems.* Academic Press, New York.

[107] Zečević A.I. e Šiljak D.D. (1992). A Block-Parallel Newton Method via Overlapping Epsilon Decompositions. *1992 American Control Conference.* Chicago, Illinois.

[108] Zielke G. (1988). Some Remarks on Matrix Norms, Condition Numbers, and Error Estimates for Linear Equations. *Linear Algebra and Its Applications,* Vol. 110, pp. 29-41.