UMA ABORDAGEM PARALELA DO MÉTODO BRANCH-AND-BOUND INTERVALAR PARA OTIMIZAÇÃO GLOBAL

Henrique Limaverde Cabral de Lima

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 MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Nelson Maculan Filho, D.Sc.

Prof. Adilson Elias Xavier, D.Sc.

Profa. Fernanda Raupp, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2001

DE LIMA, HENRIQUE LIMAVERDE CABRAL

Uma Abordagem Paralela do Método Branch-and-Bound Intervalar Para Otimização Global [Rio de Janeiro] 2001

XI, 69 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2001)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1. Branch-and-Bound Paralelo.

2. Aritmética Intervalar.

3. Otimização Global.

I. COPPE/UFRJ II. Título (série).

À minha família e meus amigos que,
estando perto, compartilharam comigo
o que eles têm de mais importante: suas vidas
e que, mesmo estando longe,
nunca realmente deixaram de estar perto.

Agradecimentos

Em primeiro lugar, quero me desculpar de antemão pelas injustiças que estou a ponto de cometer já que conheci tantas pessoas no percurso do meu mestrado que seria inviável mencionar cada uma mesmo acreditando que todas elas, direta ou indiretamente, com mais ou menos intensidade, contribuíram para tornar esse trabalho possível quanto tornaram a nova vida no Rio bem mais agradável.

Gostaria de agradecer à grande pessoa e pesquisador que é o Prof. Nelson Maculan, de quem admiro o respeito pelo indivíduo e o senso de humor, pela sua orientação.

Ao primo Carlile, com quem, por mais de 8 meses, trabalhei bem de perto, a quem ensinei e de quem aprendi muita coisa, dentre elas que a Ciências da Computação e Matemática podem ser peças do mesmo quebra-cabeças mas não são necessariamente vizinhas.

Aos companheiros da República Dragão do Mar Elder, Leonardo e Tibérius com quem convivi desde o começo do mestrado e por quem hoje tenho um respeito e amizade muito fortes. Aprendi muito sobre convivência, sobre eles e sobre mim mesmo na RDDM e isso não tem preço. Quero estender o agradecimento aos outros que também passaram pela república (Iuri, Prata, Gabriel, Rafael, Otávio e Fábio). E obviamente a toda a Bancada Cearense que por ser tão numerosa me impede de enumerar todos.

Aos companheiros com quem tive um convívio mais freqüente no PESC e no Labotim (Adriana, Alessandréia, Alexandre, Álvaro, Amir, Ana Flávia, Ana Lúcia, Anatércia, André Barros, André Brito, André (Luke), Carlos André, Denise, Douglas, Flávio, Jorge, Jurandir, Leonardo Lima, Lucídio, Mara, Marcone, Michele, Moisés, Passini, José (Pepe), Rosa, Shane, Talita) e tantos outros meu muito obrigado pela amizade. Desejo ainda incluir como meus amigos, o Prof. Adilson Xavier e o Prof. Amir Coelho, com quem desejaria ter podido conversar longamente num bar tomando cerveja ao invés de nos (nem sempre apropriados) corredores da universidade.

À amiga Maria de Fátima Cruz Marques, pois, apesar de tê-la conhecido já no final do mestrado, mas que logo percebi a pessoa maravilhosa que ela é.

Às secretárias e funcionários do PESC sem o trabalho dos quais as coisas não funcionariam no departamento. Em particular, gostaria de agradecer ao doce de pessoa que é a D. Gercina, nossa telefonista.

À minha família, pelo amor, apoio e carinho incondicionais que estão sempre a

oferecer e que espero sempre poder retribuir. Em particular à minha mãe que sente saudades do filho distante desde que soube que iria me mudar.

Ao governo brasileiro que, através do CNPq, investiu na minha formação.

A todos vocês o meu sincero muito obrigado!

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

UMA ABORDAGEM PARALELA DO MÉTODO BRANCH-AND-BOUND INTERVALAR PARA OTIMIZAÇÃO GLOBAL

Henrique Limaverde Cabral de Lima

Setembro/2001

Orientador: Nelson Maculan Filho

Programa: Engenharia de Sistemas e Computação

A motivação desse trabalho veio de um problema de otimização global originário da Bioquímica: Problema de Conformação Molecular (PCM). Primeiramente introduzimos o problema e os assuntos subjacentes ao método empregado: *branch-and-bound* intervalar. Em seguida, passamos a nos preocupar com os problemas decorrentes da paralelização de tal método. Apresentamos algumas técnicas para tornar o método mais eficiente e possivelmente capaz de tratar instâncias mais interessantes de um ponto de vista prático.

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

A PARALLEL APPROACH OF INTERVAL BRANCH-AND-BOUND METHOD FOR GLOBAL OPTIMIZATION

Henrique Limaverde Cabral de Lima

September/2001

Advisor: Nelson Maculan Filho

Department: Computing and Systems Engineering

The motivation of this work came from a Global Optimization Problem originated in Biochemestry: Molecular Conformation Problem (MCP). Firstly, we introduce the problem and the subjects subjacent to the employed method: intervalar *branch-and-bound*. After that, we address the issues arising from paralelization of that method. We present some techniques that can be applied in order to make the method more efficient so it can possibly cope with more interesting instances from a practical point of view.

Conteúdo

Lista de Figuras

Capítulo 1

Problema de Conformação Molecular

1.1 Motivação

O problema que motivou este trabalho se chama Problema de Conformação Molecular (PCM) que consiste em determinar a estrutura tridimensional de uma molécula a partir da sua estrutura primária. Essa estrutura depende apenas da seqüência de átomos e das interações covalentes e não-covalentes entre eles. Conhecer a estrutura tridimensonal é importante pois dela se infere as propriedades fisiológicas da molécula.

As técnicas tradicionais da Bioquímica para determinar a estrutura de uma molécula são a Cristalografia de Raios-X (para proteínas apenas), Ressonância Nuclear Magnética e Espectro de Dicroísmo Circular. Ambos utilizam processos químicos e físicos para determinar essa estrutura e por esse motivo estão sujeitos a limitações físicas, como por exemplo: falhas, incertezas, serem inadequados para algumas moléculas, o processo é demorado e requer o preparo prévio de material. Tudo isso torna essas técnicas custosas para moléculas de grande porte.

A abordagem matemática do problema, por outro lado, se propõe a identificar a estrutura de uma molécula apenas por meios que não são influenciados por nenhuma limitação física. Além disso, o método matemático se aplica a qualquer molécula de qualquer tamanho, podendo gerar respostas que estejam tão próximas da realidade quanto se deseje (e o modelo permita). O único requerimento para isto é tão so-

mente a implementação de um programa que calcule a energia potencial da molécula em questão (bem como suas derivadas, se possível) como vamos ver mais adiante. Apesar da formulação matemática não envolver questões muito complexas, a resolução do modelo consiste num desafio pois o modelo é não-linear não-convexo, e, o número de mínimos locais do modelo cresce exponencialmente com o tamanho das instâncias.

1.2 Formulação do Problema

Resultados da Bioquímica indicam que a estrutura tridimensional de uma molécula no seu estado nativo geralmente está associado à menor quantidade de energia potencial dessa molécula. Essa energia potencial é calculada de acordo com as forças físicas que atuam entre os átomos dessa molécula. Dessa forma, podemos expressar a quantidade de energia em função da posição dos átomos em relação a um referencial fixo.

A energia potencial é resultante das interações entre os átomos de uma molécula. As interações (de natureza química) podem ser: covalentes (associadas às ligações entre os átomos vizinhos) ou não-covalentes (associadas às interações entre átomos não ligados covalentemente). As interações geralmente utlizadas podem ser modeladas da seguinte maneira:

- Ligações covalentes: mantêm átomos vizinhos a uma distância constante. Pode ser modelada como:

$$E_b = k_b(r - r_0)^2,$$

  onde k_b é a constante de energia da ligação, r é o comprimento da ligação e r_0 é a constante de equilíbrio da ligação.

- Ângulo de ligação (covalente): o ângulo de ligação θ entre os átomos A, B e C, ligados covalentemente, é definido como o ângulo entre as ligações covalentes $\overline{AB}$ e $\overline{BC}$ na figura 1.1. A energia potencial é representada por:

$$E_\theta = k_\theta(\theta - \theta_0)^2,$$

onde k_θ é a constante de energia do ângulo, θ é o ângulo de ligação e θ_0 é a constante de equilíbrio do ângulo.

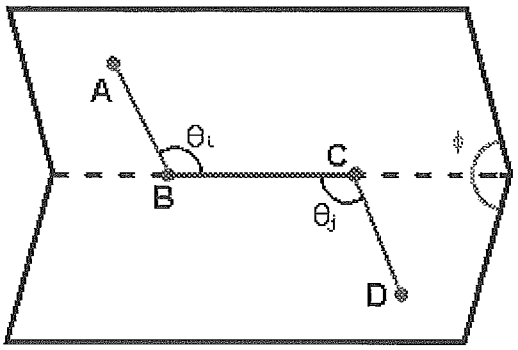

Figura 1.1: Ligações covalentes: ângulos de ligação (θ) e diedrais (ϕ)

- Ângulos diedrais (covalente): Um ângulo diedral (ou ângulo de torsão) ϕ entre quatro átomos A, B, C e D, ligados covalentemente, é definido como o ângulo entre os planos $\overline{ABC}$ e $\overline{BCD}$ representados na figura 1.1. A energia potencial associada é dada por

$$E_\phi = \frac{V_n}{2}[1 + \cos(n\phi - \gamma)],$$

onde ϕ é o ângulo diedral e V_n, n e γ são constantes que dependem dos átomos envolvidos.

- Interações iônicas (não-covalente): ocorrem entre grupos eletricamente carregados e exercem influência importante na estabilização da molécula. Segue a lei de Coulomb e pode ser dada por:

$$E_{el} = \frac{q_i q_j}{D r_{ij}},$$

onde q_i e q_j representam o valor das cargas parciais dos átomos i e j e D é a constante dielétrica do meio (solvente) e r_{ij} é a distância entre eles.

- Pontes de hidrogênio (não-covalente): são decorrentes das polaridades de grupos neutros de átomos. Podem causar efeitos semelhantes às interações iônicas. São calculadas segundo a expressão:

$$E_{hb} = \frac{C_{ij}}{r_{ij}^{12}} - \frac{D_{ij}}{r_{ij}^{10}},$$

onde C_{ij} e D_{ij} são constantes que dependem dos átomos i e j, e r_{ij} é a distância entre eles.

- Interações hidrofóbicas (não-covalentes): associação de grupos ou compostos não-polares entre si, nos sistemas aquosos, por causa da tendência das moléculas de água circundantes de adquirir o seu estado mais estável (desordenado).

$$\varepsilon(r_{ij}) = D - \frac{D-2}{2}\left[(sr_{ij})^2 + 2sr_{ij} + 2\right]e^{-sr_{ij}}, \tag{1.1}$$

onde r_{ij} é a distância entre os átomos i e j, $D = 78$ e $s = 0,3$.

- Interações de van der Waals (não-covalentes): são interações decorrentes da indução de cargas parciais quando os orbitais de elétrons se aproximam. Essas interações decorrem dos efeitos de dispersão e de exclusão de Pauli. São importantes na estabilização da molécula. A expressão é dada por:

$$E_{vw} = \frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}},$$

onde A_{ij} e B_{ij} são constantes que dependem dos átomos i e j, e r_{ij} é a distância entre eles.

A expressão completa da função de energia potencial fica:

$$\begin{aligned}
FEP \quad = \quad & \sum_{\mathbf{r}}(k_b(\mathbf{r} - r_0)^2) \\
& + \sum_{\theta}(k_\theta(\theta - \theta_0)^2) \\
& + \sum_{\omega}(k_\omega(\omega - \omega_0)^2) \\
& + \sum_{\phi}(\frac{V_n}{2}[1 + \cos(n\phi - \gamma)]) \\
& + \sum_{i,j}\left(\frac{q_i q_j}{D\mathbf{r}_{ij}}\right) \\
& + \sum_{i,j}\left(D - \frac{D-2}{2}\left[(sr_{ij})^2 + 2sr_{ij} + 2\right]e^{-sr_{ij}}\right) \\
& + \sum_{i,j}\left(\frac{C_{ij}}{\mathbf{r}_{ij}^{12}} - \frac{D_{ij}}{\mathbf{r}_{ij}^{10}}\right) \\
& + \sum_{i,j}\left(\frac{A_{ij}}{\mathbf{r}_{ij}^{12}} - \frac{B_{ij}}{\mathbf{r}_{ij}^{6}}\right)
\end{aligned}$$

1.3 A Função-objetivo

As duas primeiras moléculas utilizadas eram pequenas, de estrutura conhecida e serviram para verficar o método. Eram elas: pseudoetano (que não existe na natureza) e 123-tricloro, 1-flúor propano mostradas na figura 1.2.

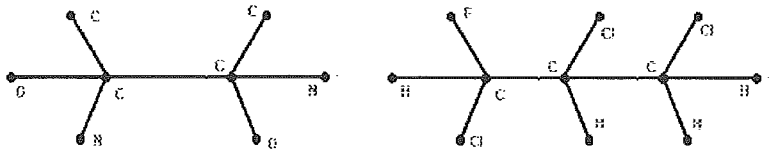

Figura 1.2: Pseudoetano (esq.) e 123-tricloro, 1-flúor propano (dir.)

A intenção inicial era encontrar a estrutura tridimensional de uma molécula real. A molécula de polialanina (figura 1.3) foi escolhida por ser composta de seqüências repetidas de átomos. Isso possibilitaria gerar uma família de exemplos com dimensões tão grandes quanto se desejasse com uma mesma implementação computacional. Mas logo se notou que a avaliação das derivadas da função de energia potencial para esta molécula era deveras complexo. Uma primeira simplificação foi feita usando-se uma molécula fictícia consistindo de apenas uma seqüência de átomos. A essa molécula deu-se o nome de cadeia.

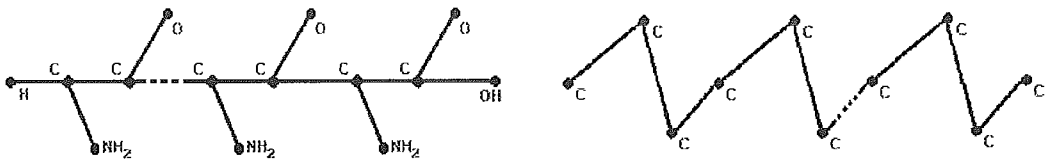

Figura 1.3: Alanina (esq.) e Cadeia (dir.)

Ainda assim, as derivadas estavam muito mais complicadas do que se imaginava. Isso se devia mais pela heterogeneidade dos termos do que pela sua quantidade. Partiu-se então (abrindo mão de aplicar o algoritmo a uma função que modelasse uma molécula) para uma função sintética que foi montada em [53] cujo mínimo é conhecido, se comporta como as funções de energia potencial (número exponencial de mínimos locais) e possui uma expressão bem mais simples (o que permitiu a

avaliação de suas derivadas). Essa função será chamada de CF em referência ao seu autor.

1.4 Implementação

A implementação da função computacional que avaliasse tanto a função matemática da energia potencial quanto suas derivadas primeira e segunda para as moléculas polialanina e cadeia logo se mostrou um trabalho muito complexo. Em primeiro lugar tentou-se usar a facilidade de geração de código-fonte a partir do pacote matemático Maple [9]. Essa opção foi logo descartada pois o código-fonte gerado é muito ineficiente e de difícil compilação. Partiu-se para a implementação da função computacional manualmente (em oposição à forma automática do Maple) da função usando-se o Maple para validar a versão manual. Conseguimos maiores avanços mas nem todas as derivadas funcionavam de forma que fomos obrigados por motivo de tempo a simplificar mais ainda nosso exemplo. Desta forma, terminamos optando pela função CF.

1.5 Organização do Texto

Esse trabalho está dividido nas seguintes partes. O capítulo 2 discorre sobre a Otimização Global, alguns métodos tradicionais da área e o método *branch-and-bound* em particular. O capítulo 3 dá uma introdução aos conceitos básicos da Aritmética Intervalar que será usada no método. O capítulo 4 detalha a implementação do método *branch-and-bound* intervalar e o capítulo 5 acrescenta detalhes da paralelização desse método. O capítulo 6 dá detalhes de implementação. Por fim, o capítulo 7 mostra as conclusões do trabalho.

Capítulo 2

Otimização Global

Como foi visto no capítulo anterior, matematicamente, o PCM consiste em descobrir o menor dentre os mínimos locais da função de energia potencial. Essa função é extremamente complexa o que enquadra o problema numa categoria mais abrangente: a de otimização global.

2.1 Introdução

Há muitos problemas práticos, notadamente na indústria, que podem ser modelados simplesmente usando funções lineares, por exemplo: transporte, distribuição, localização de facilidades, escalonamento de tarefas e de tripulação. Para estes problemas, vários algoritmos computacionais são bastante conhecidos e desenvolvidos.

Mas, não menos freqüentemente, encontra-se problemas intrinsecamente não-lineares. Exemplos estão espalhados em muitos campos de atuação, na física, química, biologia, economia, engenharia, sistemas sociais, e mesmo nas áreas já citadas anteriormente. Para alguns casos de problemas não-lineares, podemos verificar uma característica especial (convexidade) que, por garantir a unicidade da solução mínima, permite certificar-se de que a solução encontrada por determinado algoritmo será a ótima. Essa característica é especial porque define de forma precisa a solução, além de fornecer subsídios para a construção de algoritmos. Além do mais, também são conhecidos vários algoritmos eficientes (uns mais genéricos, outros mais

especializados) que podem ser empregados na resolução desses problemas.

Continuando na direção do mais genérico, existem problemas não-lineares para os quais não se garante a propriedade de convexidade. Parte-se do princípio de que existe mais de um ponto que represente uma solução ótima ou mesmo uma boa solução (próxima à solução ótima). Dependendo da função e da região do conjunto de soluções viáveis (espaço viável) onde a busca deve ser feita, soluções boas tornam-se ótimas e vice-versa. Devido à hipótese de que não há uma estrutura especial deve-se supor, e é o que acontece, a existência de tantas soluções locais quantas se possa imaginar.

Já que os algoritmos clássicos da Programação Não-linear admitem convexidade (unicidade da solução ótima), eles falham quando aplicados a um problema mais geral, pois eles terminam encontrando uma solução ótima local (dependendo da função e do ponto de partida) que eventualmente pode vir a ser a solução global. Mas "*eventualmente*" não é uma característica muito desejável, dessa forma, levando à necessidade de desenvolvimento de novos algoritmos próprios para encontrar a solução ótima global entre as tantas locais existentes.

2.2 Definição

A otimização global é, portanto, a parte da Programação Matemática cuja preocupação reside em determinar o menor dos mínimos locais de um problema. Matematicamente, descreve-se um Problema de Otimização Global (POG) como:

$$\begin{array}{rl} \min \text{global} & f(x) \\ s.a.: & g_i(x) \leq 0, i = 1, ..., p \\ & h_j(x) = 0, j = 1, ..., q \\ & x \in \mathbb{R}^n \\ & f, g_i, h_j : \mathbb{R}^n \to \mathbb{R} \end{array} \tag{2.1}$$

Sem perda de generalidade, estamos assumindo um POG como um problema de minimização. Chamaremos, de agora em diante, solução global ótima de mínimo global (ou mínimo absoluto) e solução ótima local de mínimo local.

2.3 Classificação e Propriedades

De acordo com o espaço de soluções viáveis do problema, podemos classificar POGs em:

- POG Combinatório: a função objetivo é definida sobre um conjunto de soluções que é finito (discreto), porém muito grande.
- POG Irrestrito: a função objetivo é definida em todo o $\mathbb{R}^n$ ou, no máximo com restrições de caixa ($l_i \leq x_i \leq u_i, i = 1, ..., n$). Isso significa que não existem as funções g_i nem h_j ($p = q = 0$).
- POG Restrito: a função objetivo é definida como no caso anterior, só que, agora, aparecem as funções g_i e/ou h_j ($p \neq 0$ ou $q \neq 0$). Essas funções complicam ainda mais o problema, pois tornam o espaço de busca mais complexo.

Teoricamente, todos esses problemas são considerados de difícil solução (NP-Hard [69]). Mas, devido ao passo acelerado em que cresce a capacidade de processamento e de armazenamento dos computadores, instâncias cada vez maiores se tornam tratáveis em um tempo razoável. Juntamente ao desenvolvimento tecnológico, novas técnicas, antes inviáveis, podem agora ser desenvolvidas e aplicadas.

2.4 Métodos

Há várias abordagens para resolver um POG: algumas de caráter mais geral, outras que fazem uso de estruturas conhecidas do problema (exemplo: funções côncavas, Lipschitz, etc). Quanto mais a abordagem tirar proveito da estrutura do problema, mais eficiente poderá ser, apesar de estar abrindo mão da abrangência de sua aplicação.

Alguns métodos apresentam prova de convergência, mas isso não descarta a possibilidade de se melhorar os resultados pela aplicação (algumas vezes necessária)

de métodos de busca local na solução desses métodos. Isso é também motivado pela complexidade exponencial que o problema apresenta.

Segue uma lista com alguns exemplos de estratégias de solução de POGs. Deve-se lembrar que, nem essa lista é completa, nem as estratégias são mutuamente exclusivas. Por motivos de organização, divide-se os métodos em dois grandes grupos: heurísticos e exatos.

2.4.1 Métodos Heurísticos

Os métodos heurísticos não garantem encontrar a solução ótima, não fornecem nenhuma idéia de distância da sua solução para o ótimo e nem sempre possuem prova de convergência. Dentre eles podemos citar:

a) Extensões Globais de Métodos de Busca Local: A idéia é montar um método de dois níveis onde o superior faz uma amostragem aleatória no espaço viável para determinar pontos que serão usados como pontos de partida para os algoritmos de busca local executados no nível inferior (ver [83]).

b) Metaheurísticas: Implementam uma busca local modificada onde, inicialmente, o método aceita mais facilmente que o próximo passo leve a uma solução pior que a atual e, à medida que o algoritmo evolui, o dispositivo que controla a convergência vai impedindo cada vez mais que soluções piores sejam aceitas. Exemplos são: Algoritmos Genéticos e Estratégias de Evolução [27, 33, 76, 60], *Simulated Annealing* [51, 45, 1], Busca Tabu [25, 32, 26].

c) Subestimação Global Convexa Aproximada: Uma heurística que tenta estimar a convexidade total da função objetivo pela amostragem de alguns de seus pontos. Aplicável principalmente a funções suaves (ver [21]).

d) Métodos de Continuação: Consiste em partir de uma versão mais suave (simples) da função original, descobrir seus minimizadores locais e, gradualmente, retornar à função original sempre buscando os novos mínimos de cada função

tomando como pontos iniciais aqueles calculados na função anterior. Mais adequada para funções suaves (ver [22]).

e) Melhoramento Seqüencial de Ótimos Locais: Esses métodos geralmente operam sobre funções auxiliares adaptativamente construídas para assistir a busca de ótimos gradualmente melhores. Algumas técnicas usadas para definição dessas funções auxiliares são: "*tunneling*", "*deflation*" e "*filled functions*" (ver [54]).

2.4.2 Métodos Exatos

Os métodos exatos possuem prova de convergência e garantem a solução ótima.

a) Abordagens Ingênuas: Apesar de simples e de convergirem sob algumas condições, são métodos sem esperança de bons resultados para problemas não triviais (dimensão 3, 4, 5...), por usarem uma estratégia "ingênua". Exemplos: busca aleatória, amostragem de cobertura de espaço (ver [34, 70, 89]).

b) Estratégias de Busca (Enumerativa) Completa: Baseiam-se numa busca enumerativa completa das possíveis soluções. Aplicável a problemas combinatórios ou alguns problemas bem estruturados (e.g. programação côncava) (ver [35]).

c) Métodos de trajetória, homotopia e abordagens relacionadas: Pretendem visitar todos os pontos estacionários da função objetivo f, levando a uma lista de todos os mínimos locais e globais. Em geral, incluem modelos de equações diferenciais, métodos de ponto fixo e algoritmos de pivoteamento (ver [22, 20]).

d) Métodos de Aproximação (Relaxação) Sucessiva: Consistem em substituir o problema original por uma seqüência de problemas relaxados de forma que a seqüência das soluções desses problemas relaxados convirja para a solução do problema original. Para criar os problemas relaxados, várias técnicas podem ser usadas: planos de corte, construções minorantes diversas, otimização em níveis, etc (ver [35]).

e) Algoritmos de Busca (Particionamento) Bayesiana: Esses métodos se baseiam em informação previamente escolhida que permita a descrição estocástica das classes de função modeladas. Ao longo do processo de otimização, os parâmetros que caracterizam a função são dinamicamente ajustados. Geralmente qualquer modelo com dimensão maior que 1 representa uma aproximação das funções do modelo.

A convergência é provada teoricamente, desde que garantido um conjunto de pontos de busca devidamente denso, espalhado por todo o espaço viável. Um desafio no uso desses métodos estocásticos é a escolha e verificação de um modelo estatístico apropriado para o problema em questão.

Esses métodos apresentam uma dificuldade intrínseca na sua implementação de forma rigorosa e eficiente segundo (ver [63, 62]).

f) Algoritmos de Busca Estocástica Adaptativa: Outro tipo de algoritmo que se baseia na amostragem aleatória do espaço de soluções, usa estratégias de busca aleatória e pode incluir também: ajuste de parâmetros, refinamento de soluções, regras estatísticas de parada, etc.

Apresentam probabilidade de convergência 1 e podem ser aplicados tanto a problemas discretos quanto a contínuos (ver [70, 89]).

g) Algoritmos *Branch-and-bound*: Consistem em particionar e avaliar o conjunto de soluções repetidamente, eliminando as partições indesejáveis (enumeração implícita) à medida que o algoritmo caminha. Existem muitas maneiras de fazer o particionamento, a escolha do próximo subproblema e de eliminar subproblemas, o que leva a uma grande diversidade de algoritmos *branch-and-bound* (ver [31, 42, 73, 70, 89, 35, 67]).

Note que esses métodos se baseiam muito em informações sobre a estrutura particular de cada problema. Portanto, o conhecimento prévio de informações tais como constante Lipschitz, derivadas, pode ser crucial para a eficiência do método.

Podem ser aplicados a uma grande variedade de problemas tanto contínuos quanto discretos. Problemas combinatórios lineares podem ser resolvidos

com exatidão usando métodos da Programação Inteira baseados em estratégia *branch-and-bound.*

Mais informação sobre otimização global e seus métodos pode ser encontrada nos seguintes artigos (ver [29, 71, 41, 68, 66]).

Devido à complexidade do PCM, além das abordagens que usam o *Branch-and-bound* puro (*c.f.* [57, 58, 8, 59]), outras métodos têm sido utilizados para tentar resolver esse problema: meta-heurísticas (Simulated Annealing [39, 81], *Tabu-search* [50], Algoritmos Genéticos [7, 28]) e algoritmos híbridos como *Branch-and-bound* e *Tabu-search* em [4].

Neste trabalho, faremos uso de um algoritmo *branch-and-bound* por suas propriedades serem desejáveis para tratar o problema em questão. Assim sendo, passemos a examinar o algoritmo *branch-and-bound* em detalhes.

2.5 Algoritmos *Branch-and-bound*

Apesar dos métodos *branch-and-bound* serem abrangentes, grande parte da literatura os emprega na resolução de problemas de Programação Linear Inteira (PLI). Por esse motivo e pelo fato de ainda não termos introduzido todos os conceitos necessários, faremos uma explicação breve do método usando conceitos da PLI quando oportuno. Mais tarde, voltaremos a apresentar o método de acordo com a otimização global ressaltando as semelhanças e diferenças entre a abordagem tradicional e esta última.

O algoritmo *branch-and-bound* inicia considerando o problema original como um todo, o particiona e calcula cotas inferiores (*bounding*) para o mínimo de f em cada subproblema. Cada um deles é, então, testado de acordo com uma ou mais regras de eliminação na intenção de descartar (*prune*) subproblemas onde se pode garantir não encontrar uma solução melhor que a melhor solução atualmente conhecida (*incumbent*). Subproblemas não eliminados são armazenados numa lista (L) para posterior ramificação (*branching*) até que o subproblema represente uma solução inviável, uma solução ótima (inteira) ou seja dominado (não ofereça solução melhor

que outra já considerada). Sempre que se chega a uma solução, ela é comparada com a melhor solução (doravante chamado de CS) de forma que CS seja atualizada se a nova solução for melhor (menor). Assim, CS guarda sempre uma cota superior para o mínimo de f.

A propriedade básica que permite a eliminação (enumeração implícita) de subproblemas é que: *subproblemas s_i provenientes da ramificação de um subproblema s_j possuem cotas inferiores para o mínimo de f maiores que a cota inferior de s_j* ($C_{Inf}(s_j) \leq C_{Inf}(s_i)$). Essa propriedade permite a existência da regra básica de eliminação: *seja um subproblema s_i com cota inferior $C_{Inf}(s_i)$ e uma cota superior atual CS. Se $C_{Inf}(s_i) > CS$, então s_i pode ser eliminado (e seus descendentes implicitamente) já que nenhum subproblema de s_i leva a uma solução melhor (menor) que CS.* Essa regra é padrão em qualquer algoritmo *branch-and-bound.* Outras regras de eliminação (dependentes do problema ou não) podem ser adicionadas e são conhecidas como regras/dispositivos de aceleração do algoritmo (ver 4.8 e 4.9) já que permitem que mais subproblemas sejam eliminados mais cedo, levando a uma enumeração implícita maior.

As relações entre cada subproblema s_i, as partições de s_i, geradas na ramificação criam uma estrutura hierárquica representável através de uma árvore. É a essas relações que nos referimos quando falamos da *árvore do branch-and-bound.* Tomando emprestado a nomenclatura tradicional da estrutura de árvore, temos: raiz (problema inicial), filhos do nó n (partições criadas a partir do subproblema n), descendentes de n (subárvores enraizadas nos filhos de n), pai do nó n (subproblema a partir do qual n foi gerado diretamente), ancestrais de n (cada um dos subproblemas no caminho entre n e a raiz da árvore), etc.

Mitten [61], em 1970, criou uma caracterização para os métodos *branch-and-bound* de acordo com a escolha feita para cada uma de quatro regras: **seleção** (como escolher o próximo subproblema a ser considerado), **divisão** (como particionar o subproblema atualmente considerado), **limitação** (como calcular cotas inferiores para os subproblemas — possivelmente incluindo também o cálculo da cota superior para o mínimo de f) e **eliminação** (como enumerar implicitamente subproblemas

pela aplicação de testes que garantem a qualidade dos subproblemas descendentes do subproblema considerado). Em seguida, detalharemos cada regra.

2.5.1 Seleção

A seleção pode ser de dois tipos: **a priori** (quando a ordem de varredura dos nós da árvore *branch-and-bound* é pré-estabelecida antes de iniciar o algoritmo) ou **adaptativas** (quando o próximo nó é escolhido, a cada iteração, dentre os subproblemas abertos (L)). Como exemplo do primeiro caso, temos as regras *Depth-First* (testar os subproblemas abaixo de um nó n antes de testar outros subproblemas que estejam no mesmo nível de n) e *Breadth-First* (testar todos os subproblemas de um mesmo nível antes de testar subproblemas do próximo nível). Já dentre as estratégias adaptativas podemos citar: *Best-First* (escolher sempre o elemento de L com menor cota inferior, *i.e.*, subproblema mais promissor), *Best-Estimate* (escolher o elemento de L mais provável, segundo um critério qualquer, de ter uma solução) e *Quick-Improvement* (escolher o elemento de L que forneça maior melhora da solução atual).

2.5.2 Divisão

A divisão se dá, como o nome indica, através da partição do espaço de busca em dois. No caso da PLI, consiste em escolher uma variável com valor não inteiro para adicionar uma restrição que tenta impedir essa variável de continuar com valor não-inteiro. Um exemplo de restrições seriam $x_i \leq \lfloor \overline{x_i} \rfloor$ e $x_i \geq \lceil \overline{x_i} \rceil$ que seriam adicionadas uma a cada um dos subproblemas gerados. A escolha da variável é feita de acordo com uma heurística adequada ao problema.

2.5.3 Limitação

A limitação é feita através do relaxamento de restrições de integralidade que fornece problemas com soluções menores servindo de cota inferior para o subproblema processado. Esta estratégia trabalha muito bem em conjunto com a estratégia de se-

leção *Depth-First* já que o trabalho para achar a solução ótima (cota inferior) do subproblema relaxado a partir da solução do subproblema-pai é bem pequeno. Mesmo assim, esse costuma ser o passo que mais demanda esforço computacional no algoritmo.

2.5.4 Eliminação

A eliminação basicamente se restringe aos casos onde ou o subproblema é inviável ou sua cota superior é pior que CS (teste do limite). Um outro caso, que pode ser incluído, é o de se encontrar uma solução ótima inteira para o subproblema relaxado, pois apesar do nó não ser de fato desconsiderado pelo algoritmo (ele pode passar a ser um novo CS), o algoritmo pára a enumeração nesse nó.

Durante a execução do algoritmo, cada subproblema pode estar em um desses 4 estados [24]:

a) Gerado: subproblema criado por ramificação, mas ainda não processado;

b) Limitado: cota inferior já foi calculada para o subproblema;

c) Examinado e ramificado: subproblema não é simples o bastante para ser resolvido e por isso foi ramificado e

d) Examinado e eliminado: subproblema não possui solução melhor que a atual.

A esses podemos incluir um quinto, por questões de completude:

e) Resolvido: subproblema apresenta uma possível solução ótima.

Subproblemas nos estados a) e b) são armazenados numa lista de subproblemas chamados ativos ou abertos (L). Subproblemas em c) e d) são simplesmente removidos da lista ativa e são chamados de fechados. Subproblemas em e) são transferidos da lista ativa para uma outra lista que chamaremos *lista de possíveis soluções* ($\bar{L}$). Problemas bem comportados (lineares e não-lineares convexos) geralmente possuem

apenas uma solução distinta e, por esse motivo, dispensam a existência dessa última lista. Nesses casos, a solução é dada por CS quando o algoritmo termina.

Apesar de os algoritmos *branch-and-bound* terem complexidade exponencial no pior caso [23], costumam apresentar bom desempenho em aplicações práticas. Além disso, existem dispositivos tanto intrínsecos ao método (como será visto mais adiante nas seções 4.8 e 4.9) quanto não (paralelismo) que permitem acelerar o andamento de tais algoritmos.

Passaremos, agora, a introduzir os conceitos básicos de aritmética intervalar para que possamos descrever o algoritmo *branch-and-bound* que empregamos na resolução do PCM.

Capítulo 3

Aritmética Intervalar

Por muito tempo, acreditou-se que nenhum algoritmo numérico poderia garantir encontrar uma solução global para o problema de otimização não-linear. A razão para isso era: "a função a ser minimizada só pode ser testada num número finito de pontos. Portanto, não como garantir saber-se se a função cai para um valor menor entre dois pontos testados". Embora o argumento possa ser verdadeiro usando-se a avaliação da função em pontos, isso não é verdade para métodos que podem produzir limites inferiores assintoticamente precisos para a faixa de valores da função sobre conjuntos compactos. A aritmética intervalar é capaz de fornecer esses limites inferiores assintoticamente precisos para a faixa de valores de uma função sobre um intervalo contínuo.

3.1 Introdução

A aritmética intervalar teve origem no trabalho de [64] com o objetivo original de controlar erros de arredondamento gerados pelas operações entre números ponto-flutuante utilizadas nos computadores. A idéia é utilizar um intervalo $[a; b]$ (onde a e b são números representáveis pela notação ponto-flutuante) ao invés de um único número a para representar um número real x de forma que a relação $a \leq x \leq b$ sempre se mantenha. Garantida essa relação, o número x será representado adequadamente, mesmo sendo a e b representados na inadequada (mas necessária) notação ponto-flutuante. Quanto menor for a diferença $b - a$, maior a precisão que

se tem sobre o valor de x.

Números constantes ou valores conhecidos são representados como intervalos do tipo $[x; x]$ (intervalos degenerados). Caso o valor a não seja um número representável segundo a notação ponto-flutuante, sempre poderemos determinar dois números a e b que sejam representáveis de forma que $x \in [a; b]$ onde $b - a$ forneça uma precisão suficiente para os cálculos que se pretende realizar. Por outro lado, as incógnitas de um problema podem ser representadas inicialmente por intervalos maiores os quais vão sendo diminuídos no decorrer da execução de um algoritmo até alcançarem uma precisão pré-estabelecida desejada. Material introdutório sobre aritmética intervalar pode ser encontrado em [3, 65, 67, 73]. Para um texto mais completo veja [74].

3.2 Convenções

Deste ponto em diante, adotaremos as seguintes convenções:

- $\mathbb{PF}$ é o conjunto dos números reais representáveis pela notação ponto-flutuante.
- $\mathbb{I}$ é o conjunto de todos os intervalos $[a; b]$ $(a, b \in \mathbb{PF})$. Usamos $\mathbb{PF}$ pois estamos interessados na representação computacional da aritmética intervalar, mas poderíamos definir $\mathbb{I}$ como subconjunto direto de $\mathbb{R}$.
- X é um intervalo determinado pelo intervalo $[\underline{x}; \overline{x}]$ $(\underline{x}, \overline{x} \in \mathbb{PF})$.
- Y é um intervalo determinado pelo intervalo $[\underline{y}; \overline{y}]$ $(\underline{y}, \overline{y} \in \mathbb{PF})$.
- f e g são funções reais.
- F e G são funções intervalares (definidas adiante).

3.3 Operações Básicas

A aritmética intervalar define as quatro operações $(+, -, \times, \div)$ básicas sobre os intervalos de forma que o seguinte princípio seja sempre mantido: *o intervalo resultante*

da operação de um intervalo X com um intervalo Y é um intervalo que contém todos os valores possíveis da operação real de um elemento de X com um elemento de Y.

$$X \circ Y := [\min_{\substack{x \in X \\ y \in Y}}(x \circ y); \max_{\substack{x \in X \\ y \in Y}}(x \circ y)]. \tag{3.1}$$

Por exemplo, $[2;3]+[1;3] = [3;6]$ pois o menor valor possível para a soma é $1+2$ ao passo que o maior é $3+3$.

Considerando agora que essas operações serão realizadas num computador, devemos testar os seguintes casos. Seja $[a;b]$ o resultado de uma operação:

- $a \notin \mathbb{PF}$: a receberá o maior $pf \in \mathbb{PF}$ menor que o valor de a calculado.
- $b \notin \mathbb{PF}$: b receberá o menor $pf \in \mathbb{PF}$ maior que o valor de b calculado.

A esse esquema de arredondamento dá-se o nome de *arredondamento exterior.* Ele permite que o intervalo resultante $[a;b]$ contenha o intervalo que de fato deveria ser retornado (se estes pertencessem a $\mathbb{PF}$) e, agora, pode ser representado sem perda de informação.

O arredondamento exterior está especificado no padrão IEEE sobre operações com ponto-flutuante para computadores [10]. Nem todas as arquiteturas seguem toda a especificação desse padrão. Para aquelas que o seguem, a implementação de uma biblioteca intervalar é possível e, em geral, existe disponível na Internet. Por exemplo: CXSC++ [46], PROFIL/BIAS [47, 48]. Em [79] discute-se implementações de *hardware* e de *software* para aritmética intervalar e o artigo [40], disponível na *Internet*, fornece muita informação introdutória sobre o assunto: aplicações, livros, centros de pesquisa, bibliografia (*on-line* ou não), bibliotecas, além de uma introdução básica.

A seguir temos as definições básicas das operações entre intervalos segundo o

critério definido na equação 3.1:

$$\begin{array}{rcl} X+Y & := & [\underline{x}+\underline{y};\overline{x}+\overline{y}], \\ X-Y & := & [\underline{x}-\overline{y};\overline{x}-\underline{y}], \\ X\times Y & := & [a;b] \ \ (a=\min\{\underline{x}\underline{y},\overline{x}\underline{y},\underline{x}\overline{y},\overline{x}\overline{y}\} \text{ e } b=\max\{\underline{x}\underline{y},\overline{x}\underline{y},\underline{x}\overline{y},\overline{x}\overline{y}\}), \\ \frac{1}{Y} & := & \left[\frac{1}{\overline{y}};\frac{1}{\underline{y}}\right] \ \ (0\notin Y), \\ \frac{X}{Y} & := & X*\frac{1}{Y} \ \ (0\notin Y), \end{array} \tag{3.2}$$

$$\sqrt{X} = \left[\sqrt{\underline{x}};\sqrt{\overline{x}}\right] \ \ (X\geq 0),$$

$$X^n := \begin{cases} [1;1] & \text{se } n=0, \\ [\underline{x}^n;\overline{x}^n] & \text{se } \underline{x}\geq 0 \text{ ou se } n \text{ é ímpar}, \\ [\overline{x}^n;\underline{x}^n] & \text{se } \overline{x}\leq 0 \text{ e } n \text{ é par}. \\ [0;\max(\underline{x}^n,\overline{x}^n)] & \text{se } \underline{x}\leq 0\leq\overline{x} \text{ e } n \text{ é par}. \end{cases}$$

O *centro* m de X é:

$$m(X)=\frac{\underline{x}+\overline{x}}{2}. \tag{3.3}$$

O *comprimento* w de X é

$$w(X)=\overline{x}-\underline{x}. \tag{3.4}$$

A *magnitude* de X ($|X|$) é

$$|X|=\max\{|\underline{x}|,|\overline{x}|\}. \tag{3.5}$$

A *mignitude* de X ($mig(X)$) é

$$mig(X)=\begin{cases} \underline{x} & \text{se } \underline{x}>0, \\ -\overline{x} & \text{se } \overline{x}<0 \text{ e} \\ 0 & \text{se } \underline{x}\leq 0\leq\overline{x}. \end{cases} \tag{3.6}$$

Mais sobre a especificação das operações aritméticas sobre intervalos pode ser encontrada em [31, 80, 10].

É importante notar que:

a) $\forall r\in\mathbb{R}$, vale que $r-r=0$. Mas para $i\in\mathbb{I}, i-i=[0;0]$ só vale se i for um intervalo degenerado;

b) $1/Y$ e X/Y não estão definidos para $0\in Y$;

c) X^n não está definido em função da operação de multiplicação;

d) A operação de multiplicação é subdistributiva.

Detalhes sobre o assunto serão dados nas seções 3.4 e 3.6 e pode ser encontrados em [30, 31].

3.4 Aritmética Intervalar Estendida

A divisão por intervalo contendo zero (caso c) não pôde ser definida anteriormente, pois os intervalos estão definidos apenas para extremidades finitas. Precisamos definir intervalos com extremidades infinitas antes de permitir tal operação.

Partindo-se das seguintes relações:

$$\lim_{x \to 0^+} n/x = +\infty \ (n > 0)$$
$$\lim_{x \to 0^-} n/x = -\infty \ (n > 0)$$

É razoável definir X/Y (com $\underline{y} \leq 0 \leq \overline{y}$ e $\underline{y} < \overline{y}$) da seguinte maneira:

$$X/Y := \begin{cases} [\overline{x}/\underline{y}; \infty] & \text{se } \overline{x} \leq 0 \text{ e } \overline{y} = 0, \\ [-\infty; \overline{x}/\overline{y}] \cup [\overline{x}/\underline{y}; \infty] & \text{se } \overline{x} \leq 0 \text{ e } \underline{y} < 0 < \overline{y}, \\ [-\infty; \overline{x}/\overline{y}] & \text{se } \overline{x} \leq 0 \text{ e } \underline{y} = 0, \\ [-\infty; \infty] & \text{se } \underline{x} < 0 < \overline{x}, \\ [-\infty; \underline{x}/\underline{y}] & \text{se } \underline{x} \geq 0 \text{ e } \overline{y} = 0, \\ [-\infty; \underline{x}/\underline{y}] \cup [\underline{x}/\overline{y}; \infty] & \text{se } \underline{x} \geq 0 \text{ e } \underline{y} < 0 < \overline{y} \text{ e} \\ [\underline{x}/\overline{y}; \infty] & \text{se } \underline{x} \geq 0 \text{ e } \underline{y} = 0. \end{cases} \tag{3.7}$$

Por motivos de consistência, devemos adicionar mais algumas definições para que a definição de divisão intervalar seja coerente com as outras:

$$\begin{array}{ll} [\underline{x}; \overline{x}] + [-\infty; \overline{y}] & := [-\infty; \overline{x} + \overline{y}], \\ [\underline{x}; \overline{x}] + [\underline{y}; \infty] & := [\underline{x} + \underline{y}; \infty], \\ [\underline{x}; \overline{x}] \pm [-\infty; \infty] & := [-\infty; \infty], \\ [\underline{x}; \overline{x}] - [-\infty; \overline{y}] & := [\underline{x} - \overline{y}; \infty] \text{ e} \\ [\underline{x}; \overline{x}] - [\underline{y}; \infty] & := [-\infty; \overline{x} - \underline{y}]. \end{array} \tag{3.8}$$

A operação estendida de divisão (assim como no método de Newton intervalar discutido na seção 4.9.b) permite como solução a união de conjuntos disjuntos. Surge um problema matemático e um problema computacional no tratamento desse caso. O primeiro é resolvido com a seguinte definição:

$$(V \cup W) \pm Z := (V \pm Z) \cup (W \pm Z) \tag{3.9}$$

Já o problema computacional se resolve com a inclusão, na estrutura de dados usada para representar intervalos, de informações adicionais que permitam a repre-

sentação dos dois intervalos disjuntos. Uma solução, que inclui a informação do "buraco" entre os dois intervalos é dada na implementação encontrada em [36, 75].

3.5 Funções Intervalares

Funções intervalares são funções $F : \mathbb{I}^n \rightarrow \mathbb{I}$ definidas em termos de operações sobre intervalos. O objetivo na definição de funções intervalares é o de obter, através da função F, informações sobre o comportamento de uma função f real. Por exemplo, se queremos obter informação sobre a faixa de valores onde varia a função f num dado intervalo do seu domínio.

Definimos então que F é *uma extensão intervalar* de f se, e somente se, $F(x) = f(x)$, *i.e.*, o intervalo retornado por F quando aplicada ao intervalo degenerado x é um intervalo degenerado de mesmo valor que $f(x)$. Uma extensão intervalar pode ser conseguida substituindo-se valores reais por intervalos degenerados e as operações reais pelas correspondentes intervalares.

Mais ainda, queremos não apenas obter informação de f num intervalo degenerado, mas em qualquer intervalo. Para tanto, a propriedade de extensão intervalar não é suficiente. Definimos então que: F é *monótona inclusiva* ou que é uma *inclusão monotônica* de F se satisfaz:

$$Y_i \subset X_i \ \ (i = 1, 2, ..., n) \Rightarrow F(Y_1, Y_2, ..., Y_n) \subset F(X_1, X_2, ..., X_n), \qquad (3.10)$$

que significa: se o intervalo considerado diminui, o intervalo da imagem também diminui.

A figura 3.1 mostra o comportamento de uma função monótona inclusiva. Note que $X \subset Y$ e $F(X) \subset F(Y)$. Ainda na figura 3.1 podemos ver a função $f(x)$ em branco, a sua extensão intervalar $F(x)$.

Em [31], é mostrado que, funções definidas segundo as operações intervalares como as aqui definidas, são monótonas inclusivas. E [3] mostra que essas funções implementadas num computador usando arredondamento exterior também gozam da mesma propriedade.

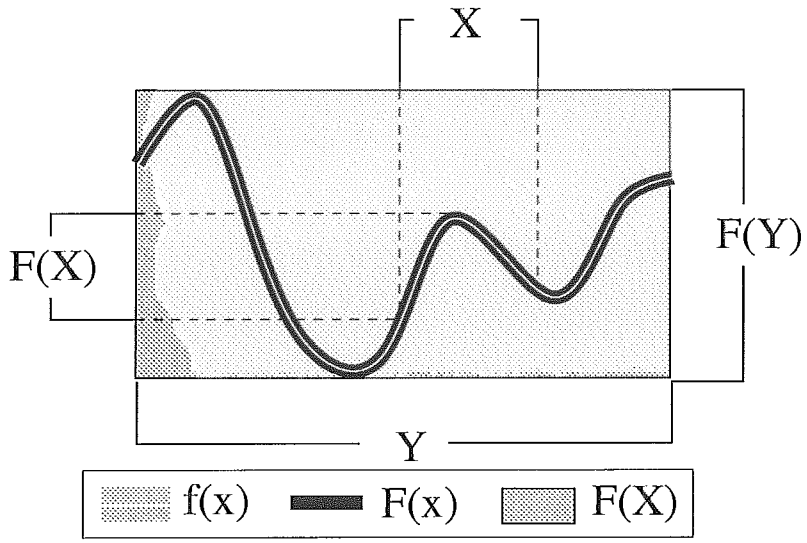

Figura 3.1: f(x), F(x), F(X) e a propriedade de inclusão monótona

Mesmo funções trancendentais (sen, cos, log, exp, etc) por serem implementadas em computadores usando uma aproximação por séries (construídos sobre as operações básicas) como por exemplo:

$$sen(x) = \sum_{i=1}^{\infty}(-1)^i \times \frac{x^{2i+1}}{(2i+1)!} \qquad \cos(x) = \sum_{i=1}^{\infty}(-1)^i \times \frac{x^{2i}}{(2i)!}$$

também podem ter versões intervalares disponíveis.

Agora temos condições suficientes para estabelecer o Teorema Fundamental da Aritmética Intervalar [31]:

Teorema 3.5.1 *Se $F(X_1, X_2, ..., X_n)$ é uma extensão intervalar monótona inclusiva de uma função real $f(x_1, x_2, ..., x_n)$, então $f(x_1, ..., x_n) \in F(X_1, ..., X_n)$, $\forall x_i \in X_i, (i = 1, 2, ..., n)$.*

Prova: Ver [31]. ∎

Ou seja, F contém informação de toda a imagem de f para qualquer valor x no intervalo X. Esse resultado é muito mais abrangente que a propriedade de extensão intervalar já que se aplica a qualquer intervalo e não apenas a intervalos degenerados.

3.6 Discrepâncias Entre Aritmética Intervalar e Aritmética Real

As observações feitas na seção 3.3 acontecem porque a aritmética intervalar não é exatamente equivalente à aritmética real. As diferenças mencionadas são:

a) Problema de Dependência da Subtração: Enquanto na aritmética real $x - x = 0, \forall x \in \mathbb{R}$, na aritmética intervalar acontece que o resultado de $X - X$ é avaliado como $X - Y$, *i.e.* sem levar em conta que $X = Y$. Exemplo: $[1;1] - [1;1] = [0;0]$ e $[1;2] - [1;2] = [-1;1]$. Como se pode perceber, a equivalência manter-se-á apenas para intervalos degenerados.

b) Problema de Dependência da Multiplicação: Assim como na subtração, $X \times X$ é avaliado como $X \times Y$, para $X = Y$. Isso afeta diretamente a definição de potência intervalar pois $X^2 \subseteq X \times X$ (ao invés de se manter a igualdade como na aritmética real). Exemplo: $X = [-1;3], X \times X = [-3;9], X^2 = [0;9]$. Note que $X \times X$ não está errado, apesar de não devolver o intervalo mais estreito, pois é sabido que, qualquer que seja $x \in X$, seu quadrado não desce abaixo de zero. Por esse motivo, a operação de potência tem sua própria definição ao invés de ser definida em função da multiplicação.

c) Subdistributividade da multiplicação: Eis mais um exemplo de propriedade intervalar que é mais geral que a propriedade real correspondente: $(X + Y) \times Z \subseteq X \times Z + Y \times Z$. Exemplo:

	$(X+Y) \times Z$	$X \times Z + Y \times Z$
$X = [1;2], Y = [-3;3], Z = [0;4]$	$[-8;20]$	$[-12;20]$
$X = [1;2], Y = [0;4], Z = [-3;3]$	$[-18;-18]$	$[-18;-18]$

Para eliminar esses efeitos, devemos escolher expressões que possuam pouca ou nenhuma repetição da mesma variável e que forneçam o menor intervalo já que a intenção é obter intervalos cada vez mais estreitos ao redor da(s) solução(ões). Por todos esses motivos, algumas expressões equivalentes na aritmética real o deixam de ser na aritmética intervalar.

Tabela 3.1: Expressões intervalares não equivalentes

Intervalo	Expressão 1	Expressão 2
	$X^2 - X$	$(X - 1/2)^2 - 1/4$
$X = [0; 2]$	$[-2; 4]$	$[-1/4; 2]$

Em [78], o autor sustenta que esses problemas se originam da definição de igualdade intervalar e propõe uma definição mais relaxada chamada "igualdade local".

3.7 Vetores e Matrizes

Para que possamos lidar com funções intervalares de dimensão n ($F : \mathbb{I}^n \rightarrow \mathbb{I}$), existem algumas definições relativas a vetores e matrizes que devem ser feitas.

Um vetor intervalar $V, U \in \mathbb{I}^n$ é chamado *caixa* para o qual se define:

$$\begin{array}{l} x \in V \Leftrightarrow x_i \in V_i \text{ para } i = 1, ..., n \\ V \subset U \Leftrightarrow V_i \subset U_i, \text{para } i = 1, ..., n \\ m(V) = (m(V_1), m(V_2), ..., m(V_n)) \\ w(V) = \max_{i=1,...,n}(w(V_i)) \\ \| V \| = \max_{i=1,...,n} |V_i| \end{array} \tag{3.11}$$

Da mesma forma, para matrizes A^I e B^I da forma (a_{ij}^I) $i = 1, ..., m$, $j = 1, ..., n$ e $a_{ij}^I \in \mathbb{I}$, temos:

$$\begin{array}{l} A^I \subset B^I \Leftrightarrow a_{ij}^I \subset b_{ij}^I \text{ para } i = 1, 2, ..., m; j = 1, 2, ..., n; \\ A \in A^I \leftrightarrow a_{ij} \in a_{ij}^I \text{ para } i = 1, 2, ..., m; j = 1, 2, ..., n; \\ m(A^I) = m(a_{ij}^I) \text{ para } i = 1, 2, ..., m; j = 1, 2, ..., n; \\ w(A^I) = \max_{i=1...m, j=1,...,n}(w(a_{ij}^I)); \\ \| A^I \| = \max_{i=1,...,m, j=1,...,n} |a_{ij}^I|. \end{array} \tag{3.12}$$

A^I é diagonal dominante, se e somente se:

$$mig(a_{ii}^I) \geq \sum_{j=1 j \neq i}^{n} |a_{ij}^I|, i = 1, 2, ..., n. \tag{3.13}$$

3.8 Aplicações

Originalmente, como já afirmamos, a aritmética intervalar foi desenvolvida para eliminar/reduzir erros de arredondamento em cálculos computacionais. Mas, re-

sultados posteriores permitiram que algoritmos intervalares trabalhassem com informação sobre a função objetivo de um problema de otimização num intervalo contínuo e não mais em um número finito de pontos como os algoritmos clássicos que utilizam a aritmética real. Dessa forma, ao invés de procurar, por exemplo, um ponto crítico de f através de seu gradiente g, podemos determinar se um intervalo possui ou não tal ponto pela simples avaliação de $G(X)$. Se 0 pertence a $G(X)$, então o intervalo X contém um ponto crítico de f (a recíproca é verdadeira).

Passaremos, agora, a abordar o algoritmo *branch-and-bound* aplicado à otimização global fazendo uso da aritmética intervalar.

Capítulo 4

Branch-and-bound Intervalar

O uso da aritmética intervalar não se restringe à simples obtenção de cotas inferiores para os subproblemas, apesar desta ser talvez sua maior contribuição (fornecer limites para f a um baixo custo, como veremos a seguir), a aritmética intervalar está presente em todo o algoritmo. Visto que o algoritmo *branch-and-bound* foi explicado em termos gerais usando um leve enfoque de PLI, vamos agora descrever o algoritmo *branch-and-bound* intervalar ressaltando suas diferenças/semelhanças com o *branch-and-bound* clássico (PLI).

4.1 Esquema Geral e Solução

A estrutura básica de seleciona-divide-testa-elimina se mantém. Como já foi mencionado, devemos admitir a existência de muitos pontos de mínimo local. Alguns deles podem estar a uma distância muito pequena do mínimo global. Por esse motivo, mantém-se a lista de possíveis soluções $\overline{L}$ que conterá, ao final da execução do algoritmo, uma lista de caixas de largura pequena contendo cada uma delas pelo menos um mínimo local de f, e, dentre as quais, devem estar uma ou mais soluções globais.

4.2 Cota Superior

A forma de obtenção de cotas superiores, que antes acontecia apenas quando a solução de um subproblema relaxado era inteiro, não faz mais sentido. Ao invés disso, passamos a escolher um ponto qualquer (tradicionalmente o ponto médio do intervalo) dentro de cada subproblema e compará-lo com CS para substituir por este se for o caso. Note que essa comparação ocorre muito mais freqüentemente agora que no *branch-and-bound* tradicional propiciando, a princípio, maior enumeração implícita devido a uma atualização mais freqüente de CS.

4.3 Pré-processamento

De forma a acelerar o processamento, pode-se executar algum algoritmo de busca local na tentativa de conseguir uma boa cota superior inicial e, assim, aumentar o número de subproblemas eliminados. Esse é um exemplo de dispositivo de aceleração (mencionado em 2.5).

4.4 Término

O critério de parada de qualquer algoritmo *branch-and-bound* é "terminei de processar todo o espaço de busca?" que, computacionalmente, traduz-se em "$L = \emptyset$?". Diferenças aparecem entre a otimização discreta e contínua no critério de remoção de elementos de L (o que influencia diretamente o esvaziamento de L).

Como estamos trabalhando com intervalos e, teoricamente, sempre podemos dividi-los ao meio, o critério de eliminação para se detectar uma solução é substituído pelos testes de tolerância do intervalo

$$w(X) < \varepsilon_X \text{ e/ou } w(F(X)) < \varepsilon_f. \tag{4.1}$$

Isso quer dizer que, sempre que um intervalo for mais estreito que uma precisão pré-estabelecida, consideramos o subproblema correspondente resolvido. Como não

estamos considerando restrições que não sejam de caixa, não há eliminação por inviabilidade (já que todos os subproblemas são viáveis).

4.5 Regras de Seleção

Uma diferença importante entre o *branch-and-bound* intervalar e o tradicional acontece nas regras de limitação e tem reflexos diretos nas regras de seleção. Como foi sugerido na seção 2.5.3, implementações do algoritmo *branch-and-bound* tradicional costumam empregar seleção *Deph-First* para economizar tempo de processamento no passo correspondente dos subproblemas relaxados (partindo-se da solução do subproblema-pai). Como a limitação não é mais feita através de resolução de um PPL e também não demanda tanto esforço computacional, podemos desassociar a escolha da regra de seleção da regra de limitação. Isso nos deixa outras opções a considerar.

As regras de seleção agem como uma função que ordena os subproblemas abertos identificando o próximo a ser considerado/selecionado. Uma função heurísitca h deve seguir as seguintes propriedades:

$$\begin{array}{c} P_i \neq P_j \Rightarrow h(P_i) \neq h(P_j) \\ P_i \text{ é ancestral de } P_j \Rightarrow h(P_i) < h(P_j). \end{array} \tag{4.2}$$

A função heurísitca possui outras propriedades são *desejáveis* para essa "função heurística", como por exemplo:

$$h(P_i) < h(P_j) \Rightarrow \underline{f}(P_i) \leq \underline{f}(P_j) \tag{4.3}$$

(onde P_i e P_j são subproblemas abertos), *i.e.*, se a função de seleção indica P_i para ser considerado antes de P_j então P_i possui uma cota inferior menor que P_j, quando se aplica a regra *Best-First* explicada mais adiante. Uma regra ou função que observa essa propriedade é chamada *não-enganosa* (*nonmisleading*).

A seguir damos uma explicação detalhada do uso das principais regras de seleção.

a) *Depth-first*: segundo essa estratégia, o próximo subproblema a ser considerado será **aquele mais recentemente ramificado**. Para tanto, é conveniente que

a lista de subproblemas ativos seja implementada usando uma estrutura de dados tipo pilha (*c.f.* [2]) que é capaz de manipular seus elementos de maneira eficiente exatamente de acordo com a estratégia. Essa estratégia possui a vantagem de poder achar uma solução próxima à ótima mais rápido, além de não fazer crescer a lista de problemas ativos demasiadamente. Por outro lado, essa estratégia faz o algoritmo testar muitos subproblemas, além do quê, sua capacidade de eliminar subproblemas (enumerar implicitamente) depende de uma boa cota superior ser encontrada o mais cedo possível durante a execução do algoritmo. Ver figura 4.1.

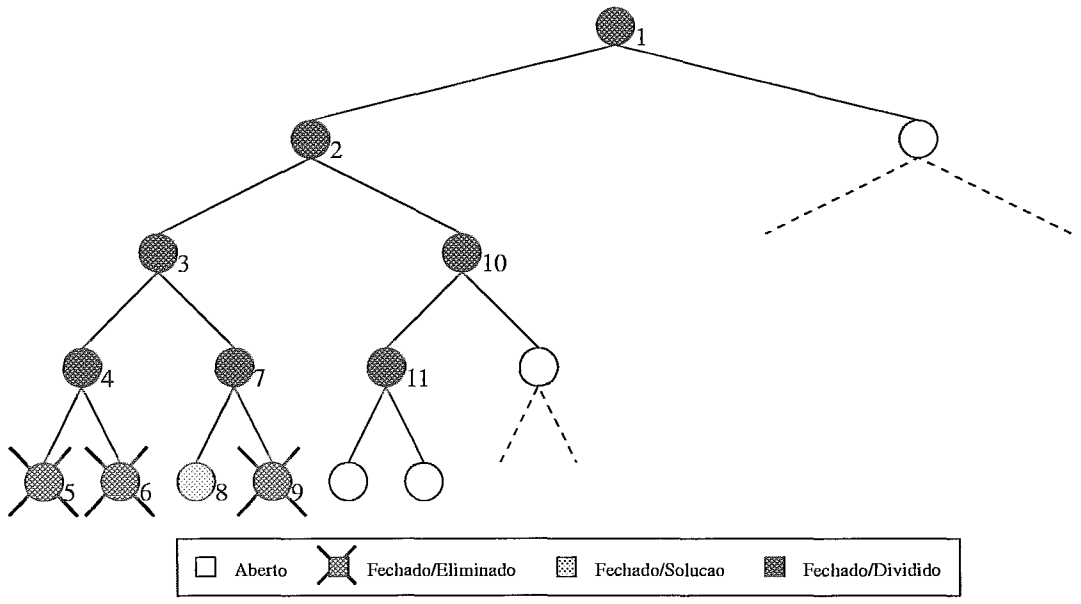

Figura 4.1: Seqüência-exemplo de nós testados usando a estratégia *Depth-First*. Os números indicam a ordem de seleção. Números indicam a ordem de seleção.

b) *Best-first*: segundo essa estratégia, o próximo subproblema a ser considerado será **aquele que possuir a menor cota inferior** seguindo a intuição de que este é um subproblema promissor já que pode levar a uma solução de valor mais baixo (segue a mesma intuição que *Best-Estimate* na seção 2.5.1). Para uso dessa estratégia, é aconselhável implementar a lista L como uma fila de prioridades (*c.f.* [2]) que mantém os elementos de maior prioridade (menor cota inferior) no topo da estrutura. Essa estratégia costuma gerar muitos subproblemas que vão se acumulando na lista de subproblemas ativos (e consumindo mais memória). Mas esse fato se reflete na eficiência da estratégia

que testa/decompõe um número pequeno de subproblemas e, em certas condições, pode minimizar esse número. É possível mostrar que essa estratégia não seleciona subproblemas com cota inferior maior que a solução global, *i.e.*, encontra a solução global antes e elimina estas implicitamente. Ver figura 4.2.

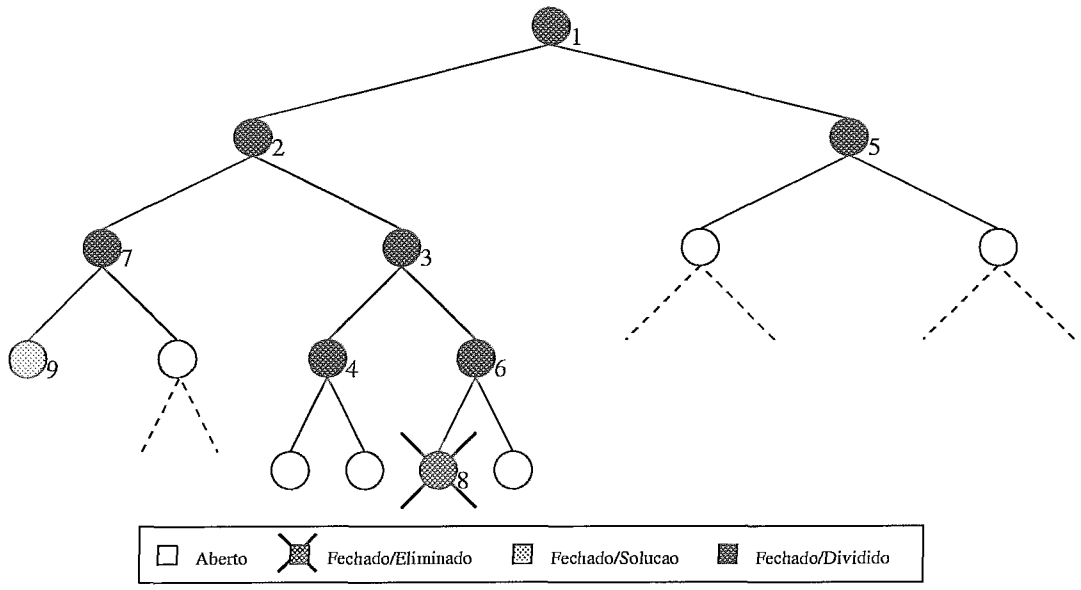

Figura 4.2: Seqüência-exemplo de nós testados usando a estratégia *Best-First*. Os números indicam a ordem de seleção. Números indicam a ordem de seleção.

c) Índice de Rejeição: uma variação da estratégia *Best-First* foi proposta recentemente (1998) em [14] que consiste na redefinição do que seja o subproblema mais promissor. Nesse trabalho (e em outros subseqüentes) analisa-se o comportamento do chamado Índice de Rejeição que, idealmente, mede o quão próximo o intervalo atual está da solução global. Esse índice é calculado segundo a seguinte expressão:

$$pf^* = \frac{f^* - \underline{F}(X)}{\overline{F}(X) - \underline{F}(X)} \tag{4.4}$$

onde f^* é a solução ótima, $\underline{F}(X)$ e $\overline{F}(X)$ são a cota inferior e superior para F em X (o intervalo correspondente ao subproblema atualmente processado). Portanto, os subproblemas são armazenados numa fila de prioridade como no caso anterior (*Best-First*), mas **a prioridade desta vez é dada pelo maior** pf^* ao invés da menor cota inferior.

Como a solução ótima nem sempre (na prática, poderíamos até dizer nunca) é conhecida, o índice pode ser calculado aproximando-se o valor da solução ótima pela cota superior desta (CS) em vigor no momento:

$$p(CS, X) = \frac{CS - \underline{F}(X)}{\overline{F}(X) - \underline{F}(X)} \tag{4.5}$$

e, à medida que o algoritmo prossegue, CS tende para f^*, o que nos leva à equação 4.4.

Essa estratégia carece de cuidados especiais em determinadas situações (intervalo X estreito e função variando pouco no intervalo X), pois, como pode ser visto pela expressão acima, o denominador pode se aproximar demasiadamente de zero nesses casos. Os resultados reportados do uso desse índice são bons, apesar do cuidado necessário com o denominador. Outra peculiaridade na literatura é do fato da maioria dos artigos encontrados usarem pf^* (valor quase sempre desconhecido na prática),

d) *Breadth-First*: Também chamada de *Oldest-First*, processa nós da árvore de forma que **todos os nós do nível n são considerados antes que o primeiro nó do próximo nível o seja.** Isso nos leva a um algoritmo que vai testar quase todos os nós da árvore, vai demorar para encontrar uma solução ótima (com precisão adequada) e a enumeração implícita não será favorecida. Por estes motivos, essa estratégia de seleção não será considerada.

4.6 Regras de Divisão

Aqui, as idéias são semelhantes às apresentadas na seção 2.5.2, mas são implementadas de maneira diferente. As regras mais usadas na literatura são explicadas e comparadas em [16]. A escolha da direção de divisão pode ser formulada matematicamente como:

$$k := \arg \min_{i=1,\ldots,n} \max D(i) \tag{4.6}$$

onde $J = 1, 2, 3, ..., n$ é o conjunto de índices que especifica as variáveis do problema.

Na figura 4.3 temos uma ilustração das três opções de divisão de uma caixa tridimensional.

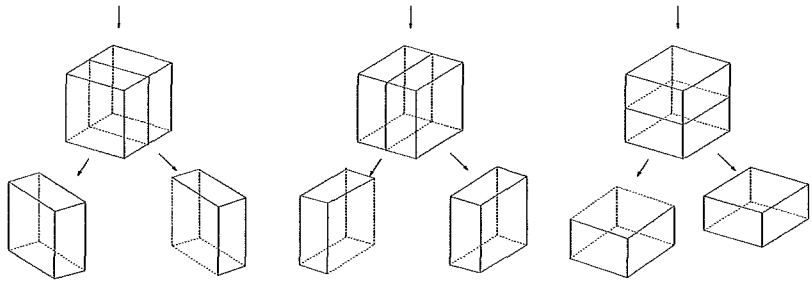

Figura 4.3: Exemplo de divisão.

A seguir discutiremos as regras usadas para calcular D(i):

Regra A:

$$D(i) := w(X_i). \tag{4.7}$$

Consiste em escolher a variável cujo intervalo possui maior largura ($w(X)$) (*c.f.*, [73]). A idéia é dividir o intervalo original de maneira uniforme. Essa regra possui uma análise de convergência relativamente simples (*c.f.*, [73, Cap 3, Teorema 6]). Além disso, não depende de teste de monotonicidade para convergir (*c.f.*, [15, 73]).

Regra B:

$$D(i) := w(G_i(X)) \times w(X_i). \tag{4.8}$$

Essa heurística foi proposta por Hansen e Walster [31]. Admita que x é o ponto médio da caixa X. A regra B pretende medir o quanto f varia quando x_i varia em X_i e escolher a variável que provoque a maior variação. Matematicamente:

$$\begin{aligned} W_i = \; & \max_{t \in X_i} F(m(X_1), ..., m(X_{i-1}), t, m(X_{i+1}), ..., m(X_n)) \; - \\ & \min_{t \in X_i} F(m(X_1), ..., m(X_{i-1}), t, m(X_{i+1}), ..., m(X_n)) \end{aligned}$$

Regra C:

$$D(i) := w(G_i(X) \times (X_i - m(X_i))). \tag{4.9}$$

Em [74], o autor propôs essa nova regra baseada na regra de Hansen e Walster (regra B). Nesse caso, com a ajuda de algum algebrismo, Ratz encontra uma nova expressão que representa a contribuição de cada variável para a largura da função objetivo ($w(F(X))$). A diferença para a regra anterior é que, aqui a largura do produto de intervalos é maximizada e não o produto de larguras. O que fornece diferentes valores devido àsubdistributividade da multiplicação intervalar (seção 3.6).

Outra questão envolvida na divisão consiste em definir em quantas partes será dividido o subproblema. Tradicionalmente usa-se uma bipartição simples, mas alguns autores (*c.f.*, [14, 13]) advogam que a multipartição, bem calibrada, pode trazer bons resultados tanto no número de avaliações da função quanto no tamanho máximo de L.

4.7 Regras de Limitação

As estratégias de limitação se torna trivialmente implementável quando se faz uso da aritmética intervalar. Basta avaliar F para a caixa correspondente ao subproblema atual para conseguir as cotas inferior e superior de f. É interessante lembrar que, quanto menos f sofre dos problemas de dependência, mais eficiente se torna o algoritmo por motivos óbvios. Ou seja, a eficiência do algoritmo depende da implementação das funções intervalares.

4.8 Regras de Eliminação

Costumam ser a parte do algoritmo *branch-and-bound* mais dependente do problema. Curiosamente, a aritmética intervalar permite descrever algumas regras novas que são independentes do problema. As regras de eliminação podem ser:

a) Teste do ponto médio ou do limite: Seja x um ponto qualquer de X (tradicionamente usa-se o ponto médio) contido na caixa inicial B. Seja também $F(x) = [\underline{x}; \overline{x}]$. Se existe uma outra caixa $Y \subset B$ cuja $F(Y) = [\underline{y}; \overline{y}]$ é tal que $\underline{y} > \overline{x}$, então Y não pode conter um minimizador menor que x e por isso Y não pode conter um minimizador global sendo portanto possível eliminar Y. Na figura 4.4 (a), o intervalo $X1$ pode ser eliminado pois $F(X1)$ está acima do intervalo CS definido por $[\underline{f}; \overline{f}]$.

b) Teste de dominância: É uma generalização do teste anterior (a) onde um subproblema P_i pode ser eliminado sem perda de otimalidade se existir um subproblema P_j para os quais a relação de dominância D garanta, com auxílio de informação intrínseca do problema, que P_i não pode conduzir a uma solução melhor que a de P_j (P_j domina P_i ou $P_j D P_i$). Uma relação de dominância D' é dita mais forte que $D (D' \supset D)$ se, e somente se, $P_i D P_j$ implica que $P_i D' P_j$ também.

   Relações de dominância foram introduzidas em [49] e estudadas em detalhe em [37]. Para que uma relação entre subproblemas seja considerada de dominância, ela deve obedecer às seguintes propriedades:

   (a) $P_i D P_j \leftrightarrow f(P_i) \leq f(P_j)$;

   (b) D é uma ordem parcial, *i.e.*, transitiva ($P_i D P_j$ e$P_j D P_k \leftrightarrow P_i D P_k$), reflexiva ($P_i D P_i$) e antisimétrica ($P_i D P_j$ e $P_j D P_i \leftrightarrow P_i = P_j$);

   (c) $P_i D P_j$ e $P_i \neq P_j$ implica que algum descendente P_i' satisfaz $P_i' D P_j'$ e $P_i' \neq P_j', \forall P_j'$ descendente de P_j;

   (d) Não existir um conjunto de problemas $P_{i_1}, P_{i_2}, ..., P_{i_{k+1}}$ ($k \geq 2$ e P_{i_1}, P_{i_2}, ..., P_{i_k} distintos) gerados pelo algoritmo *branch-and-bound* considerado tal que: (1) P_{i_t} é descendente próprio de P_i ou então $P_{i_t}, P_{i_{t+1}}$ satisfazem $P_{i_t} D P_{i_{t+1}} \wedge f(P_{i_t}) = f(P_{i_{t+1}})$ para $t = 1, ..., k$ e (2) $P_{i_{k+1}} = P_{i_1}$ (caminho fechado).

   Em [37] tais propriedades são extensamente estudadas.

c) Teste de monotonicidade: Admitindo-se que f seja continuamente diferenciável, sabemos que o gradiente g de f é zero no mínimo global. Na verdade, isso

acontece em todos os pontos estacionários.

O teste consiste em avaliar a função $G(X)$ num dado subproblema X. Se o subproblema possui algum ponto crítico nesse intervalo então $G(X) \subset \mathbf{0}$. Caso exista pelo menos uma componente de $G(X)$ que não contenha 0, então não existe um ponto-crítico e X pode ser eliminado. A figura 4.4 (b) ilustra esse tipo de teste onde $X2$ está associado a um intervalo onde a função é monótona.

Esse teste, bem como o seguinte, não contempla a possibilidade de que o mínimo esteja em umas das extremidades da caixa analisada. Esse caso deve ser analisado à parte.

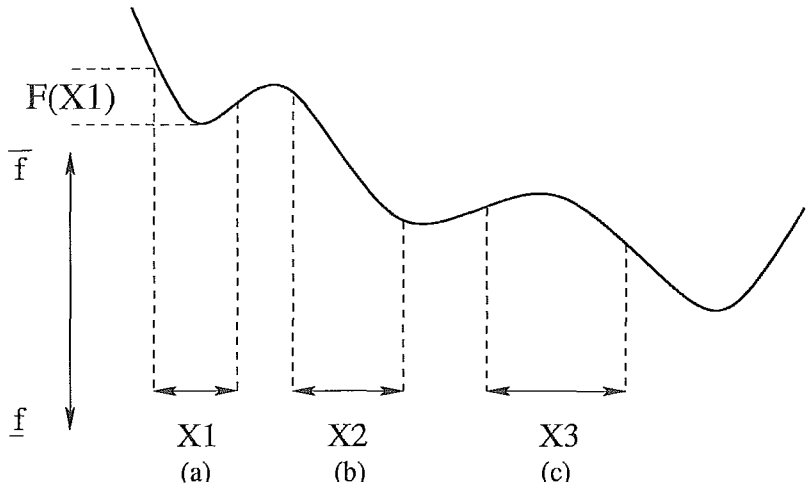

Figura 4.4: Exemplos de testes de eliminação

d) Teste de concavidade: Supondo agora que $f \in \mathcal{C}^2$, f deve apresentar convexidade em alguma vizinhança do mínimo global (assim como dos mínimos locais). Assim, usando um argumento similar ao do teste de monotonicidade: se formos capazes de mostrar que $H(X)$, a hessiana intervalar de f, não é semidefinida positiva em X, poderemos eliminar o intervalo X.

Uma condição necessária para tanto é $H_{ii}(X) < 0$. Nesse caso, basta avaliar os termos da diagonal de $H(X)$ procurando por pelo menos uma das componentes que seja estritamente negativa fazendo com que $H(X)$ não seja semidefinida positiva, o que significa que não existe um ponto de mínimo nesse intervalo, nos permitindo, portanto, eliminá-lo.

Uma condição suficiente para H ser semidefinida positiva é que o primeiro menor principal de cada ordem $i = 1, ..., n$ deve ser não-negativo. Essa condição demanda muita computação e, por isto, não compensa o esforço computacional para realizar o teste. A figura 4.4 (c) também ilustra o uso desse teste, $X3$ é eliminado porque não apresenta ponto de mínimo (apesar do teste de monotonicidade não poder eliminar esse intervalo).

Note que, aplicando-se mais esforço computacional (testes mais complexos) podemos eliminar mais pedaços do espaço de busca. É interessante identificar até quando o custo desses testes vai efetivamente acelerar a execução do algoritmo.

Em [77] podemos encontrar uma boa explicação de regras de eliminação para *branch-and-bound* intervalar.

4.9 Regras de Redução

As regras de redução, apesar de possuírem a mesma finalidade das regras de eliminação, são mais flexíveis à medida que permitem a eliminação de parte do intervalo associado ao subproblema (reduzindo, portanto, o mesmo). Duas regras de redução baseadas na aritmética intervalar são: redução via expansão de Taylor e redução pelo método de Newton intervalar que serão descritas a seguir.

a) Redução via expansão de Taylor: Seja $V \in \mathbb{I}^n$ a caixa sendo atualmente processada e $v, u \in \mathbb{R}^n$ pontos de V onde $v = m(V)$. Desejamos remover a parte U (possivelmente $U = V$) de V tal que $F(U) > CS$. Para tanto, ao invés de avaliar $F(U)$, calcularemos $f(u)$ pela expansão de Taylor até primeira ordem:

$$f(u) \in f(v) + \sum_{i=1}^{n} (u_i - v_i) g_i(V_1, V_2, ..., V_i, v_{i+1}, ..., v_n) \qquad (4.10)$$

onde $g_i(V_1, V_2, ..., V_i, v_{i+1}, ..., v_n) = g_i^1$ é a i-ésima componente do gradiente de f. Para cada componente $j = 1, ..., n$ temos:

$$f(u) \in f(v) + (u_j - v_j) g_j^1 + \sum_{i=1, i \neq j}^{n} (u_i - v_i) g_i^1, \text{ para } j = 1, ..., n. \qquad (4.11)$$

Para cada uma destas componentes, podemos aplicar algumas transformações algébricas e substituir em $f(u) \leq CS$ para chegarmos a uma expressão da forma:

$$U + Vt \leq 0 \tag{4.12}$$

em t. Seja T o conjunto de valores de t para os quais a desigualdade 4.12 se mantém. Daí se tira um T' em função de u para o qual $f(u) \leq CS$. Podemos então dizer que $U := V \cap T'$ é a subcaixa (que pode até ser vazia) onde $f(u) \leq CS$. Reduzimos então a subcaixa V a U sem perda de otimalidade. Um processo análogo pode ser feito usando a expansão intervalar de Taylor de segunda ordem para $f(u)$. Mais detalhes podem ser encontrados em [31, Seção 9.5].

b) Redução pelo método de Newton intervalar: O método de Newton intervalar é uma combinação do método de Newton clássico, do teorema do valor médio e da aritmética intervalar. Ele pode ser usado, por exemplo, para resolução de sistemas de equações não-lineares ou para obter-se vizinhanças mais estreitas ao redor de pontos críticos de funções num problema de otimização restrita. É possível ainda, como vamos ver, decidir sobre a existência e/ou unicidade ou não de tais pontos em um dado intervalo.

Sejam $f : X \in \mathbb{I}^n \to \mathbb{R}^n, x' \in \mathbb{I}^n, x' \in X$ e x^* tal que $f(x^*) = 0$. Pelo teorema do valor médio temos: $0 = f(x^*) = f(x') + f'(\xi) * (x^* - x')$, para $\xi \in X$ onde f' é a matriz jacobiana da função f. Podemos substituir f' por uma extensão intervalar F' derivada de f e ξ por X e fazer $t = x^* - x'$ para obter:

$$F'(X) \times t + f(x') = 0 \tag{4.13}$$

ou seja, um sistema de equações intervalares em t. Seja T o conjunto de soluções t tal que t é solução para a equação 4.13. Mas $t = x^* - x'$, logo, podemos definir $T' = T + x'$. Desta forma, T' conterá todos os x^* que satisfazem $f(x) = 0$. Denotamos por $N(f, X, x') = T'$.

A figura 4.5 ilustra aplicações repetidas de iterações do método de Newton a um intervalo X que possui apenas um ponto-crítico.

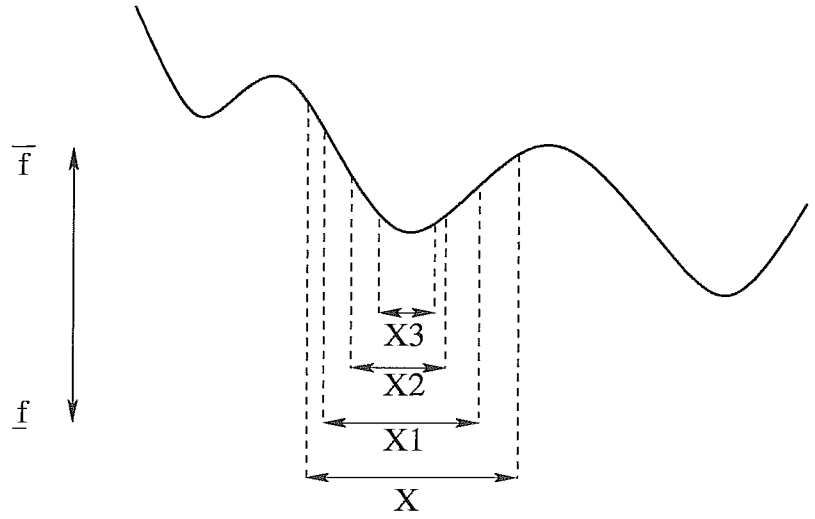

Figura 4.5: Reduções sucessivas pelo método de Newton intervalar

As vantagens desse método são a convergência quadrática (em determinadas condições) e as seguintes propriedades:

- Já que $N(F, X, x')$ contém todos os x^*, se $N(F, X, x') \subset X = \emptyset$ então podemos garantir que não existe um ponto crítico em X (isso permite eliminar o intervalo original X);
- Se $N(F, X, x') \subset int(X)$, então há uma única solução para $f(x)$ dentro de X (isso possibilita aplicar um método de busca mais eficiente no intervalo $N(F, X, x')$ para encontrar a solução);
- Se $N(F, X, x') \not\subset int(X)$, então pode-se garantir que, se houver solução em X, ela deve estar em $X \cap N(F, X, x')$ (isso permite reduzir o intervalo original X).

As duas desvantagens principais desse método são a necessidade de derivadas de f e o custo computacional para achar T que consiste na aplicação das versões intervalares do método de Gauss-Seidel ou da eliminação de Gauss. Esse último motivo, leva a aplicação dessa regra apenas nos casos mais promissores. Remetemos o leitor a [31, 42, 43] para saber mais sobre o assunto.

4.10 Desempenho

O desempenho de um algoritmo *branch-and-bound* pode ser medido através de vários parâmetros:

- tempo de CPU;
- número de subproblemas processados;
- número de subproblemas gerados;
- tamanho máximo alcançado por L;
- número de avaliações das funções f, F, g, G, h e H.

A comparação desses parâmetros referentes ao momento em que se encontra aquela que virá a ser a solução ótima e para toda a execução pode dar uma noção da dificuldade existente para se encontrar a solução e para provar que esta é ótima.

Capítulo 5

Branch-and-bound Paralelo

5.1 Considerações Iniciais sobre Paralelismo

Dentre as vantagens que o paralelismo oferece está a possibilidade de resolução de instâncias maiores que aquelas que um algoritmo seqüencial suporta devido à menor quantidade de recursos (memória, capacidade de processamento, etc) que um único computador possui. O paralelismo não altera a dificuldade de resolução (complexidade) dos problemas que o empregam pois a aceleração obtida pelo paralelismo não se equipara à complexidade geralmente exponencial de tais problemas.

O uso de vários processadores para resolver um problema permite distribuir o algoritmo de diversas maneiras e requer mais cuidados durante a implementação.

5.2 Conceitos Básicos e Classificação

Nesta seção introduziremos definições (algumas básicas) de termos que serão usados no decorrer do texto.

5.2.1 Definições

- **Máquina/computador** se refere genericamente aos equipamentos eletrônicos conhecidos como computadores (incluindo processador(es), memória, periféri-

cos, etc).

- Cada elemento capaz de processar dados armazenados em alguma porção de memória será chamado **processador**.

- **Tarefa** diz respeito ao programa (algoritmo) sendo executado por um processador. Como cada processador idealmente executa uma tarefa, esses dois termos podem ser empregados indistintamente para se referir ao processamento executado por um processador.

- Máquinas paralelas que possuem uma porção única de memória acessível por todos os processadores são ditas de **memória compartilhada**. Nas máquinas de **memória distribuída** cada processador possui sua porção de memória acessível apenas localmente e estes trocam informações através de mensagens enviadas por uma rede de comunicação.

- **Algoritmos síncronos** são aqueles onde é garantido, através de um mecanismo de temporização (relógio) global (único e acessível por todos os processadores), que todos os processadores estarão executando o mesmo passo do algoritmo a cada instante.

- **Algoritmos assíncronos** permitem que processadores diferentes estejam em pontos diferentes do algoritmo, exatamente pela ausência de uma forma de garantir o contrário. Este paradigma, por ser mais flexível, requer o tratamento de problemas que podem ocorrer, tais como: múltiplos acessos à mesma informação, incerteza na ordem em que a comunicação entre processos ocorre, divulgação de informação, etc.

- **Algoritmo mestre-escravo**: paradigma de programação assíncrona onde um dos processos gerencia a execução dos outros possuindo, assim, conhecimento global sobre o andamento de todo o algoritmo paralelo.

- A lista de subproblemas ativos L (ver seção 2.5) pode crescer demais a ponto de impedir o processamento em um determinado processador. Por esse motivo, considera-se a possibilidade de distribuir esta lista entre vários processadores disponíveis fazendo cada um destes manter localmente uma parte

de L. Algoritmos que mantêm essa lista centralizada são ditos terem **lista única**, enquanto que algoritmos que a mantêm distribuída são ditos terem **múltiplas listas**.

- **Unidades de trabalho**: porção mínima de dados que será tratada por um processador em cada passo do algoritmo. No nosso caso, ela corresponde a uma divisão de um subproblema incluindo o cálculo dos limites inferiores dos subproblemas gerados. O próprio termo subproblema muitas vezes pode sugerir a mesma idéia.

- **Subproblema básico ou primário**: todo aquele que é gerado pelo algoritmo *branch-and-bound* seqüencial

Podemos encontrar alguns exemplos desses conceitos na literatura sobre algoritmos *branch-and-bound*, Por exemplo: [55] descreve um algoritmo assíncrono, distribuído e de lista única; [86] e [44] trazem algoritmos mestre-escravo, também de lista única, assíncronos com passagem de mensagens e [88] e [11] propõem algoritmos assíncronos de múltiplas listas, distribuídos e com troca de mensagens.

5.2.2 Classificação

Em [24], os autores classificam algoritmos *branch-and-bound* paralelos em três categorias básicas segundo seu desenho:

- **Paralelismo tipo 1**: consiste em realizar as operações em cada unidade de trabalho de forma distribuída (usando vários processadores para realizar cada operação básica). Acelera-se a execução na medida em que diminui-se o tempo para processar cada subproblema. Por outro lado, não altera a estrutura geral do algoritmo *branch-and-bound*, ou seja, o algoritmo paralelo vai tratar os mesmos subproblemas na mesma ordem que o algoritmo seqüencial. Adequado para problemas onde as unidades de trabalho demandam muito processamento.

- **Paralelismo tipo 2**: consiste em construir a árvore *branch-and-bound* de maneira distribuída entre os processadores de forma que eles tratem independentemente os subproblemas atribuídos a si. Por esse motivo, o algoritmo paralelo possivelmente (e provavelmente) não vai tratar subproblemas na mesma ordem em que o algoritmo seqüencial o faria.

- **Paralelismo tipo 3**: consiste em gerar várias árvores *branch-and-bound* em paralelo, cada árvore se caracteriza por uma operação diferente (divisão, seleção, limitação, etc) e a informação gerada na construção de uma árvore pode ser (e é) usada na construção das outras.

Em [87], encontramos uma taxonomia detalhada para algoritmos branch-and-bound. Em [18] e [24] também se apresentam modelos mais completos de classificação de algoritmos branch-and-bound paralelos que o mostrado aqui.

5.2.3 Anomalias

Uma diferença marcante entre os algoritmos paralelos e seqüenciais é que, cada processador executando a versão paralela tem acesso a apenas uma parte da base de conhecimento (subproblemas, CS, etc) gerada pelo algoritmo como um todo. Isso faz com que os processadores tomem decisões diferentes daquelas que seriam tomadas pelo algoritmo seqüencial.

Definamos $T(k)$ o tempo gasto para resolver uma dada instância de um problema aplicando um algoritmo paralelo a k processadores. Usaremos $T(1)$ para representar tempo de execução do algoritmo seqüencial com o qual será comparada a versão paralela.

O conceito de **paralelização perfeita** [84] é baseado em hipóteses de divisibilidade, independência e interação pouco realistas. Mas pode ser usado para caracterizar o comportamento de algoritmos paralelos. Define-se $TPP = T(1)/k$ o tempo de paralelização perfeita e **anomalia** o comportamento que foge desse "comportamento ideal". Alguns poucos testes mostram que o que normalmente occorre é a **anoma-**

lia com prejuízo [1] ($T(1)/k \leq T(k) \leq T(1)$). Outras anomalias são: **anomalia desacelerativa** ($T(k) > T(1)$) e **anomalia acelerativa** ($T(k) < T(1)/k$). Além disso, devido ao não-determinismo, o algoritmo paralelo pode gastar tempos diferentes em execuções distintas para resolver a mesma instância. Quando a diferença entre esses tempos varia muito, estamos diante de uma **anomalia de flutuação**. É importante ressaltar que estamos comparando um algoritmo determinístico com um não-determinístico.

Vários autores (*c.f.*, [85, 84, 52, 55]) já estudaram essas anomalias e determinaram algumas propriedades que o *branch-and-bound* paralelo deve apresentar para evitar a ocorrência de anomalias desaceleratívas (permitindo ainda a aceleração). Essas propriedades são:

- função de busca heurística (ver 4.5) deve ser injetiva e
- função de busca heurística deve ser não-enganosa (ver Eq. 4.3).

As três condições a seguir decorrem dessas propriedades. As duas primeiras garantem que o algoritmo paralelo não irá executar num tempo superior ao do algoritmo seqüencial. E a última permite que aceleração do algoritmo ocorra.

- se durante a execução do algoritmo paralelo não restarem subproblemas primários na ou em nenhuma das listas L, então o algoritmo já deve estar de posse da solução ótima e pode terminar (o conhecimento gerado pelo *branch-and-bound* paralelo deve ser um superconjunto do conhecimento gerado pelo *branch-and-bound* seqüencial);
- em cada ponto da execução do algoritmo paralelo, pelo menos um subproblema primário deve estar sendo processado e
- a eliminação de subproblemas primários é possível durante a execução paralela do algoritmo.

[1] Do inglês: detrimental, significa danoso, nocivo, e no nosso caso deve ser entendido como *com prejuízo* já que o tempo de execução é maior do que TPP.

Se o algoritmo fizer uso de regras de dominância, mais algumas propriedades precisam ser colocadas para garantir essas condições. Propriedades estas que restringem as regras de dominância. [84] demonstra todas essas propriedades e [85] é um artigo bem explicativo sobre o assunto. [17] resume este último.

5.3 Particularidades Oriundas do Paralelismo

Assim como todo algoritmo *branch-and-bound* se caracteriza pelas quatro regras propostas por Mitten [61], um algoritmo branch-and-bound paralelo deve definir as seguintes estratégias:

5.3.1 Geração e Alocação de Unidades de Trabalho

Geralmente o algoritmo *branch-and-bound* inicia com um número de unidades de trabalho menor que o número de processadores. Por esse motivo, algum tempo será gasto decidindo que parte do problema inicial será entregue a que processador. Isso tem sido feito de várias maneiras, mas a estratégia geral citada em [24] pode ser expressa como: "*Use todos os processadores o mais cedo possível evitando, no entanto, entregar apenas subproblemas não promissores (que não têm uma boa chance de conter uma solução) para qualquer um dos processadores*". Isso significa que, mais vale esperar e distribuir igualmente a probabilidade de sucesso entre os processadores do que não gastar o tempo necessário e obter um algoritmo cuja eficiência vai depender pesadamente da estratégia de balanceamento adotada.

5.3.2 Aplicação da Segunda Regra de Eliminação

Como foi visto na seção 4.8, a segunda regra de eliminação diz respeito à eliminação de subproblemas que não produzem soluções melhores que uma cota superior (CS) conhecida até o momento. Manter o CS como variável local de cada processador apenas provoca uma ineficiência desnecessária no algoritmo pois a melhor cota superior (conhecida por apenas um dos processadores em cada momento) permitiria

a eliminação de mais subproblemas caso fosse conhecida pelos outros processos. A solução para isso é manter um esquema de divulgação periódico onde cada processador pode compartilhar um novo CS local encontrado e todos os processadores tomariam conhecimento do menor CS encontrado por todos eles em um dado instante.

Esse não chega a ser um problema em arquiteturas de memória compartilhada (já que todos os processadores têm acesso à mesma posição de memória onde se guardaria um único valor para CS). Já em arquiteturas de memória distribuída, esquemas de divulgação são necessários e dentre eles podemos citar:

a) Periodicamente, os processadores executam um algoritmo de divulgação de informação (*c.f.* [5]) após o qual todos os processadores conhecem o menor dos CS. Desvantagem: induz pontos de sincronização que podem diminuir a eficiência em sistemas assíncronos.

b) Ao encontrar um novo CS, um processo divulga este para todos os outros processadores que o aceitarão se o CS recebido for menor que o CS local. Esse esquema é chamado em [87] de *compartilhamento de conhecimento completo* pois todos os processadores conhecem o melhor CS. No caso de gargalo na comunicação, pode-se optar por não compartilhar o conhecimento logo que é gerado (*compartilhamento de conhecimento completo atrasado*).

c) Outro esquema pode garantir que o novo CS é enviado apenas para os processadores topologicamente vizinhos daquele que gerou o novo dado (compartilhamento de conhecimento parcial ou local [87]). Como admite-se que todos os processadores são ligados entre si mesmo que indiretamente, ainda é possível que o melhor CS chegue a qualquer processador ainda que leve mais tempo.

5.3.3 Alocação e Compartilhamento de Unidades de Trabalho

O balanceamento da quantidade de unidades de trabalho com que cada processador vai trabalhar (balanceamento de carga) pretende garantir que nenhum dos proces-

sadores fique ocioso por falta de unidades de trabalho (*starvation, c.f.* [5]) enquanto outro esteja sobrecarregado, além de mantê-los como na seção 5.3.1: com chances iguais de encontrarem uma boa solução. Dentre as várias estratégias de balanceamento podemos citar:

a) Manter todos os subproblemas num só processador (lista única) facilitando a implementação do balancemaneto mas limitando a quantidade de unidades de trabalho à quantidade de memória deste processador.

b) Manter múltiplas listas de subproblemas o que exigirá que alguns destes sejam transferidos da lista de um processador para a de outro de tempos em tempos. Esse tipo de estratégia é denominada *alocação dinâmica.* Ela possui variações em relação ao momento em que os subproblemas são transferidos:

   - com pedido: um processo ocioso pede mais subproblemas,
   - sem pedido: um processo muito ocupado envia subproblemas e
   - combinação: ambas as situações anteriores podem acontecer.

[72] descreve uma taxonomia para estratégias de balanceamento de carga e [6] descreve um paradigma híbrido de processamento que incorpora esquema de balanceamento e divulgação de CS que será explicado mais adiante. [56] apresenta um esquema de balanceamento de carga com realimentação (feedback).

5.3.4 Detecção de Término

Algoritmos assíncronos possuem um problema a mais para resolver que é: cada processador saber que todos os outros já acabaram seu processamento. Nos algoritmos assíncronos, mesmo que todos os processadores tenham esvaziado suas respectivas listas de subproblemas ativos, por causa do balanceamento de carga, pode ainda haver transferências de unidades de trabalho em andamento o que significa que pelo menos um processador ainda vai ter unidades de trabalho para processar. Pode-se aplicar algoritmos genéricos para detectar a terminação de algoritmos distribuídos

como os descritos em [5], algum esquema centralizado (mestre-escravo) ou mesmo específicos da aplicação que podem diminuir a complexidade do algoritmo.

[77] faz várias considerações sobre algoritmo *branch-and-bound* intervalar paralelo.

5.4 Medição de Desempenho

A medição de desempenho de algoritmos distribuídos é particularmente difícil pois existem vários critérios muitas vezes independentes uns dos outros e que conduzem a resultados incompatíveis. Como por exemplo:

a) Qualidade da solução: em algoritmos aproximativos, aquele que encontrar uma solução mais próxima da solução ótima que um outro algoritmo é considerado ter melhor desempenho.

b) Número de problemas gerados: aquele algoritmo que gerar menos subproblemas é considerado mais eficiente. Existem algumas variantes aqui:

   - considerar o número de subproblemas total gerado;
   - considerar apenas o número de subproblemas gerado até encontrar o ótimo;
   - para uma mesma estratégia de seleção, o número máximo de elementos na lista de subproblemas abertos.

   Algumas desvantagens são: não mede trabalho gasto com paralelismo (divulgação do CS, envio/recepção de subproblemas, etc), pode ser difícil avaliar em algoritmos assíncronos ou branch-and-bound tipo 3, além do quê, essa medida pode variar muito em algorimos não-determinísticos.

c) Aceleração e eficiência: Essa medição é feita com relação ao tempo. Seja T(p) o tempo necessário para resolver o problema com $p \geq 1$ processadores. Definimos aceleração (*speedup*) como A(p) = T(1)/T(p) e eficiência E(p)=A(p)/p.

Essas medidas já levam em consideração o esforço gasto no paralelismo, mas elas possuem questões não resolvidas tais como:

- Como medir o tempo? Como evitar que o sistema interfira no desempenho do algoritmo distribuído?
- T(1) se refere a quê? Ao tempo do algoritmo paralelo executado em apenas um processador? Ou ao tempo do melhor algoritmo seqüencial? Qual seria esse algoritmo?

Uma variação seria medir o número de avaliações de funções objetivos (no nosso caso, CF e suas derivadas, ver seção 1.4). Isso daria alguma independência da medição sobre o tempo e a carga do sistema, mas não sobre a questão de T(1).

Além disso, os conceitos de aceleração e eficiência não consideram necessidade de sincronia, admitem tempo constante para cada comunicação e admitem que os processadores possuem mesmo poder computacional (*c.f.*[84]).

Capítulo 6

Algoritmo

Neste capítulo descreveremos as regras escolhidas para implementação do *branch-and-bound* (tanto seqüencial quanto paralelo) bem como algumas melhorias incorporadas à implementação para torná-la mais eficiente.

6.1 Componentes do Algoritmo Seqüencial

A fim de não gastar esforço com problemas de implementação que são importantes para o algoritmo proposto, mas que fogem do objetivo do trabalho, usamos um pacote disponível publicamente chamado VerGO [38] que implementa um algoritmo *branch-and-bound* intervalar (possui sua própria biblioteca de funções intervalares). Deve ser dito aqui que a escolha desse pacote se deu pela adequação ao trabalho, já que ele oferecia uma versão do algoritmo *branch-and-bound* testado (embora não exatamente aquele definido por [31], mas que poderia ser modificado) e que oferecia a facilidade da sintaxe natural dos operadores intervalares (*e.g.* $z = x + y$ ao invés de $SUM(x, y, z)$). Outra observação a ser feita é a de que o código-fonte é pouco documentado e por vezes sua lógica se tornou quase ilegível (ou ilógica!), fato este que não compromete a corretude do algoritmo.

Dito isso, podemos seguir para a definição dos componentes do algoritmo *branch-and-bound* usado tanto no que se refere à implementação seqüencial quanto à distribuída, além de especificar os mecanismos adicionados ao método.

6.1.1 Regra de Seleção

As duas estratégias mais usadas na literatura são *Best-first* e *Depth-first*. A nossa idéia é aproveitar as vantagens de cada estratégia através de uma estrutura de dados híbrida que possa se comportar ora como *heap* (*Best-first*), ora como pilha (*Depth-first*) (ver [12]). Desa forma, podemos mudar a estratégia em vigor durante a execução do *branch-and-bound*.

Numa implementação seqüencial, essa estrutura permite que a estratégia seja mudada em determinado momento dado que alguma situação seja atingida onde outra estratégia seja recomendada. Por exemplo: pode-se começar a execução usando a estratégia *Best-first*, gera-se vários subproblemas e, à medida que se garanta que um dado subproblema possui apenas um mínimo local (teste de concavidade na seção 4.8), podemos usar a estratégia *Depth-first* para chegar a uma subcaixa dentro da tolerância pré-definida que contenha o mínimo e assim eliminar toda aquela região e possivelmente obtendo uma cota superior baixa mais rapidamente que a estratégia anterior. Outra idéia seria começar usando *Depth-first* para tentar conseguir uma boa cota superior no início da execução de forma que, na fase subseqüente, a estratégia *Best-first* mais subproblemas seriam eliminados implicitamente. A alternância entre estratégias pode ser feita sempre que conveniente.

Numa implementação distribuída, essa estratégia híbrida pode ser usada para diversificar a atuação dos processos fazendo com que uns procurem diminuir a cota superior atual usando a estratégia *Depth-first* enquanto outros procuram o mínimo mais efetivamente usando *Best-first*. Note-se que, mesmo nesse caso, ainda temos uma única árvore *branch-and-bound* (ver classificação na seção 5.2.2 segundo [24]) o que nos garante que todos os subproblemas ativos representam regiões não intersectantes do espaço de busca. Isso nos exime da preocupação de que processos distintos estejam operando numa mesma região do espaço (como acontece em algoritmos do tipo 3).

Aproveitando a estrutura híbrida que permite uso de mais de uma estratégia de seleção, podemos incluir, sem muito esforço, a estratégia Índice de Rejeição (seção

4.5.c), que também pede uma estrutura tipo *heap*. A inclusão de *Breadth-First* (seção 4.5.d) também é possível mas não foi considerada pelo pouco interesse na estratégia.

6.1.2 Regra de Divisão

Como regra de divisão, usamos a regra B (4.6) pois, apesar da literatura mostrar que as regras B e C apresentam desempenhos compatíveis, a regra C (4.6) possui um produto por um vetor intervalar, o que torna o teste mais caro computacionalmente.

6.1.3 Regra de Eliminação

Como regra de eliminação, usamos o teste do limite superior ($\underline{F}(X) > CS$ elimina subcaixa), teste de monotonicidade ($0 \notin G(X)$ elimina subcaixa) e concavidade ($\exists i \mid H_{ii}(X) < 0$ elimina subcaixa). Usamos uma versão relaxada do teste concavidade por motivos de custo computacional como comentado na seção 4.8. Não foi aplicado nenhum teste de dominância.

6.1.4 Regra de Redução

Os métodos de redução mencionados na seção 4.9 são os sugeridos em [31]. No VerGO, estão disponíveis, além do método de Gauss-Seidel, algumas rotinas de busca local (Gradiente Conjugado, Newton Truncado [19] e duas versões do método BFGS) que são usadas para reduzir uma subcaixa convexa a uma vizinhança do mínimo local.

O algoritmo da figura 6.1 apresenta um esboço do algoritmo seqüencial do *branch-and-bound* implementado pelo VerGO.

```
01 inicia valor de L, L_c, L̄, CS
02 enquanto (L ≠ ∅ and L_c ≠ ∅)
03     enquanto (L ≠ ∅)
04         X := próxima caixa de L
05         se (F(X) < CS)
06             se (divide(X,B_1,B_2) < tol)
07                 adiciona X a L̄
08             senão
09                 se não conseguir eliminar B_1 e/ou B_2
10                     adicione B_1 e/ou B_2 a L
11     fim enquanto
12     aplica teste limite e algoritmos de redução a L_c
13     enquanto (L_c ≠ ∅)
14         X := próxima caixa de L_c
15         se (w(X) < tolcaixa)
16             adiciona X a L
17         senão
18             para cada vetor intervalar V(i) contendo a vizinhança das bordas de X
19                 se (0 ∈ G(V(i)))
20                     adiciona V(i) a L
21         fim se
22     fim enquanto
23 fim enquanto
```

Figura 6.1: Algoritmo básico do VerGO (seqüencial)

6.2 Algoritmo Paralelo

A seguir, detalharemos os aspectos de paralelização do algoritmo *branch-and-bound*. Para efetuar a comunicação entre os processos foi feito uso da biblioteca de funções chamada MPI (*Message Passing Interface*) [82]. Esta biblioteca foi criada para promover o desenvolvimento de aplicações paralelas e distribuídas fornecendo um padrão que fosse ao mesmo tempo eficiente e portável. Os esforços de padronização começaram em 1992 e, desde então, o MPI vem se tornando cada vez mais popular contando com várias implementações tanto comerciais quanto de domínio público. A implementação utilizada (MPICH) é pública e está disponível em http://www.mcs.anl.gov/mpich/download.html.

Adotamos, como já foi mencionado, um esquema proposto em [6], onde um dos

processos centraliza algumas informações e responsabilidades, mas de forma que não se torne um gargalo para o processamento como um todo. Esse processo será chamado **mediador** enquanto os outros processos que implementam o *branch-and-bound* propriamente dito serão chamados **trabalhadores**.

6.2.1 Geração e Alocação de Unidades de Trabalho

O processo mediador inicia a execução dividindo a caixa inicial (região viável) em cada uma das suas dimensões até conseguir um número de subcaixas igual ao número de processos e então envia uma caixa para cada um deles.

6.2.2 Aplicação da Segunda Regra de Eliminação

Com certa freqüência (a ser definida de forma que não provoque excesso de comunicação e pouco benefício), os processos enviam ao mediador suas cotas superiores (CS) assim como recebem deste o melhor (menor) valor conhecido por todos em um dado momento. Desta forma, todos os processos podem usufruir do conhecimento global do algoritmo e eliminar subcaixas mais rapidamente do que se fosse esperar ele próprio (um processo isolado) chegar a essa cota superior. O processo mediador simplesmente mantém a menor cota superior enviada para ele e, de tempos em tempos divulga a informação para os trabalhadores (compartilhamento de conhecimento completo atrasado). Esse intervalo de tempo pode ser abreviado caso a variação no valor da cota superior seja considerável.

6.2.3 Alocação e Compartilhamento de Unidades de Trabalho

Responsável pelo balanceamento de carga, o mediador monitora a carga (tamanho da lista L ($|L|$)) de cada trabalhador e coordena a transferência desse excesso de um trabalhador para outro(s) através de uma variável que indica o número máximo (MAX) de caixas que cada trabalhador deve ter. De tempos em tempos, o mediador avalia a informação sobre carga recebida dos trabalhadores, determina um novo valor

para MAX e o divulga. Cada trabalhador, ao receber o novo valor, decide se está em excesso ou não, devolvendo este ao mediador.

A decisão de quantas caixas devem ser enviadas ao mediador é simples, mas a de quais caixas, não. Tanto enviar as caixas mais promissoras [1] (início de L) quanto as menos (fim de L) podem não diminuir o desbalanceamento pois algum trabalhador (o que envia essas caixas no primeiro caso e o que recebe no segundo) pode ficar com caixas pouco promissoras o que consiste em má utilização do processamento daquele processador, *i.e.* seu processamento não estará contribuindo para procurar a solução. Um esquema de escolha simples é tomar caixas dentre as de ordem par em L. Resolvemos escolher as caixas de maneira uniforme dividindo o resto da lista L em $|L| - MAX - 1$ partes e selecionando uma caixa de cada (ver exemplo da figura 6.2).

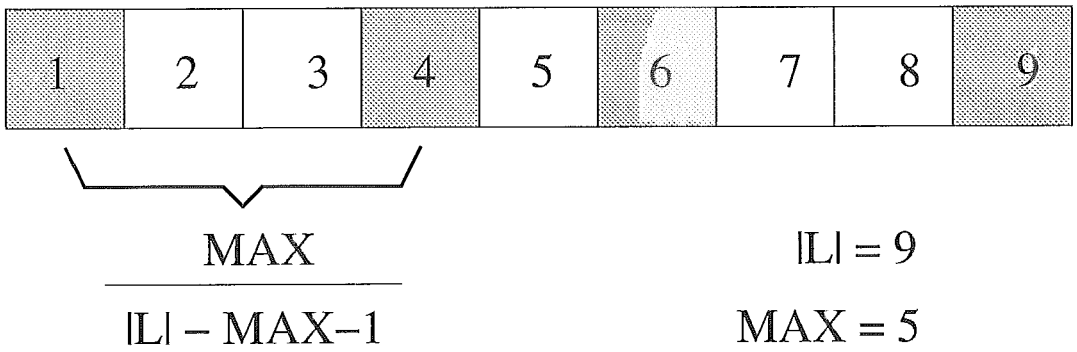

Figura 6.2: Exemplo de caixas excedentes selecionadas para envio ao mediador

Decidido quantas e quais caixas serão enviadas, o mediador se encarregará de recebê-las, guardá-las, testá-las e redistribuí-las quando necessário.

O valor de MAX não é estático e deve ser decidido pelo mediador com base na carga atual dos trabalhadores ($|L|$) bem como no tamanho da lista do mediador. Resolvemos levar em consideração um fator de correção na distribuição que permita enviar mais caixas aos processadores que possuam mais recursos (memória, por exemplo).

Para dar maior flexibilidade ao balanceamento, resolvemos adotar o seguinte

[1] O fato de uma caixa ser promissora ou não tem maior influência nas estratégias de seleção adaptativas

esquema: Dados MEM_i (tamanho da memória do processador i), $CARGA_i$ ($|L|$ no processador i) e N (o número de processos, mediador incluído), definimos:

$$\text{Utilização média } (UTmd) := \frac{\sum_{i=1}^{N} CARGA_i}{\sum_{i=1}^{N} MEM_i} \tag{6.1}$$

$$\text{Utilização ideal } (UTid_i) := MEM_i \times UTmd \tag{6.2}$$

$$\text{Desvio } (DSV) := \frac{\sum_{i=1}^{N} UTid_i - CARGA_i}{N} \tag{6.3}$$

O objetivo é manter DSV o mais baixo possível. Para isso, o mediador envia $UTmd$ para todos os trabalhadores de modo que cada um deles calcule sua $Utid$ e decida sobre sua situação quanto ao balanceamento. Para evitar que os processos iniciem muitas comunicações tentando chegar ao balanceamento idealizado pelo mediador, uma tolerância (TOL) estipulada por este mesmo processo e enviada aos trabalhadores faz com que estes tomem as decisões quando o excesso não estiver dentro da tolerância ($\frac{(CARGA_i - UTid_i)}{CARGA_i} > TOL$).

Note que a estratégia de reação a alteração de conhecimento desse esquema de balanceamento de carga ocorre tanto por demanda (o mediador interrompe a execução do trabalhador para enviar-lhe informações que o farão tomar decisões) quanto por pedido (trabalhador com lista vazia espera mais caixas do mediador).

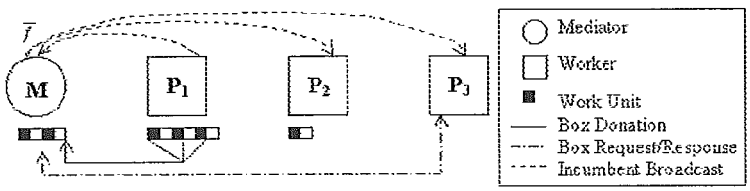

Figura 6.3: Esquema explicativo da comunicação entre mediador e trabalhadores

6.2.4 Detecção de Término

No esquema aqui descrito, a tarefa de detectar a terminação global do algoritmo é relativamente simples e não requer muito processamento. Já que um trabalhador só pede mais caixas para o mediador quando sua lista está vazia, a única fonte direta de subproblemas do trabalhador é o mediador e, ele (o trabalhador) não faz mais nada enquanto não receber uma resposta do mediador sobre o pedido de mais subcaixas, então podemos garantir que não há nenhuma mensagem pendente que pudesse alterar a situação desse trabalhador. Quando todos os trabalhadores estão esperando caixas do mediador e esse também está vazio, ele "sabe" que o algoritmo terminou e pode informar esse evento para os demais processos enviando zero caixas como resposta ao pedido de cada trabalhador.

As figuras 6.4 e 6.5 mostram os algoritmos usados para o mediador e o trabalhador respectivamente.

```
01 inicia valor de L, L_c, L̄, CS
02 enquanto (#pedidos < #processos − 1)
03     recebe qualquer mensagem
04     caso (tipo da mensagem seja)
05         PEDIDO: anote novo pedido na lista de pedidos
06         CS:     se (CS_local > CS_recebido)
07                     CS_local := CS_recebido
08                 senão
09                     envia CS_local
10         NUM:    recebe tamanho e lista caixas para incluir em L
11         CARGA:  recebe informação da carga de um trabalhador
12     se (L ≠ ∅)
13         para cada pedido pendente
14             envia caixas para trabalhador
15     aplica teste de eliminação em L
16     se (carga dos trabalhadores mudou)
17         recalcula UTmd e TOL e envia para todos os trabalhadores
18 fim enquanto
19 envia zero caixas para todos os trabalhadores
```

Figura 6.4: Algoritmo do processo mediador

```
01 inicia valor de L, L_c, L̄, CS
02 repita
03     enquanto (L ≠ ∅  and L_c ≠ ∅)
04         enquanto (L ≠ ∅)
05             . . .
06         envia CS para mediador
07         possivelmente recebe CS, UTmd e TOL do mediador
08         aplica teste limite e algoritmos de redução a L_c
09         enquanto (L_c ≠ ∅)
10             . . .
11         se UTmd e TOL indicam que processo está com excesso
12             envia excesso de caixas para mediador
13     fim enquanto
14     envia pedido de caixas para mediador
15     recebe tamanho e lista de caixas
16     inclui caixas em L
17 até que tamanho da lista recebida seja zero
```

Figura 6.5: Algoritmo do processo trabalhador

Capítulo 7

Conclusão

Problemas de Otimização Global, como por exemplo o de conformação molecular (PCM), são de difícil resolução, principalmente quando não se deseja apenas uma resposta aproximada.

Neste trabalho, aplicamos um método *branch-and-bound* intervalar paralelo na busca do mínimo global de uma função-teste bem conhecida (CF). O algoritmo paralelo apresentado é classificado como do tipo 2 por [24] e apresenta as seguintes características principais:

- a utilização de uma estratégia híbrida de seleção do próximo subproblema escolhido onde a estratégia pode alternar entre as 3 escolhidas (duas das quais são mais utilizadas na literatura) durante a execução do algoritmo;
- o compartilhamento de informação é dinâmico e ocorre tanto por demanda quanto por pedido;
- o balanceamento de carga segue o esquema descrito em [6] (mediador $\times$ trabalhador). Dentre suas vantagens estão:
  - dinâmico (se adapta de acordo com o andamento do algoritmo);
  - decisões não são totalmente centralizadas (mediador não é gargalo de comunicação);
  - transferências de unidades de trabalho tem custo fixo devido à topologia simplificada (mediador $\longleftrightarrow$ trabalhador);

  - oferece poucos pontos de sincronização;
  - facilita a detecção de término distribuído.

- desenhado para suportar ambientes heterogêneos (carga pode ser distribuída de acordo com os recursos de cada processador).

Referências Bibliográficas

[1] Aarts, E.H.L., Lenstra, J.K. "Local search in combinatorial optimization". John Wiley & Sons, Chichester, 1997.

[2] Aho, A., Hopcroft, J., Ullman, J. "Data Structures and Algorithms", 1983.

[3] Alefeld, G., Herzberger, J. *Introduction to Inteval Computations.* Academic Press, New York, 1983.

[4] Barbosa, H., Raupp, F., Lavor, C., Lima, H., Maculan, N. "A hybrid genetic algorithm for finding stable conformations of small molecules". In *Proc. of the VIth Brazilian Symposium on Neural Networks*, pp. 90–94. IEEE Computer Society Press, Los Alamitos, 2000.

[5] Barbosa, Valmir. *An Introduction to Distributed Algorithms.* The MIT Press, Cambridge, Massachusetts, 1996.

[6] Berner, S. "Parallel Methods for Verified Global Optimization Practice and Theory". *Journal of Global Optimization*, v. 9, pp. 1–22, 1996.

[7] Brodmeier, T., Pretsch, E. "Application of genetic algorithms in molecular modeling". *Journal of Computational Chemistry*, v. 6, n. 15, pp. 588–595, June 1994.

[8] C. D. Maranas, L. P. Androulakis, Floudas, C. A. "Adaptive Multisection In Interval Global Optimization". In *Global Minimization of Nonconvex Energy Functions: Molecular Conformation and Protein Folding*, pp. 133–150, Amer. Math. Soc. Providence, RI, 1996.

[9] Char, B. W. W., Geddes, K. O., Gonnet, G. H., Leong, B. L., Monagan, M. B., Watt, S. M. *Maple V Language Reference Manual.* Springer-Verlag New York, Incorporated, 1996.

[10] Chiriaev, D., Walster, G. *Fortran 77 Interval Arithmetic Specification*, 1997.

[11] Clausen, J., Traff, J. "Implementation of Parallel Branch-and-Bound Algorithms - Experiences with the Graph Partitioning Problem". In *Annals of Operations Research*, v. 33, pp. 331–349, 1991.

[12] Cormen, T.H., Leiserson, C.E., Rivest, R.L. *Introduction to Algorithms.* M.I.T. Press, Cambridge, Massachusetts, U.S.A., 1990.

[13] Csallner, A.-E., Csendes, T., Markót, M.-C. "Multisection in interval Branch-and-Bound methods for global optimization". *Journal of Global Optimization*, 1998.

[14] Csendes, T., Casado, L.G., Garcia, I. "Adaptive Multisection In Interval Global Optimization". In *Volume of Extended Abstract of SCAN-98*, pp. 27–29, Budapest, 1998.

[15] Csendes, T., Pintér, J.D. "The Impact of Accelerating Tools on the Interval Subdivision Algorithm for Global Optimization". *European Journalon Operational Research*, n. 65, pp. 314–320, 1993.

[16] Csendes, T., Ratz, D. "Subdivision Direction Selection in Interval Methods for Global Optimization". *SIAM Journal on Numerical Analysis*, v. 34, n. 3, pp. 922–938, 1997.

[17] de Bruin, A., Kindervater, G.A.P., Trienekens, H.W.J.M. "Asynchronous Parallel Branchand -Bound and Anomalies", 1995.

[18] de Bruin, A., Kindervater, G.A.P., Trienekens, H.W.J.M. "Towards an abstract parallel branch and bound machine". In Ferreira, A., Pardalos, P. (eds), *Solving Combinatorial Optimization Problems in Parallel: Methods and Techniques*, v. 1054, *Lecture Notes in Computer Science*, pp. 145–170. Springer-Verlag, Berlin, 1996.

[19] Dembo, R., Steihaug, T. "Truncated Newton method algorithms for large–scale unconstrained optimization". *Mathematical Programming*, v. 26, pp. 190–212, 1983.

[20] Diener, I. "Trajectory nets connecting all critical points of a smooth function". *Mathematical Programming*, v. 36, pp. 340–352, 1986.

[21] Dill, K.A., Phillips, A.T., Rosen, J.B. *Molecular structure prediction by global optimization.* Manuscript (phillips@nadn.navy.mil), 1996.

[22] Forster, W. *Homotopy Methods.* Kluwer, 1995.

[23] Garey, M.R., Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NPCompleteness W.* W.H. Freeman and Company, New York, 1979.

[24] Gendron, B., Crainic, T.G. "Parallel branch-and-bound algorithms: Survey and synthesis". *Operations Research*, v. 42, pp. 1042?1066, 1994.

[25] Glover, F., Laguna, M. *Tabu Search.* Kluwer Academic Publishers, Boston, 1997.

[26] Glover, F., Taillard, E., de Werra, D. "A user's guide to tabu search". *Annals of Operations Research*, v. 41, pp. 3–28, 1993.

[27] Goldberg, D.E. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company, Inc., Reading, MA, 1989.

[28] Grand, S. Le, Merz, K. "The application of genetic algorithms to the minimization of potential energy functions". *Journal of Global Optimization*, v. ?, n. 3, pp. 49–66, 1993.

[29] Gray, P., Hart, W., Painton, L., Phillips, C., Trahan, M., Wagner, J. *A Survey of Global Optimization Methods.* Technical report, Sandia National Laboratories Albuquerque, NM 87185, 1997.

[30] Hansen, E.R. "A generalized interval arithmetic". In *Interval Mathematics: Proceedings of the International Symposium*, pp. 7–18, Karlsruhe, West Germany, maio 1975. Springer-Verlag.

[31] Hansen, E.R. *Global Optimization Using Interval Analysis.* Marcel Dekker, New York, 1992.

[32] Hansen, P. "The steepest ascent mildest descent heuristic for combinatorial programming". In *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy, 1986.

[33] Holland, J.H. *Adaption in Natural and Artificial Systems.* MIT Press, 1992.

[34] Horst, R., Pardalos, P. M. *Handbook of Global Optimization*, v. 2. Kluwer Academic Publishers, Dordrecht; Boston, 1995.

[35] Horst, R., Tuy, H. *Global optimization: deterministic approaches.* Springer-Verlag, Berlin, 1993.

[36] Hyvonen, E., Pascale, S. De. "InC++ Library Family For Interval Computations". In *Reliable Computing, Supplement of Proceedings of Application of Interval Computations*, pp. 85–90, El Paso, Texas, 1995.

[37] Ibaraki, T. "The Power of Dominance Relations in Branch-and-Bound Algorithms". *Journal of the ACM*, v. 24, n. 2, pp. 264–279, 1977.

[38] Iwaarden, R.V. *Global Optimization using VerGO: Verified Global Optimization in C++.* http://www.cs.hope.edu/r̃vaniwaa.

[39] Kawai, H., Kikuchi, T., Okamoto, Y. "A prediction of tertiary structures of peptide by the Monte Carlo simulated annealing method". *Protein Engineering*, v. 3, pp. 85–94, 1989.

[40] Kearfott, R. B. "Interval computations: introduction, uses, and resources". *Euromath Bulletin*, v. 2, n. 1, pp. 95–112, 1996.

[41] Kearfott, R.B. "A Review of Techniques in the Verified Solution of Constrained Global Optimization Problems". In Kearfott, R.B., Kreinovich, V. (eds), *Applications of Interval Computations*, pp. 23–59. Kluwer, Dordrecht, Netherlands, 1996.

[42] Kearfott, R.B. *Rigorous Global Search: Continuous Problems.* Kluwer Academic Publishers Group, Norwell, MA, USA, 1996.

[43] Kearfott, R.B. *Interval Newton Methods.* http://interval.louisiana.edu/preprints/EoO/KEARFO01.ps, janeiro 1998.

[44] Kindervater, G.A.P. *Exercises in Parallel Combinatorial Computing.* PhD thesis, Centre for Mathematics and Computer Science, Amsterdam, 1989.

[45] Kirpatrick, S., C.D. Gelatt, Jr., Vecchi, M.P. "Optimization by Simulated Annealing". *Science*, v. 220, pp. 671–680, maio 1983.

[46] Klatte, R., Kulisch, U., Wiethoff, A., Lawo, C., Rauch, M. *C-XSC - A C++ Class Library for Extended Scientific Computing.*

[47] Knuppel, O. "PROFIL/BIAS - a fast interval library". *Computing*, v. 53, pp. 277–288, 1994.

[48] Knuppel, O. *A PROFIL/BIAS implementation of a global optimization algorithm.* Technical Report 95.4, Harburg, Hamburg University, Germany, Out 1995.

[49] Kohler, W.H., Steiglitz, K. "Characterization and Theoretical Comparison of Branch-and-Bound Algorithms for Permutation Problems". *Journal of the ACM*, v. 21, n. 1, pp. 140–156, 1974.

[50] L. B. Morales, J. M. Aguilar-Alvarado, R. Garduno-Juarez, Riveros-Castro, F. J. *A Parallel Tabusearch For Conformational Energy Optimization Of Oligopeptides.* Technical report, IF Cuernavaca, UNAM, Apdo. Postal 48-3, 62250, Cuernavaca, Morelos, Mexico, 1992.

[51] Laarhoven, P.J.M., Aarts, E.H.L. *Simulated Annealing: Theory and Applications.* D. Reidel Publishing, 1987.

[52] Lai, T., Sahni, S. "Anomalies in parallel branch-and-bound algorithms". *Communications of the ACM*, v. 27, pp. 594–602, 1984.

[53] Lavor, C. *Uma abordagem determinística para a minimização global da energia potencial de moléculas.* PhD thesis, COPPE/UFRJ, jun 2001.

[54] Levy, A., Gómez, S. "The Tunneling Method Applied to Global Optimization", 1985.

[55] Li, G., Wah, B. "Coping with anomalies in parallel branch-and-bound algorithms". *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, v. 35, pp. 568–573, 1986.

[56] Lüling, R., Monien, B. "Load balancing for distributed branch and bound algorithms". In *International Parallel Processing Symposium (IPPS)*, pp. 543–549, 1992.

[57] Maranas, C., Floudas, C. *A Global Optimization Approach for Lennard-Jones Microclusters.* Technical report, Department of Chemical Engineering, Princeton University, 1992.

[58] Maranas, C., Floudas, C. "A Deterministic Global Optimization Approach for Molecular Structure Determination". *J. of Chemical Physics*, v. 100, n. 2, pp. 1247–1261, 1994.

[59] Maranas, C., Floudas, C. "Global minimum potential energy conformations of small molecules". *Journal of Global Optimization*, v. 4, pp. 135–170, 1994.

[60] Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer, Berlin, 1992.

[61] Mitten, L.G. "Branch-and-bound methods: General formulation and properties". *Journal of Operations Research*, n. 18, pp. 315–344, 1970.

[62] Mockus, J., Eddy, E., Mockus, A., Mockus, L., Reklaitis, G., global, . *Bayesian discrete and global optimization.* Kluwer Academic Publishers, 1997.

[63] Mockus, J.B. "The Bayesian approach to global optimization". In *Statistics: applications and new directions*, pp. 405–430. Indian Statistical Institute, Calcutta, 1984.

[64] Moore, R.E. *Interval Arithmetic and Automatic Error Analysis in Digital Computing.* PhD thesis, Stanford University, 1962.

[65] Moore, R.E. "Methods and Applications of Interval Analysis". *SIAM Studies in Applied Mathematics*, 1979.

[66] Moore, R.E., Hansen, E.R., Leclerc, A.P. "Rigorous Methods for Parallel Global Optimization". *Recent Advances in Global Opimization*, pp. 321–342, 1992.

[67] Neumaier, A. *Interval Methods for Systems of Equations*, v. 37. Cambridge University Press, Cambridge, UK, 1990.

[68] Neumaier, A. "Molecular modeling of proteins: The mathematical prediction of protein structure". In *SIAM Reviews*, v. 39, pp. 407–460. SIAM, 1997.

[69] Ngo, J., Marks, J., Karplus, M. "Computational complexity, protein structure prediction, and the Levinthal paradox". In *In The Protein folding problem and structure prediction*, pp. 435–508. Birkhausen, 1994.

[70] Pintér, J.D. *Global Optimization in Action, Continuous and Lipschitz Optimization: Algorithms, Implementations and Applications.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1996.

[71] Pintér, J.D. *Continuous Global Optimization: An Introduction to Models, Solution Approaches, Tests and Applications.* Technical report, Pintér Consulting Services, and Dalhousie University, http://catt.bus.okstate.edu/itorms/pinter/index.html.

[72] Plastino, A., Ribeiro, C.C.C., Rodriguez, N.L.R. *Uma ferramenta para desenvolvimento de aplicações SPMD com suporte para balanceamento de carga.* PhD thesis, Departamento de Informatica, PUC-Rio de Janeiro, 1999.

[73] Ratschek, H., Rokne, J. *New Computer Methods for Global Optimization.* Ellis Horwood Ltd, Chichester, 1988.

[74] Ratz, D. *Automatische Ergebnisverifikation bei globalen Optimierungsproblemen.* PhD thesis, Universitaet Karlsruhe, 1992.

[75] Ratz, D. "Techniques For Gap Treating And Box Splitting In Interval Newton, Gauss-Seidel Steps For Global Optimization". In *ZAMM*, v. 76, pp. 323–226, 1996.

[76] Reeves, C.R., Yamada, T. "Genetic Algorithms, Path Relinking, and the Flowshop Sequencing Problem". *Evolutionary Computation*, v. 6, n. 1, pp. 45–60, 1998.

[77] Revol, N., Denneuliny, Y., Méhaut, J.-F., Planquellex, B. "Parallelization of continuous verified global optimization". In *19th Conference on System Modelling and Optimization*, pp. 128–131, july 1999.

[78] Santiago, Regivan H. N., Acioly, Benedito M. "Interval Local Equality". In *SCAN/2000*, Germany, 2000.

[79] Schulte, M.J., Jr., E.E. Swartzlander. "Software And Hardware Techniques For Acurate Self-Validating Arithmetic". In *Applications of Interval Compuations*, pp. 381–404, 1996.

[80] Sengupta, A., Pal, T.K. *On comparing interval numbers, Theory and Methodology.* Technical report, Department of Applied Mathematics woth Oceanology & Computer Programming, Vidyasagar University, 1999.

[81] Shin, J., Jhon, M. "High directional Monte Carlo procedure coupled with the temperature heating and annealing method to obtain the global energy minimum structure of polypeptides and proteins". *Biopolymers*, v. 31, pp. 177–185, 1991.

[82] Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J. *MPI: The Complete Reference*, 1997.

[83] Törn, A. A., Zilinskas, A. "Global Optimization". *Lecture Notes in Computer Science*, v. 350, 1989.

[84] Trienekens, H.W.J.M. *Computational Experiments with an Asynchronous Parallel Branch and Bound Algorithm.* Technical report, Department of Computer Science, Erasmus University Rotterdam, 1989.

[85] Trienekens, H.W.J.M. *Parallel branch and bound and anomalies.* Technical report, Department of Computer Science, Erasmus University, Rotterdam, 1989.

[86] Trienekens, H.W.J.M. *Parallel Branch and Bound Algorithms.* Technical report, Department of Computer Science, Erasmus Universiteit Rotterdam, 1990.

[87] Trienekens, H.W.J.M., de Bruin, A. *Towards a taxonomy of parallel branch and bound algorithms.* Technical report, Department of Computer Science, Erasmus University Rotterdam, 1992.

[88] Vornberger, O. "Load Balancing in a Network of Transputers". In *WDAG: International Workshop on Distributed Algorithms*. LNCS, Springer-Verlag, 1987.

[89] Zhigljavsky, , Anatoly, A. *Theory of global random search*, v. 65. Kluwer Academic Publishers Group, Dordrecht, 1991.