Ministério da
Ciência, Tecnologia e Inovação

sid.inpe.br/mtc-m19/2012/02.22.17.13-TDI

META-HEURÍSTICAS PARALELAS NA SOLUÇÃO DE PROBLEMAS INVERSOS

Eduardo Fávero Pacheco da Luz

Tese de Doutorado do Curso de Pós-Graduação em Computação Aplicada, orientada pelos Drs. Haroldo Fraga de Campos Velho, e José Carlos Becceneri, aprovada em 08 de março de 2012.

URL do documento original:
<http://urlib.net/8JMKD3MGP7W/3BDGLGH>

INPE
São José dos Campos
2012

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/6921
Fax: (012) 3208-6919
E-mail: pubtc@sid.inpe.br

Ministério da
Ciência, Tecnologia e Inovação

sid.inpe.br/mtc-m19/2012/02.22.17.13-TDI

META-HEURÍSTICAS PARALELAS NA SOLUÇÃO DE PROBLEMAS INVERSOS

Eduardo Fávero Pacheco da Luz

Tese de Doutorado do Curso de Pós-Graduação em Computação Aplicada, orientada pelos Drs. Haroldo Fraga de Campos Velho, e José Carlos Becceneri, aprovada em 08 de março de 2012.

URL do documento original:
<http://urlib.net/8JMKD3MGP7W/3BDGLGH>

INPE
São José dos Campos
2012

Dados Internacionais de Catalogação na Publicação (CIP)

Luz, Eduardo Fávero Pacheco da.
L979m Meta-heurísticas paralelas na solução de problemas inversos / Eduardo Fávero Pacheco da Luz. – São José dos Campos : INPE, 2012.
xxiv + 131 p. ; (sid.inpe.br/mtc-m19/2012/02.22.17.13-TDI)

Tese (Doutorado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2012.
Orientadores : Drs. Haroldo Fraga de Campos Velho, e José Carlos Becceneri.

1. meta-heurística. 2. problemas inversos. 3. otimização. 4. paralelas. I.Título.

CDU 004.023

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de Doutor(a) em

Computação Aplicada

Dr. Nandamudi Lankalapalli Vijaykumar

Presidente / INPE / SJCampos - SP

Dr. José Carlos Becceneri

Orientador(a) / INPE / SJCampos - SP

Dr. Haroldo Fraga de Campos Velho

Orientador(a) / INPE / São José dos Campos - SP

Dr. Stephan Stephany

Membro da Banca / INPE / SJCampos - SP

Dr. Wagner Figueiredo Sacco

Convidado(a) / UFOPA / Santarém - PA

Dr. Luiz Leduíno de Salles Neto

Convidado(a) / UNIFESP / São Paulo - SP

Este trabalho foi aprovado por:

( ) maioria simples

(X) unanimidade

Aluno (a): Eduardo Fávero Pacheco da Luz

São José dos Campos, 08 de março de 2012

"The purpose of computing is insight, not numbers".

R. W. Hamming
em *"Numerical Methods for Scientists and Egineers"*, 1973

Á minha familia, amigos, e colegas...

AGRADECIMENTOS

O desenvolvimento desta tese não seria possível sem o constante suporte e apoio de diversas pessoas. Dentre estas pessoas agradeço,

Meus orientadores, Dr. José Carlos Becceneri e Dr. Haroldo Fraga de Campos, por terem me acolhido e guiado com ideias inspiradas na natureza desde meu primeiro dia no INPE, assim como por terem me apresentado ao fascinante mundo dos problemas inversos e pelas discussões sobre computação de alto desempenho.

Também agradeço a todos os professores do programa de pós-graduação em Computação Aplicada do INPE, que desde o mestrado e durante o doutorado vêm me treinando a ser um cientista.

Agradeço a todos os amigos fiz durante este caminho, e que com conversas e discussões muito me ajudaram a desenvolver a capacidade de fazer pesquisa.

Parte deste trabalho foi financiado por bolsa de doutorado na modalidade de demanda social pela Coordenação de Aperfeiçoamento de Pessoal de Ensino Superior (CAPES) pela qual eu agradeço imensamente.

Também tenho muito o que agradecer à minha família, Verônica, Orley, Lorena, Inácio e em especial Cynthia, pela compreensão da ausência durante os últimos anos.

A todos eu agradeço.

RESUMO

Métodos de otimização inspirados na natureza têm sido constantemente desenvolvidos e aperfeiçoados nas últimas décadas. Da mesma forma, o avanço no desenvolvimento de sistemas computacionais de alto desempenho tem se estendido a ponto de disponibilizar processadores com múltiplos núcleos em computadores pessoais. Este trabalho apresenta os resultados do desenvolvimento do *Multiple Particle Collision Algorithm* (MPCA), algoritmo de otimização voltado a ambientes de computação massivamente paralelo e do *parallel Firefly Algorithm with Predation* (pFAP), para ambientes de múltiplos núcleos com memória compartilhada. Ambos os algoritmos foram validados com aplicações em funções de teste de até 100 dimensões e seus desempenhos foram analisados, tendo inclusive apresentado *speedup* super-linear em casos específicos. Estes novos algoritmos também foram utilizados na solução de problemas inversos de estimação de fonte/sumidouro de um gás, de estimação de condição inicial na equação do calor (para até 50 dimensões) e no novo problema de estimação de pesos de hipóteses de fechamento na construção de mapas de precipitação. Os resultados apresentados demonstram a viabilidade de utilização destes métodos assim como capacitam uma melhor utilização de recursos computacionais disponíveis. Além destes métodos, discussões preliminares sobre uma técnica de seleção automática de parâmetros também são apresentadas, assim como uma visão alternativa sobre a interação entre os componentes envolvidos na solução de problemas inversos.

PARALLEL META-HEURISTICS ON THE SOLUTION OF INVERSE PROBLEMS

ABSTRACT

Optimization methods inspired by nature have been constantly developed and enhanced over the last decades. In the same way, advances in the development of high performance computational systems have been extended to the point of making available multi-core processors in personal computers. This work presents the results based on the development of the Multiple Particle Collision Algorithm (MPCA), an optimization algorithm to be used in massively parallel environments, and the parallel Firefly Algorithm with Predation (pFAP), developed for shared memory, multi-core environments. Both algorithms were validated with applications to benchmark test functions up to 100 dimensions and the performances were analysed, presenting a super-linear speedup in a particular case. These new algorithms were also used in the solution of inverse problems related to the estimation of source/sink of a certain gas, the estimation of initial condition in the heat equation (up to 50 dimensions) and in the new problem of weight estimation for closing hypotheses in the build of precipitation maps. The results demonstrate the viability of using these methods just as capacitates a better use of available computational resources. Aside of these methods, preliminary discussions regarding an automatic selection of parameters technique are also presented, just as an alternative view regarding the interaction between the components involved in the solution of inverse problems.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE ABREVIATURAS E SIGLAS

PD	–	Problema Direto
PI	–	Problema Inverso
VFR	–	Very Fast simulated Re-annealing
GA	–	Genetic Algorithm
PCA	–	Particle Collision Algorithm
MPCA	–	Multiple Particle Collision Algorithm
FA	–	Firefly Algorithm
pFAP	–	parallel Firefly Algorithm with Predation
FDP	–	Função Densidade de Probabilidade
CPU	–	Central Processing Unit
RNG	–	Random Number Generator
SISD	–	Single Instruction Single Data
SIMD	–	Single Instruction Multiple Data
MISD	–	Multiple Instruction Single Data
MIMD	–	Multiple Instruction Multiple Data
TRMM	–	Tropical Rainfall Measuring Mission
BRAMS	–	Brazilian developments on the Regional Atmospheric Modelling System
GD	–	Grell & Dévényi
ZCAS	–	Zona de Convergência do Atlântico Sul

SUMÁRIO

Pág.

1 INTRODUÇÃO

"Alea iacta est"

(Gaius Iulius Caesar)

Diversas áreas das ciências têm enfrentado uma revolução recente, seja em suas ferramentas de análise, seja na maneira com que dados são coletados ou até mesmo como estes são analisados e divulgados.

As áreas que têm a computação como fim (Ciência da Computação) ou como meio (a Computação Aplicada) são os mais claros exemplos desta revolução. Estes exemplos contêm itens de hardware e software que sofrem atualização quase que diária e fomentam esta revolução.

Os usuários da computação como meio, aqueles que neste trabalho serão denominados "Cientistas Programadores", devem estar atentos às rápidas mudanças que podem afetar as suas áreas de interesse. James Nicholas "Jim" Gray, um cientista da computação, ganhador do Prêmio Turing, apresentava em suas palestras o quanto esta revolução tem atuado nas ciências (HEY et al., 2009). Ele pregava a mudança do paradigma das ciências ao longo do tempo, que pode ser sumarizado como:

- Ciência empírica: que era regra há milhares de anos e visava descrever os fenômenos físicos;
- Ciência teórica: que era regra há algumas centenas de anos e fortaleceu o uso de modelos e/ou generalizações;
- Ciência computacional: que é a regra há algumas dezenas e anos e consegue simular fenômenos complexos dado o crescente poder computacional;
- Ciência da exploração dos dados (*eScience*): a regra atual, que busca unificar a teoria, a experimentação e a simulação.

Este fenômeno atual, a *eScience*[1], coloca sobre um mesmo leque os passos de: obtenção dos dados (seja experimentalmente ou por simulação); processamento destes

[1] Ainda segundo Jim Gray, a *eScience* ocorre quando a "Tecnologia da Informação (TI) encontra os cientistas" (HEY et al., 2009).

dados por software; o armazenamento desta informação/conhecimento em computadores; e a análise destes dados por ferramentas estatísticas e de mineração de dados.

Dada a grande importância da *eScience*, pesquisadores como Jim Gray e os autores do livro "*The Fourth Paradigm*", classificam esta ciência da exploração dos dados, como o próprio nome do livro diz, como o quarto paradigma, o paradigma atual, que requer o desenvolvimento de novas técnicas e até mesmo novo hardware para tratar a grande quantidade de informação coletada.

Esta tese apresenta novas ferramentas para ajudar esta tarefa de trabalhar com uma grande quantidade de dados experimentais e/ou gerados por simulações de computadores.

As ferramentas apresentadas neste trabalho são algoritmos heurísticos, baseados no comportamento de animais e de iterações físicas básicas. São classificadas como meta-heurísticas e seu uso principal é na otimização de determinados problemas matemáticos. Estas meta-heurísticas visam localizar, ou estimar, soluções próximas do resultado ótimo, ou até mesmo localizar a solução ótima, para os problemas que estão sendo resolvidos. Elas guiam este processo de busca, tentando otimizar o tempo gasto nesta busca, reduzindo o custo computacional e mantendo o seu foco em uma solução "aceitável" para o Cientista Programador.

Existem diversas meta-heurísticas disponíveis. Algumas são técnicas clássicas, como os Algoritmos Genéticos e o Recozimento Simulado, outras são técnicas que já se estabeleceram com sucesso entre os Cientistas Programadores, como a Otimização por Colônia de Formigas e a Otimização por Enxame de Partículas, outras técnicas são recentes e oferecem ótimas oportunidades para o desenvolvimento de variantes e melhorias, como a Otimização por Colisão de Partículas e a Otimização por Vaga-Lumes, usadas como base para o desenvolvimento deste trabalho. A existência desta ampla variedade de métodos de otimização, a grande maioria estocástica, atende à ideia proposta por Wolpert e Macready (1997) no artigo intitulado *No free lunch theorem for optimization*, que baseado em uma teoria equivalente da economia, estabelece, a grosso modo, que não existe um método de otimização universal. Desta forma, os mais diversos algoritmos de otimização apresentam "melhores" soluções para um determinado subconjunto de problemas enquanto que em outro subconjunto seu desempenho não é aceitável. Desta maneira, a média de resultados de

cada método é sempre intermediária (HO; PEPYNE, 2002a; HO; PEPYNE, 2002b).

O constante desenvolvimento de novos métodos de otimização se justifica pelo exposto acima, pois sempre haverá uma classe de problemas que ainda não é bem resolvida e estes novos métodos possuem, teoricamente, a capacidade de resolver bem esta classe (mesmo que isto implique em resolver mal outras classes de problemas).

Com base nesta justificativa, este trabalho apresenta duas novas meta-heurísticas, a primeira derivada do Algoritmo de Colisão de Partículas, um método estocástico com "leve inspiração" na física de colisão de partículas. Este algoritmo, agora aperfeiçoado com a adição de uma estrutura de múltiplas soluções concorrentes (agora se caracterizando como um algoritmo populacional) tem seu funcionamento expandido para ambientes computacionais de alto desempenho, apresentando melhores resultados em comparação com o algoritmo original. Sua força reside principalmente na nova capacidade de exploração de várias soluções candidatas ao mesmo tempo. A segunda contribuição é derivada do método de otimização baseado no comportamento de vaga-lumes. Esta nova variante incorpora o princípio de seleção natural, proposto pelo naturalista inglês Charles Darwin, que prega a sobrevivência do indivíduo mais apto, garantindo assim a propagação de seus genes para a próxima geração. Os resultados obtidos por esta técnica mostram a eficácia deste mecanismo no escape de regiões de ótimo local, e quando melhorado com implementação paralela, via uso de OpenMP, o tempo de execução é extremamente reduzido.

O texto aqui apresentado está organizado da seguinte forma: o Capítulo 2 descreve brevemente a teoria da otimização, assim como também conceitua e descreve generalidades sobre meta-heurísticas. Conceitos sobre computação de alto desempenho, e suas principais ferramentas de análise de desempenho, também são descritos neste capítulo. Na sequência, são apresentados os algoritmos que deram base a este trabalho. Em cada seção as variantes aqui desenvolvidas são apresentadas e uma breve análise de seu desempenho é exposta. O capítulo é finalizado com uma descrição rápida do trabalho orientado sobre seleção automática de parâmetros.

O Capítulo 3 descreve a teoria de problemas diretos, inversos e os principais operadores de regularização assim como formas de seleção do parâmetro de regularização. Na seção reservada à representação de problemas inversos é apresentada uma nova visão complementar à comumente usada entre as relações dos atores na solução de

problemas inversos. O Capítulo 4 detalha os problemas inversos que serão usados como exemplos de aplicações às meta-heurísticas desenvolvidas neste trabalho. Suas formulações direta e inversa são apresentadas. O Capítulo 5 apresenta resultados de comparação entre os novos algoritmos e suas versões originais inicialmente aplicados a funções de testes com o objetivo de demonstrar vantagens das novas variantes. Posteriormente, as novas meta-heurísticas são aplicadas aos problemas inversos descritos no Capítulo 4. O Capítulo 6 tece as conclusões, considerações finais e apresenta possíveis trabalhos futuros derivados deste trabalho.

2 OTIMIZAÇÃO COM META-HEURÍSTICAS

"You can't suppress 65 million years of gut instinct."

(Jurassic Park, o filme)

Este capítulo apresenta os conceitos-base sobre otimização, meta-heurísticas e também descreve os algoritmos desenvolvidos como fruto deste trabalho. As versões canônicas que foram base para o desenvolvimento dos novos algoritmos também são apresentadas assim como uma análise de desempenho da nova variante frente a versão canônica.

2.1 Otimização

A teoria da otimização é o ramo da matemática que engloba o estudo quantitativo do ótimo e os métodos usados para encontrar o ótimo. Esta teoria posta em prática é definida pela coleção de técnicas, métodos, procedimentos e algoritmos que podem ser usados para a localização do ótimo de uma função (ANTONIOU; LU, 2007).

Problemas de otimização têm por objetivo encontrar a melhor combinação dentre um conjunto de variáveis para maximizar ou minimizar uma função, definida como sendo uma função objetivo ou função custo. Considerando o espaço de busca a ser explorado, os problemas de otimização podem ser classificados como (BECCENERI, 2008):

- Problemas de otimização contínua: cujas variáveis assumem valores reais ou contínuas, e.g., $x \in \mathbb{R}$;
- Problemas de otimização combinatória ou discreta: cujas variáveis assumem valores discretos ou inteiros, e.g., $x \in \mathbb{Z}$;
- Problemas de otimização mista: com variáveis inteiras e contínuas ao mesmo tempo, e.g., $x \in (\mathbb{R}|\mathbb{Z})$.

Otimizar é aproveitar da melhor forma possível a capacidade de alguém ou de alguma coisa. Matematicamente, otimizar um problema, ou conjunto de equações, significa encontrar um valor, ou conjunto de valores ótimos, que garantam um resultado mínimo ou máximo.

Para funções na forma $f(\mu; Y)$, que medem a aptidão de um modelo com μ parâmetros a algum conjunto de dados Y, o objetivo consiste-se em escolher um conjunto de parâmetros ótimos para o qual obtenha uma melhor aptidão para os dados. Em outras palavras, o objetivo é achar o ótimo (máximo ou mínimo) da função $f(\mu; Y)$ com relação a μ.

Considerando a notação simplificada $f(\mu; Y) = f(\mu)$, o máximo de $f(\mu)$ corresponde ao mínimo de $-f(\mu)$ e, portanto, um problema de minimização engloba, de maneira geral, um problema de maximização.

De maneira formal, μ é um ponto ótimo de $f(\mu)$ se existir uma região em torno de μ de raio ε tal que (THACKER; COOTES, 1996):

$$f(\mu + \eta) > f(\mu), \forall |\eta| < \varepsilon \tag{2.1}$$

O ótimo, seja máximo ou mínimo, de uma função pode ser classificado em global, quando representa o maior ou menor valor de toda uma região de interesse, ou local, quando representa o maior ou menor valor de uma dada sub-região do espaço total (Figura 2.1).

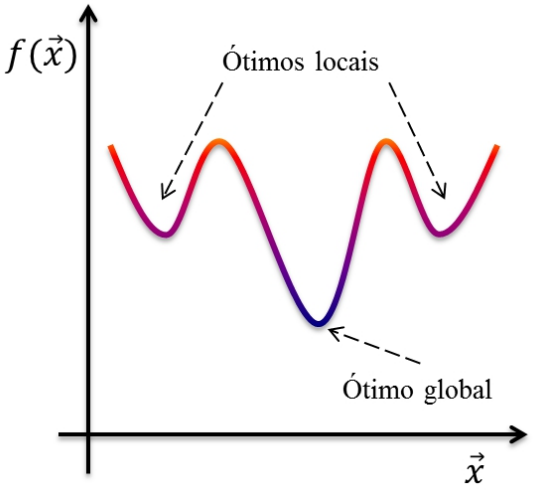


Figura 2.1 - Exemplo de ótimos locais e ótimo global para a minimização de uma função.

O melhor método para a localização do ótimo de uma função depende fortemente da

natureza da função em estudo, sendo que, em linhas gerais, duas classes de algoritmos podem ser usadas: os algoritmos de otimização local, que dado um ponto em um subdomínio da função, eles tentam, e quase sempre conseguem, localizar o ponto mais baixo deste subdomínio; e os algoritmos de otimização global, que tentam otimizar a função encontrando o ponto mais baixo dentre todos os subdomínios existentes no espaço de buscas.

Se iniciarmos a busca do ótimo global com uma boa indicação da localização deste ponto, podemos facilmente nos valer dos algoritmos de otimização local. Caso contrário, os algoritmos de otimização global são usados para explorar a totalidade do espaço de buscas. Logo, os algoritmos de otimização global têm grande uso quando o espaço de buscas possui vários pontos de ótimos locais, tornando mais difícil a localização do ótimo global.

De maneira genérica, um problema de otimização pode ser escrito como:

$$\underset{x \in \Re^n}{\text{otimizar}} f_i(x), \qquad (i = 1, 2, \ldots, M), \tag{2.2}$$

$$\text{sujeito a } \phi_j(x) = 0, \qquad (j = 1, 2, \ldots, J), \tag{2.3}$$

$$\psi_k(x) \leq 0, \qquad (k = 1, 2, \ldots, K), \tag{2.4}$$

onde $f_i(x)$, $\phi_j(x)$ e $\psi_k(x)$ são funções do vetor de decisão

$$x = (x_1, x_2, \ldots, x_n)^T, \tag{2.5}$$

onde os componentes x_i de x são designados de variáveis de decisão e podem ser números reais, inteiros ou uma mistura destes (YANG, 2010a).

Os problemas de otimização também podem ser classificados como (BALDICK, 2006):

- Problemas de otimização irrestrita;
- Problemas de otimização restrita por igualdade;
- Problemas de otimização restrita por desigualdade.

Para resolver estes problemas, podemos nos valer de procedimentos algorítmicos diretos ou iterativos. Algoritmos diretos obtém a solução exata para o problema que está resolvendo após um número finito de operações. As desvantagens no uso de algoritmos diretos residem na necessidade da existência explícita de uma formulação matemática diferenciável (o que nem sempre pode acontecer) e, em alguns casos, na inviabilidade da obtenção de uma solução ótima, ou próxima do ótimo, em um tempo computacional aceitável. Algoritmo iterativos, por sua vez, geram uma sequência aproximadas de soluções, iterando, ou convergindo, em direção a uma solução do problema, em grande parte se valendo só do valor da função objetivo, i.e., métodos de ordem-zero (BALDICK, 2006; DIWEKAR, 2008; KAO, 2008).

Os passos básicos para a operação de um algoritmo de otimização são comuns às mais diversas técnicas e envolvem: *a*) a inicialização do otimizador com valores iniciais (seja aleatoriamente ou por uma outra técnica qualquer); *b*) a obtenção das variáveis de decisão por parte do otimizador; *c*) a avaliação das variáveis de decisão por parte do problema/modelo em estudo; *d*) o retorno do valor da função objetivo e informações sobre violação de restrições para o otimizador; *e*) a convergência do otimizador para uma solução (espera-se que ótima ou próxima da ótima). Podemos ver na Figura 2.2 a representação esquemática para estes passos.

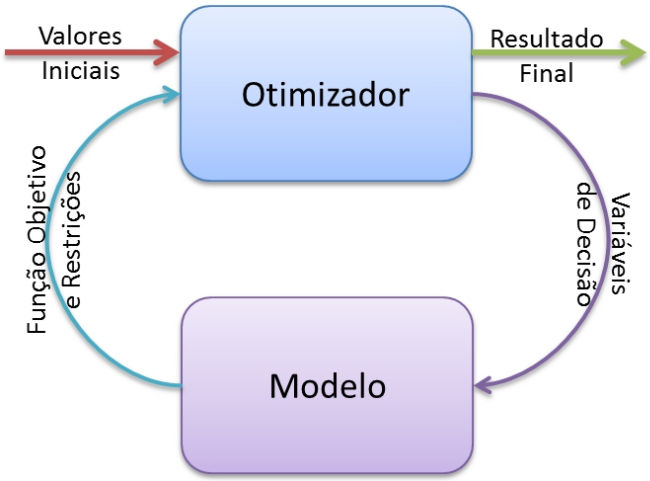


Figura 2.2 - Representação esquemática do *framework* de um algoritmo de otimização.
Fonte: Adaptado de Diwekar (2008)

2.2 Conceituação e descrição geral de meta-heurísticas

A grande maioria dos problemas que envolvem a tomada de decisões nas mais diversas áreas do conhecimento, e.g., ciências, engenharia, economia, indústria em geral (para este último caso podemos citar mais exemplos de: localização de facilidades, roteamento, alocação de recursos, dentre outros), podem ser escritos como um problema de otimização.

Estes problemas podem então ser resolvidos por uma ampla variedade de métodos e algoritmos de otimização que vêm sendo constantemente aperfeiçoados. Como exemplos podemos citar: o algoritmo de *Branch-and-bound*; o método Simplex de Nelder e Mead; o algoritmo de Colônia de Formigas; o algoritmo de Enxame de Partículas, dentre outros (FLOUDAS; PARDALOS, 2009).

Esses algoritmos podem ser facilmente classificados como:

- Algoritmos exatos;
- Algoritmos aproximados.

Os algoritmos exatos, ou métodos exatos[1] são aqueles algoritmos sequenciais que não se baseiam em processos estocásticos em nenhuma parte do seu processo de busca. Dos exemplos citados acima, o algoritmo de *Branch-and-bound* e o método Simplex são classificados como métodos exatos.

Algoritmos aproximados, ou estocásticos, são aqueles algoritmos que sustentam parte de seu funcionamento em uma variável aleatória. Esta variável aleatória é fornecida por programas Geradores de Números Aleatórios (*Random Number Generators*, RNGs)[2]. Exemplos de algoritmos estocásticos são o Algoritmo Genético, o Algoritmo de Colônia de Formigas e o Algoritmo de Enxame de Partículas. Os algoritmos estocásticos também podem ser subdivididos em algoritmos de aproximação e algoritmos heurísticos, que por sua vez podem ser novamente classificados como heurísticas específicas para o problema e meta-heurísticas.

A Figura 2.3 apresenta um esquema simplificado de classificação de métodos de otimização clássicos. Neste esquema, as meta-heurísticas podem ser novamente clas-

[1] A comunidade de inteligência artificial classifica estes como "algoritmos completos".

[2] Uma vez que os algoritmos usados para a geração destes números segue uma sequência lógica, o mais correto seria caracterizá-los como números pseudo-aleatórios.

sificadas em métodos baseados em uma solução simples ou métodos baseados em uma população de soluções (TALBI, 2009).

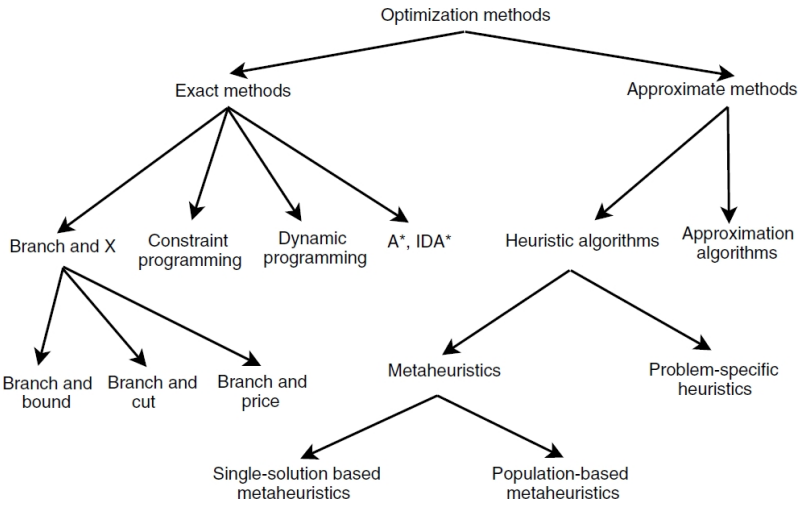


Figura 2.3 - Algoritmos clássicos de otimização.
Fonte: Talbi (2009).

Dado o tamanho e mesmo a complexidade de alguns dos problemas a serem resolvidos, a solução por algoritmos exatos se torna inviável. Neste momento os algoritmos heurísticos destacam-se por sua principal vantagem, que apesar de não garantirem a otimalidade da solução, garantem uma solução aceitável em um tempo aceitável.

Uma heurística é uma técnica, que consiste de uma regra ou conjunto de regras, que busca, e esperançosamente acha, boas soluções a um custo computacional razoável. Heurísticas são aproximadas no sentido de que fornecem uma boa solução com relativamente pouco esforço, mas que não garante a otimalidade da solução (FLOUDAS; PARDALOS, 2009).

O termo meta-heurística foi introduzido por Glover (1986) ao conceitualizar a existência de uma heurística[3] superimposta a outra heurística. O prefixo "meta" expressa o conceito de nível superior ou maior generalidade.

Assim, a meta-heurística é uma estratégia de nível superior que guia uma heurística subordinada que resolve um dado problema. Pode-se aqui estabelecer uma distinção

[3]Heurística: do verbo grego *heuriskein*, que significa "encontrar, descobrir".

entre um "processo-guia" e um "processo-aplicação". O processo-guia define possíveis movimentos e envia esta informação para o processo-aplicação que então executa o movimento escolhido (FLOUDAS; PARDALOS, 2009).

Outra boa definição para meta-heurísticas é apresentada por Floudas e Pardalos (2009) e que será adotada como a padrão para esta tese. Esta definição diz que a meta-heurística é um processo mestre iterativo que guia e modifica as operações de heurísticas subordinadas para eficientemente produzir soluções de alta qualidade. Ela pode manipular uma solução completa (ou incompleta) ou um conjunto de soluções a cada iteração. A heurística subordinada pode ser um procedimento de alto (ou baixo) nível ou uma simples busca local ou um método de construção aleatório simples.

De acordo com Blum e Roli (2003), algumas propriedades que caracterizam meta-heurísticas são:

- Meta-heurísticas são estratégias que "guiam" o processo de busca;
- O objetivo é explorar eficientemente o espaço de buscas para encontrar soluções (aproximadamente) ótimas;
- Técnicas que constituem algoritmos de meta-heurísticas variam de simples procedimentos de busca local à complexos processos de aprendizado;
- Algoritmos de meta-heurísticas são aproximados e geralmente não-determinísticos;
- Eles podem incorporar mecanismos para evitar ficar presos em áreas confinadas do espaço de busca;
- Os conceitos básicos de meta-heurísticas permitem um nível de descrição abstrata;
- Meta-heurísticas não são específicas para certos problemas;
- Meta-heurísticas podem se valer de conhecimento específico do domínio na forma de heurísticas que são controladas por uma estratégia de nível superior;
- As mais avançadas meta-heurísticas da atualidade usam experiência de busca (encorporado em alguma forma de memória) para guiar a busca.

Para as meta-heurísticas, os conceitos de diversificação (*diversification*) e intensificação (*intensification*) são de grande importância. Na língua inglesa estes termos podem ser relacionados à *exploration* e *exploitation*, respectivamente, mas que infelizmente na língua portuguesa ambos possuem a mesma tradução: "exploração". Os conceitos da língua inglesa representam melhor o comportamento das meta-heurísticas para a diversificação e intensificação. De maneira ilustrativa, podemos relacionar o conceito de *exploration* com a "exploração do interior do Brasil pelos bandeirantes", i.e., a exploração em modo global do espaço de buscas, e o conceito de *exploitation* com a "exploração de um poço de petróleo pela Petrobrás", i.e., a exploração em modo local do espaço de buscas.

Meta-heurísticas ainda podem ser classificadas de acordo com algumas características básicas, tais como (BLUM; ROLI, 2003):

- Inspirada na natureza ou não inspirada na natureza;
- Baseada em população ou busca de ponto único;
- Função objetivo dinâmica ou estática;
- Estrutura de vizinhança única ou variável;
- Métodos com uso de memória ou sem memória.

Uma outra característica que permite a classificação das meta-heurísticas é com relação ao seu comportamento de busca. Esta distinção entre a capacidade do algoritmo de "achar uma solução viável" e "melhorá-la" separa as meta-heurísticas entre métodos de busca local e meta-heurísticas de aspecto geral. A grande maioria dos métodos já possui um mecanismo implícito, e em algumas vezes até mesmo explícito, de ajuste automático entre o comportamento de busca global ou local. Esta, inclusive, é uma das grandes vantagens dos principais métodos apresentados nesta tese.

Neste ponto, cabe uma nota sobre a nomenclatura de meta-heurísticas. Apesar do nome ser bem difundido no meio acadêmico e industrial, alguns autores não consideram correto o seu uso. Um destes autores é Luke (2010). Em seu Capítulo 0, de introdução, ele já expressa sua opinião de que estes métodos deveriam ser denominados simplesmente de "otimização estocástica", mesmo que outras técnicas

essencialmente diferentes também sejam classificadas nesta categoria. De acordo com seu texto, quando se ouve falar de "meta-discussão", fica explícita a *discussão sobre discussão*. Da mesma maneira quando ele ouve "meta-heurística" ele acha estranho (*weird*) uma *heurística sobre (ou para) heurística*. A questão é que, dadas as definições acima, sabemos que uma meta-heurística é uma heurística que guia outra heurística na busca do ótimo (ou próximo do ótimo) do problema. Logo, esta definição não teria problema, mas fica aqui registrada a opinião de outros autores.

2.3 Computação de Alto Desempenho

Ambientes de alto desempenho são possibilitados através do uso de computação de alto desempenho (*High Performance Computing*, HPC), termo que reflete o uso de supercomputadores (máquinas massivamente paralelas ou vetoriais) e/ou clusters de computadores.

O processo de computação, que pode ser definido como a execução de uma sequência de instruções em um conjunto de dados, pode ser agilizado quando o uso de um ambiente de alto desempenho é posto em prática. Neste ponto, a computação em um ambiente paralelo consiste-se de uma ou mais tarefas que são executadas de maneira concorrente, sendo que o número destas tarefas pode variar durante a execução do programa.

Em termos gerais, os sistemas computacionais que possibilitam ao usuário um ambiente de computação que proporcione gigaflops, teraflops e até mesmo petaflops, são designados "computadores de alto desempenho".

A taxonomia apresentada por Flynn (1966) visa classificar a arquitetura dos computadores capacitados a operar em ambientes de alto desempenho. As possíveis classes são:

- *Single Instruction Stream – Single Data Stream* (SISD);
- *Single Instruction Stream – Multiple Data Stream* (SIMD);
- *Multiple Instruction Stream – Single Data Stream* (MISD);
- *Multiple Instruction Stream – Multiple Data Stream* (MIMD).

A representação demonstrada na Tabela 2.1 deixa claro o uso das classificações

com relação aos fluxos de instruções e dados e apresenta exemplos de arquiteturas relacionadas.

Tabela 2.1 - Taxonomia de Flynn para o paralelismo de instruções e dados.

	Instrução única	**Instruções múltiplas**
Dados únicos	SISD von Neumann	MISD pipeline
Dados múltiplos	SIMD CUDA	MIMD multiprocessadores

O primeiro termo da taxonomia, SISD, tem sua referência nas arquiteturas em que um único processador executa uma única série de instruções aplicadas nos dados armazenados em uma única área de memória. É o esquema correspondente à arquitetura proposta por von Neumann. A Figura 2.4(a) ilustra o esquema da SISD.

O segundo termo da classificação apresentada por Flynn, SIMD, aqui representado pela Figura 2.4(b), introduz a técnica empregada quando o paralelismo a nível de dados é a intenção do programador, pois o objetivo é a utilização de um possível processador vetorial. O exemplo de arquitetura SIMD é o de processadores gráficos, como CUDA.

A terceira classificação, MISD, é apropriada para uma arquitetura de computação paralela onde várias unidades funcionais executam diferentes operações em um mesmo conjunto de dados, com sua esquematização dada pela Figura 2.4(c). Os pipelines são exemplos de arquitetura MISD.

Finalmente, a quarta classificação, MIMD, é usada quando queremos obter um alto grau de paralelização, pois neste caso os processadores funcionam de maneira assíncrona e independente, com trechos de código (instruções de programação) diferentes em diferentes pedaços da totalidade dos dados, com seu esquema de funcionamento dado pela Figura 2.4(d). O exemplo recorrente é o dos atuais processadores de múltiplos núcleos.

Porém uma segunda classificação, baseada na análise de soluções de hardware e software em uso, pode ser feita. Baseando-se em um esquema mais simples, que leva

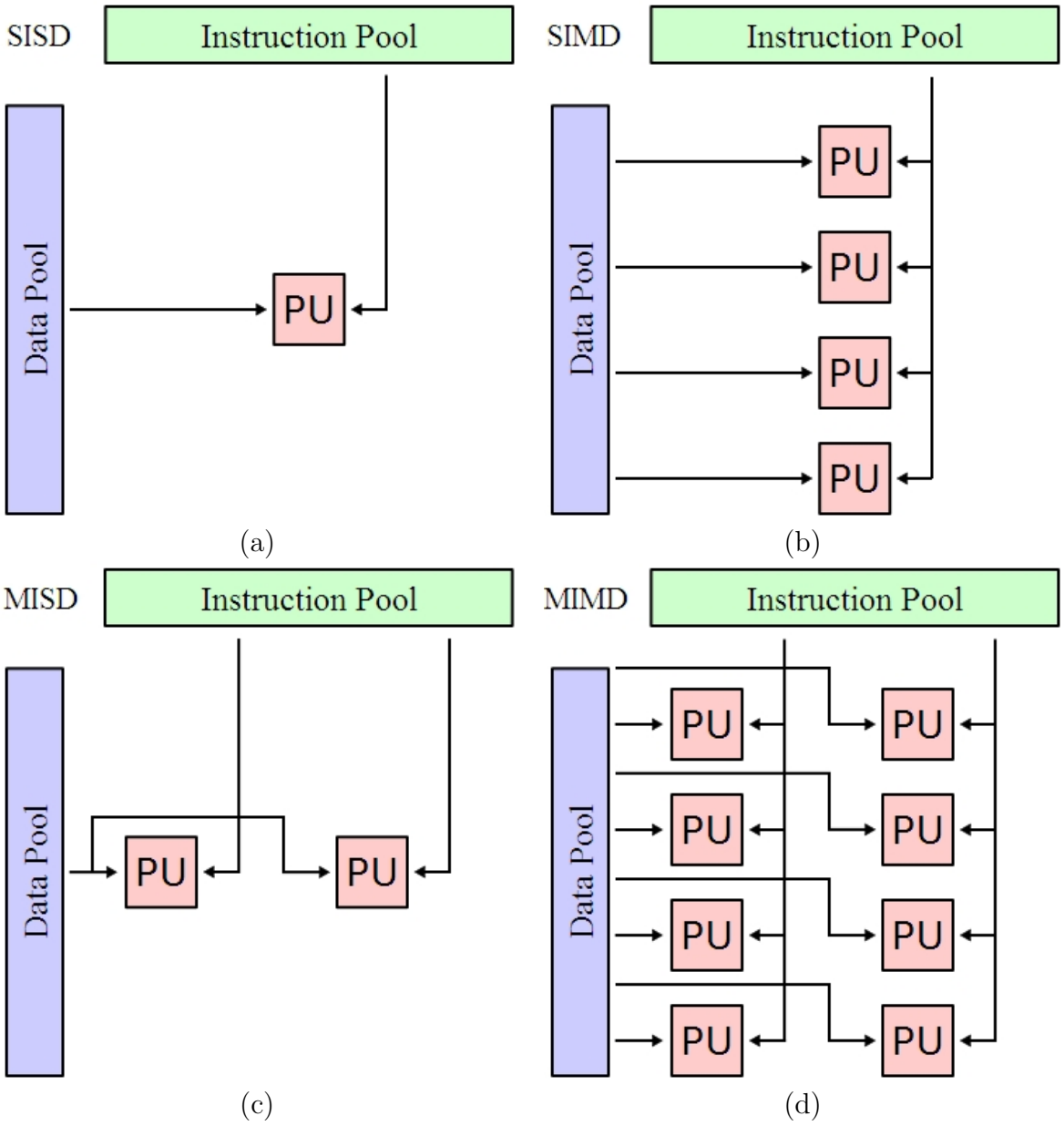


Figura 2.4 - Representações esquemáticas para a taxonomia de Flynn: (a) SISD; (b) SIMD; (c) MISD; (d) MIMD.
Fonte: Wikipedia (2010).

em consideração as principais características de cada arquitetura (SAMBATTI, 2004), temos as seguintes arquiteturas de hardware:

- Arquitetura vetorial: a mesma instrução é executada, sincronizadamente, por todos os processadores, mas em conjuntos distintos de dados;
- Arquitetura multiprocessada com memória compartilhada: máquinas com memória compartilhada, normalmente com um pequeno número de processadores que têm acesso a uma memória global, onde ocorre a comunicação entre os processadores;
- Arquitetura multiprocessada com memória distribuída: interligação de conjuntos de processadores e memórias através de uma rede de interconexão. O acesso aos dados remotos é feito através de uma troca de mensagens e os processadores executam diferentes instruções em momentos distintos. O sincronismo só é obtido com o uso de mecanismos de linguagem de programação;
- Arquitetura de sistema híbrido: existência de vários nós multiprocessados interconectados por redes semelhantes a de uma arquitetura de sistema distribuído.

Para as soluções de software, as possibilidades incluem implementação via (SAMBATTI, 2004):

- *High Performance Fortran* (HPF), uma linguagem paralela automática, extensão do Fortran 90 com a incorporação de diretivas para o paralelismo de dados;
- OpenMP, um conjunto de diretivas de programação e bibliotecas para programação de aplicações paralelas, baseado em um modelo para memória compartilhada, sendo o paralelismo explícito;
- *Parallel Virtual Machine* (PVM), um pacote de bibliotecas integradas e de ferramentas de software que permite que uma coleção heterogênea de computadores seriais, paralelos e vetoriais interligados em rede funcionem como um único recurso computacional;

- *Message Passing Interface* (MPI), uma biblioteca de troca de mensagens utilizada em ambientes de memória distribuída, sendo obtida por chamadas explícitas às rotinas de comunicação, onde todo o paralelismo é explícito.

Atualmente, técnicas de computação em grade (*grid computing*), onde os recursos computacionais estão geograficamente distribuídos, se valem de uma solução via sistema de computação, que pode ser definido, neste caso, como uma plataforma de software que gerencia o tráfego de dados de forma segura e administra a execução de tarefas atribuídas.

Os sistemas de arquitetura paralela são enquadrados na categoria MIMD, de acordo com a taxonomia de Flynn, pelo fato de executarem diferentes instruções de código em diversos conjuntos de dados disponíveis. Estes sistemas são ainda classificados como multi-processadores, que são máquinas com vários processadores e memória compartilhada (um ambiente definido como fortemente acoplado), e multi-computadores, máquinas com memória distribuída (um ambiente definido como fracamente acoplado).

As máquinas de memória compartilhada são novamente divididas entre máquinas de acesso uniforme à memória (*Uniform Memory Access*, UMA) e máquinas de acesso não uniforme à memória (*non-Uniform Memory Access*, nUMA). Nas máquinas de acesso uniforme, todos os processadores são ligados à memória por um único barramento e o tempo de acesso de qualquer processador a qualquer parte da memória é o mesmo (este tipo de arquitetura também é definido como *Symetric Multiprocessor System*, SMP). Nas máquinas de acesso não uniforme, um dado processador pode levar mais tempo do que outro para acessar uma porção de memória que não esteja em seu barramento.

A Fig. 2.5 apresenta a classificação de sistemas paralelos do tipo MIMD.

Para a implementação de programas paralelos em máquinas SMP, podemos usar um paradigma de *threading* explícito ou um paradigma baseado em diretivas de compilação. Para a implementação de programas paralelos em memória compartilhada com *threading* explícito, a ferramenta POSIX *Threads* é reconhecida como o padrão mais adotado. Já o paradigma baseado em diretivas de compilação tem como maior exemplo o padrão OpenMP, que é baseado em *threads*.

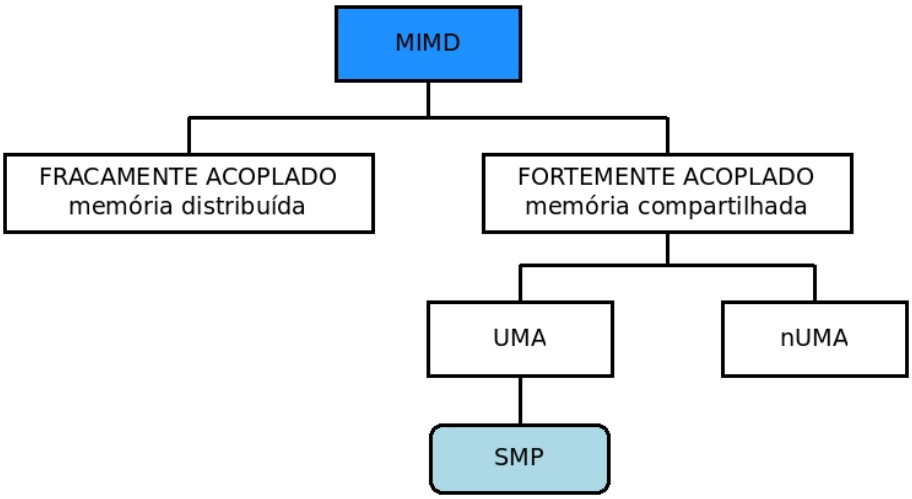


Figura 2.5 - Classificação de arquiteturas paralelas do tipo MIMD.

2.3.1 Métricas para análise de desempenho

A análise do desempenho obtido por algoritmos ou programas desenvolvidos para operar em sistemas computacionais de alto desempenho é fator importante e necessário para justificar e entender melhoras no novo sistema com relação ao sistema inicial, presumivelmente implementado em um ambiente de execução serial.

Duas análises relacionadas ao novo desempenho podem ser feitas de imediato. A primeira verifica a taxa de *speedup* do algoritmo paralelo em comparação com sua versão serial. A análise de *speedup* é definida como a razão entre o tempo de execução do algoritmo serial (T_s) e o tempo de execução do algoritmo paralelo (T_p) e pode ser matematicamente expresso como:

$$S_p = \frac{T_s}{T_p} \tag{2.6}$$

A segunda análise, relacionada à eficiência do algoritmo paralelo, é definida como a razão entre o *speedup* (S_p) e o número de processadores em uso (p), com o objetivo de verificar o quanto do total de paralelismo disponível foi usado pelo algoritmo. Sua expressão é dada por:

$$E_p = \frac{S_p}{p} \leq 1 \tag{2.7}$$

Para garantir uma relação linear com o número de processadores em uso, o algoritmo proposto deve obter uma eficiência calculada em torno de um.

A grande maioria das análises de eficiência executadas em algoritmos paralelos apresenta valores menores do que um, revelando um desempenho sub-linear (Figura 2.6). Esta discrepância é explicada nos termos da Lei de Amdahl (**??**).

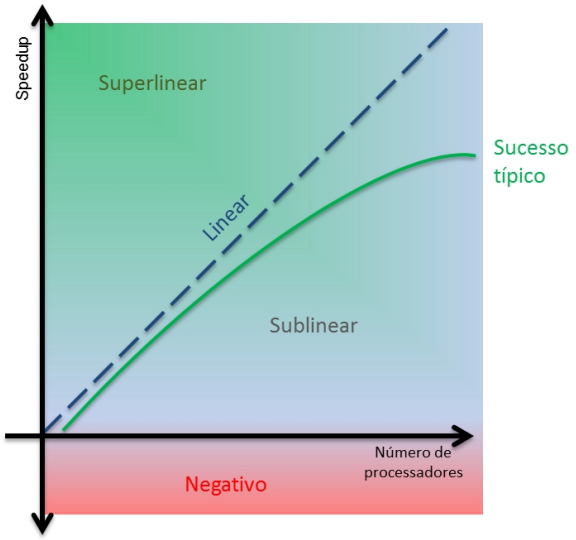


Figura 2.6 - Tipos de *speedup* que podem ser obtidos ao se calcular o ganho advindo da paralelização de um programa.
Fonte: Adaptado de Sutter (2008).

A lei, ou argumento, de Amdahl[4] é comumente usada para expressar a relação entre a parcela de código paralelizável e a sua contra-parte não-paralelizável que leva à estimativa do *speedup* máximo que pode ser obtido com sua paralelização.

A Equação 2.8 apresenta a relação citada anteriormente. O cálculo do *speedup* (S) é influenciado pela razão entre a porção do código passível de paralelização (P) divido entre o número de processadores disponíveis (n) e a porção do código não paralelizável ($1 - P$).

$$S = \frac{1}{(1 - P) + \frac{P}{n}} \tag{2.8}$$

[4]Denominação dada em homenagem ao cientista Gene Amdahl, que inicialmente apresentou a teoria.

Como um exemplo, se a porção de código paralelizável corresponde a 90% do total de código, então tem-se $P = 0,9$ e $(1 - P) = 0,1$ para o restante do código não paralelizável. Desta forma, não importando o número de processadores em uso, este código estará sempre teoricamente limitado a um fator de *speedup* no valor de 10.

Para a lei de Amdahl, quanto maior o valor de P, melhor as condições de paralelização do código. Nestes casos, quando o algoritmo ou problema tem altos valores de P, diz-se que estes são "embaraçosamente paralelos", o que implica que altos valores de *speedup* podem ser obtidos.

Em alguns casos, *speedups* super-lineares, i.e., algoritmos ou programas paralelos com eficiência calculada maior do que um (veja a Equação 2.7 e a área verde na Figura 2.6), podem ocorrer. Para estes casos, o efeito de cache, obtido pela alta taxa de *cache hits* nos diversos níveis de cache atualmente disponíveis nos sistemas computacionais, pode ser usado como uma explicação plausível para este efeito.

2.4 Algoritmo de colisão de partículas

O algoritmo de colisão de partículas (*Particle Collision Algorithm* - PCA) foi apresentado por Sacco e Oliveira (2005), tendo inspiração no algoritmo de recozimento simulado (KIRKPATRICK et al., 1983).

A inspiração para a construção do algoritmo tem suas raízes na física das reações de colisão de partículas em um reator nuclear, com grande ênfase nos comportamentos de espalhamento (*scattering*) e absorção (*absorption*). Seu uso tem sido comprovadamente eficaz em diversos casos de otimização, tanto de problemas de teste (*benchmark*) quanto em problemas de aplicações reais (SACCO et al., 2006; SACCO et al., 2007; SACCO et al., 2008).

O trecho principal do algoritmo referente à técnica de PCA para a maximização de uma função é descrita na Figura 2.7.

A estrutura de funcionamento do PCA leva em consideração a escolha de uma solução inicial (`Old_config`), que sofre uma modificação estocástica através da função `Perturbation()`, que é descrita na Figura 2.8, e então tem-se a construção de uma nova solução (`New_config`). As qualidades das duas soluções são calculadas através da função `Fitness()` e imediatamente comparadas, desta maneira a nova solução pode ser aceita, ou não, dependo de sua performance frente a solução anterior.

Algoritmo de Colisão de Partículas

```
Gera uma solução inicial Old_Config
Best_Fitness = Fitness(Old_Config)
Para n=0 até # de iterações
   Perturbation()
   Se Fitness(New_Config) > Fitness(Old_Config)
      Se Fitness(New_Config) > Best_Fitness
        Best_Fitness := Fitness(New_Config)
      Fim-Se
      Old_Config = New_Config
      Exploration()
   Senão
      Scattering()
   Fim-Se
Fim-Para
```

Figura 2.7 - Algoritmo de colisão de partículas.

Procedimento para perturbação no PCA

```
Perturbation()
   Para i=0 até (Dimensões-1)
      Superior = Limite_superior[i]
      Inferior = Limite_inferior[i]
      Aleatorio = Random(0,1)
      New_Config[i] = Old_Config[i] + ((Superior - Old_Config[i] *
Aleatorio) - ((Old_Config[i] - Inferior)*(1-Aleatorio))
      Se New_Config[i] > Superior
        New_Config[i] = Limite_superior[i]
      Senão
        Se New_Config[i] < Inferior
          New_Config[i] = Limite_inferior[i]
        Fim-Se
      Fim-Se
   Fim-Para
Retorna
```

Figura 2.8 - Procedimento para perturbação.

No caso de uma melhor solução ser obtida neste ponto, esta solução é aceita e um esquema de exploração local é ativado pela função `Exploration()`, função esta descrita pela Figura 2.9.

Procedimento para exploração no PCA

```
Exploration()
   Para n=0 até # de iterações
      Small_Perturbation()
      Se Fitness(New_Config) > Fitness(Old_Config)
        Old_Config = New_Config
      Fim-Se
   Fim-Para
Retorna
```

Figura 2.9 - Procedimento para exploração local.

Dentro da função de exploração, pequenas perturbações estocásticas são aplicadas à solução através da função `Small_Perturbation()` com o objetivo de executar uma pequena exploração local, tal como o demonstrado na Figura 2.10.

Procedimento para pequena perturbação no PCA

```
Small_Perturbation()
   Para i=0 até (Dimensoes-1)
      Superior = Random(1.0,1.2) * Old_Config[i]
      Se Superior > Limite_superior[i]
        Superior = Limite_superior[i]
      Fim-Se
      Inferior = Random(0.8,1.0) * Old_Config[i]
      Se Inferior < Limite_inferior[i]
        Inferior = Limite_inferior[i]
      Fim-Se
      Aleatorio = Random(0,1)
      New_Config[i] = Old_Config[i] + ((Superior -
Old_Config[i])*Aleatorio)-((Old_Config[i] - Inferior)*(1-Aleatorio))
   Fim-Para
Retorna
```

Figura 2.10 - Procedimento para busca local.

Caso esta solução não seja aceita, um esquema do tipo Metropolis (METROPOLIS et al., 1953) é posto em prática através da função `Scattering()`, definida pela Figura 2.11. Este esquema estocástico tenta evitar a prisão do algoritmo em uma região de ótimo local no espaço de buscas.

Procedimento para espalhamento no PCA

```
Scattering()
   p_scattering = 1 - Fitness(New_Config) / Best_Fitness
   Se p_scattering > Random(0,1)
      Old_Config = Solução aleatória
   Senão
      Exploration();
   Fim-Se
Retorna
```

Figura 2.11 - Procedimento para espalhamento.

Para garantir que as perturbações impostas ao algoritmo não levem à construção de uma solução não-viável, a verificação dos limites é feita constantemente dentro das funções `Perturbation()` e `Small_Perturbation()`, que são usadas como ferramentas de busca com comportamento global e local, respectivamente. As descrições das funções estão na Figura 2.8 e Figura 2.10. A Figura 2.12, ilustra o desenvolvimento do PCA na busca de um ponto de máximo de uma função.

Os parâmetros que regulam o funcionamento do algoritmo PCA são:

- N_{I_G}: número de iterações globais do algoritmo, correspondendo ao laço de iteração mais externo do algoritmo;
- N_{I_L}: número de iterações para as perturbações locais, correspondendo ao laço e à quantidade de chamadas à função `Small_Perturbation()`;
- LS_{SP}: limite superior da intensidade da perturbação na busca local, usada pela função `Small_Perturbation()`;
- LI_{SP}: limite inferior da intensidade da perturbação na busca local, usada pela função `Small_Perturbation()`.

Além dos parâmetros citados acima, a função usada para a ativação do espalhamento ($p_{scattering}$), também é suscetível à escolha por parte do usuário.

Da mesma forma, o mecanismo de busca local representado pela chamada à função `Small_Perturbation()` também é passível de ajuste por parte do usuário. Como parte deste projeto de pesquisa, o mecanismo de busca aleatória implementada na versão canônica foi substituído por chamadas a um método de otimização determinístico demonstrando vantagens em algumas instâncias de testes (SOTERRONI et al.,

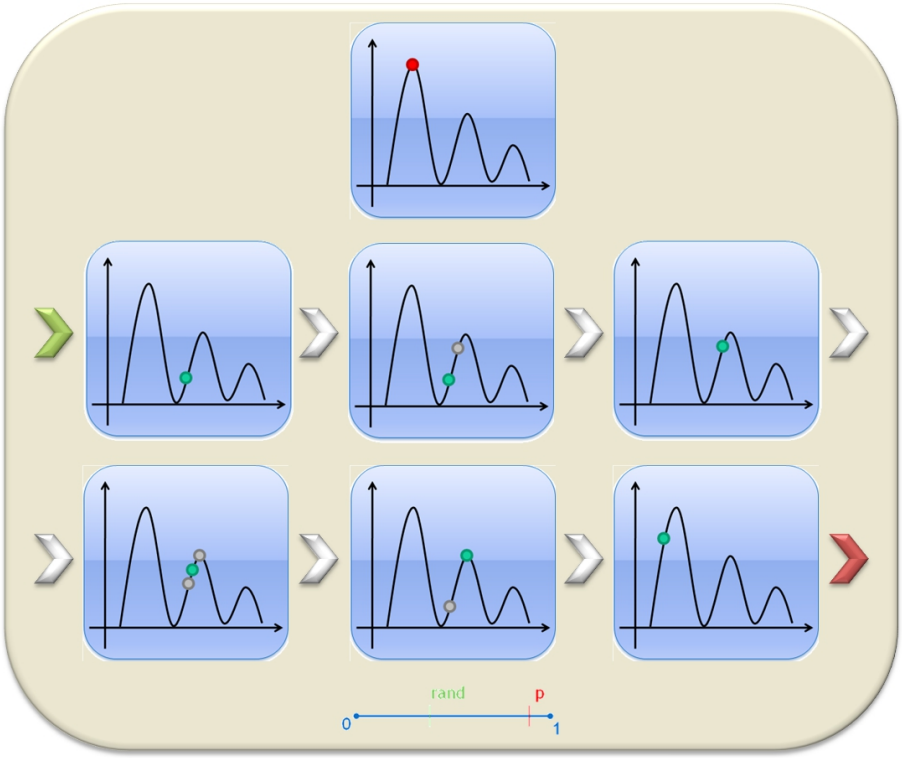


Figura 2.12 - Esta figura ilustra o funcionamento do PCA (os quadros serão referenciados de cima para baixo, da esquerda para a direita): (1) o objetivo é localizar o máximo; (2) uma solução inicial é setada aleatoriamente dentro do espaço de buscas; (3) uma solução candidata é gerada pela função `Perturbation()`; (4) se a solução candidata for melhor do que a solução anterior, ela se torna a solução atual, mimetizando o comportamento de absorção; (5) após a aborção, inicia-se o procedimento de exploração local com a função `Small_Perturbation()`, que pode levar a pequena melhora da solução anterior; (6) o algoritmo itera e a função `Perturbation()` é chamada novamente gerando uma solução pior, que pode ativar o mecanismo de espalhamento; (7) com o mecanismo de espalhamento acionado, o algoritmo escapou de uma região de ótimo local e espera-se que conforme evolua, o ótimo global seja encontrado.

2009). Maiores estudos ainda devem ser conduzidos para garantir a validade desta variante.

2.4.1 Algoritmo de colisão de múltiplas partículas

O algoritmo proposto para o *Multiple Particle Collision Algorithm* (MPCA) é em sua essência, uma variante populacional do PCA canônico, onde introduz-se o conceito e o uso de múltiplas partículas (o equivalente a uma população de soluções candidatas), adicionado de implementação via MPI em um ambiente computacional de alto desempenho, para agilizar o processo e ao mesmo tempo abrir um maior leque de exploração do espaço de buscas. O pseudo-código do algoritmo pode ser visto na Figura 2.13.

Algoritmo de Colisão de Múltiplas Partículas

```
Gera uma solução inicial Old_Config
Best_Fitness = Fitness(Old_Config)
Atualiza o Blackboard
Para n=0 até # de iterações
  Para n=0 até # de partículas
    Perturbation()
    Se Fitness(New_Config) > Fitness(Old_Config)
      Se Fitness(New_Config) > Best_Fitness
        Best_Fitness := Fitness(New_Config)
      Fim-Se
      Old_Config = New_Config
      Exploration()
    Senão
      Scattering()
    Fim-Se
    Atualiza o Blackboard
  Fim-Para
Fim-Para
```

Figura 2.13 - Algoritmo de colisão de múltiplas partículas.

Nesta nova versão, utiliza-se um conjunto de n partículas explorando, de maneira independente, porém colaborativa, o mesmo espaço (Figura 2.14). A introdução do uso de n partículas leva a necessidade da implementação de um mecanismo de comunicação indireta entre as partículas.

A coordenação foi viabilizada através da implementação de uma técnica de *blackboard*, onde o `Best_Fitness`, o melhor resultado obtido pelas partículas, é atualizado

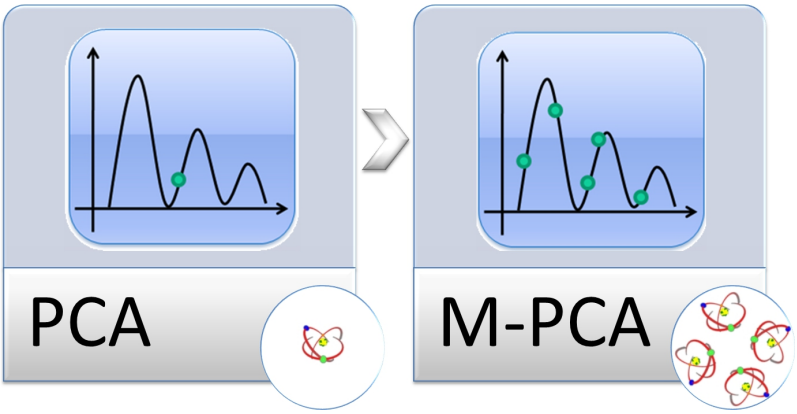


Figura 2.14 - O PCA explora o espaço de busca com uma única partícula. O MPCA, simula a adoção de n-partículas na exploração colaborativa do espaço de buscas, apresentando ganho de performance.

de maneira constante entre todas as partículas envolvidas no processo de busca. Inicialmente a comunicação com a área *blackboard* foi setada para ocorrer a cada iteração do algoritmo, porém verificou-se que a eficiência do algoritmo era afetada pela comunicação extra gerada pelo processo. Para reduzir o custo de comunicação, uma estratégia de ciclos de comunicação foi implementada, reduzindo assim o custo associado (maiores informações na Subsubseção 2.4.1.1).

O pseudo-código apresentado na Figura 2.13 é similar ao pseudo-código apresentado para o PCA, na Figura 2.7, portanto, são executadas, em um pior caso, $n \times n$ chamadas à rotina de cálculo do valor da função objetivo, rotina esta que admite-se ser a mais custosa computacionalmente dentre todo o processo, dando ao PCA uma complexidade estimada de $O(n^2)$.

Com a introdução de um novo laço de controle para as partículas, o número de chamadas à rotina de cálculo do valor da função objetivo pode aumentar e chegar ao pior caso onde são executadas até $n \times n \times n$ chamadas. Um exemplo para este caso é quando o número de partículas se equivale ao número de iterações do algoritmo.

Neste caso inicial, a complexidade do MPCA pode ser estimada como sendo $O(n^3)$. Porém, admitindo-se que o MPCA será executado em p processadores, e garantindo que $p = n_p$, n_p representando o número de partículas em uso, a complexidade do MPCA retorna para o nível de $O(n^2)$, que é a mesma complexidade do PCA canônico (Figura 2.15).

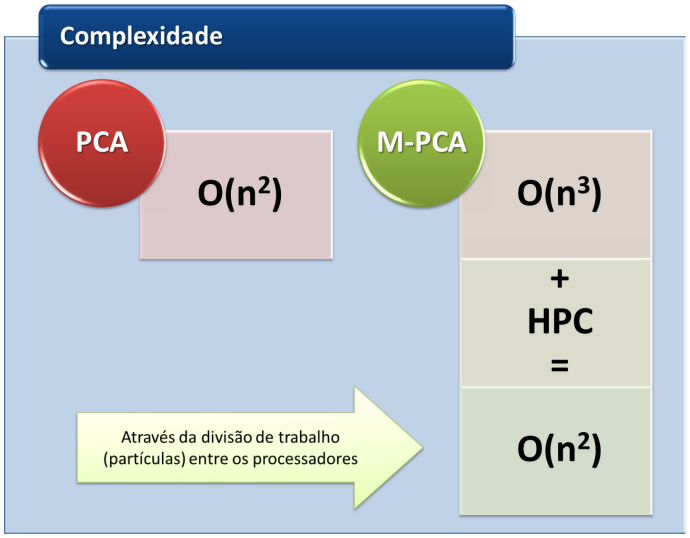


Figura 2.15 - Relação entre a complexidade do PCA e MPCA.

O ajuste do MPCA é provido com a adição de três novos parâmetros:

- N_{proc}: Número de processadores que serão usados;
- N_{part}: Número de partículas que explorarão o espaço de buscas[5];
- $iterSync$: Quantidade de iterações que serão executadas antes da ativação do ciclo de sincronização com o *blackboard*.

Este novo algoritmo foi validado com diversas funções de teste disponíveis na literatura, tal como as funções de Griewank, Easom, Shekel e Rosenbrock. Os resultados foram animadores, principalmente no quesito de tempo computacional (LUZ et al., 2008). Estes resultados são apresentados na Subseção 5.1.1.

Outra característica que deve ser ressaltada é a sua capacidade de escape de ótimos locais. A busca cooperativa, i.e., a troca de informação entre as partículas, auxilia o processo de convergência, uma vez que o processo de espalhamento leva em consideração a informação que está sendo compartilhada entre as partículas.

Esta troca de informação é a peça central deste mecanismo melhorado de escape de ótimos locais, e é o que garante os bons resultados obtidos pelo MPCA. Este método

[5]Para garantir a redução da complexidade do MPCA o número de processadores (N_{proc}) e o número de partículas (N_{part}) deve ser o mesmo.

melhorado e seus resultados são considerados entre as principais contribuições deste trabalho.

Adicionalmente às figuras que apresentam os pseudocódigos do algoritmo, apresenta-se abaixo uma descrição textual do funcionamento do *Multiple Particle Collision Algorithm*:

a) Inicialize as variáveis globais, a função objetivo e atribua os limites do espaço de buscas;
   - Inicialize o ambiente para uso da memória distribuída, para a versão paralela;

b) Aloque o número de partículas de acordo com o número de processadores em uso;

c) Inicialize a semente geradora de números aleatórios;
   - Cada processador deve receber uma semente única, para a versão paralela;

d) Gere uma solução inicial e calcula o seu valor frente a função objetivo;
   - Atribua este primeiro valor à melhor solução atual;

e) Para cada iteração, até que a condição de parada seja satisfeita, faça:
   - Se o mecanismo de atualização da melhor solução via *Blackboard* for ativado, cada processador envia a melhor solução ao processador mestre que por sua vez localiza a melhor solução entre todos e propaga a informação de volta;
   - Para cada partícula alocada ao processador, faça:
     i. Chame a rotina de perturbação e faça:
        A. Aplique uma perturbação aleatória na posição da partícula e calcule seu novo valor com a função objetivo;
     ii. Se uma melhor solução for encontrada, faça:
        A. Verifique se a nova solução é melhor que a solução global e atribua o novo valor se for o caso;
        B. Chame a rotina de exploração local e faça:

- Para cada iteração, até que a condição de parada seja satisfeita, faça:
  - Aplique pequenas perturbações na solução e calcule o novo valor da função objetivo;
  - Verifique se a nova solução é melhor que a solução atual e atribua o novo valor se for o caso;

iii. Se uma melhor solução não foi encontrada, chame a rotina de espalhamento:

A. Calcule a probabilidade de espalhamento;

B. Se o espalhamento for ativado, reinicialize a posição da partícula em algum ponto aleatório do espaço de buscas;

- Chame a rotina de exploração local e faça:
  - Para cada iteração, até que a condição de parada seja satisfeita, faça:
  - Aplique pequenas perturbações na solução e calcule o novo valor da função objetivo;
  - Verifique se a nova solução é melhor que a solução atual e atribua o novo valor se for o caso;

f) Faça uma atualização final da melhor solução via *Blackboard* para garantir que a melhor solução da última iteração de cada processador seja levada em conta;

g) Imprima o melhor resultado obtido após o término do algoritmo.

2.4.1.1 Análise do desempenho do MPCA

Estimar a complexidade de um novo algoritmo é um passo crucial para a validação e até mesmo valorização da técnica que está sendo apresentada. Se para a execução de uma dada tarefa, vários algoritmos de mesma precisão estão disponíveis, a escolha lógica recai, em geral, para aquele de menor complexidade computacional (ZIVIANI, 2004; SZWARCFITER; MARKENZON, 1994).

A análise de complexidade de um algoritmo tem por objetivo a avaliação da eficiência deste algoritmo em termos de tempo ou espaço computacional requeridos (SZWARCFITER; MARKENZON, 1994).

Quando fazemos uma análise da complexidade relativa à execução do algoritmo PCA canônico, podemos verificar, segundo o trecho principal de seu código fonte, que temos uma complexidade $O(n^2)$, dada a existência de dois laços aninhados.

O algoritmo de múltiplas partículas MPCA força a inserção de um novo laço, em um nível superior aos laços existentes, o que leva a construção de um novo trecho principal, que possui complexidade $O(n^3)$.

No momento da paralelização do MPCA se este laço for distribuído para n processadores, a complexidade do algoritmo retorna para $O(n^2)$ (Figura 2.15). Esta redução da complexidade é possível através da implementação em um ambiente de alto desempenho, com características de um esquema SIMD, tal como o apresentado na seção sobre computação de alto desempenho (veja a Seção 2.3).

Fazendo a análise baseada na classificação apresentada por Sambatti (2004), o algoritmo de MPCA é implementado em um sistema de software que tem por base a biblioteca MPI e viabilizado através de um hardware com arquitetura de um sistema distribuído.

A implementação do MPCA também pode ser dada em um ambiente SISD, o equivalente a um computador pessoal comum em implementação serial, porém neste caso a complexidade relacionada à execução do algoritmo é cúbica, $O(n^3)$, o que faz com que uma das principais vantagens desta variante, que pode ser sintetizada como sendo a rápida aplicação colaborativa de n instâncias do algoritmo original levando a uma exploração efetivamente paralela do espaço de buscas em um mesmo período de tempo, não se aplique.

Podemos então fazer uma análise de *speedup*, que é definida como a razão entre o tempo de execução do algoritmo sequencial (T_s) e o tempo de execução do algoritmo paralelo (T_p), seguida de uma análise de eficiência (E_p), que é definida pela razão entre o *speedup* e o número de processadores (p), com o objetivo de verificar o quanto o paralelismo foi explorado no algoritmo.

Adotando a resolução da função de Easom com 100 iterações de busca local e $LI_{SP} = 0,8$ e $LS_{SP} = 1,2$, sendo condição de parada a execução de 10^6 avaliações da função objetivo, para fins de comparação, obtemos os seguintes valores na análise de desempenho do MPCA: $S_p = 4,447$ e $E_p = 0,4447$, que comprovam a eficiência do algoritmo paralelizado (a Figura 2.16 apresenta a curva de *speedup* para

2, 5 e 10 processadores).

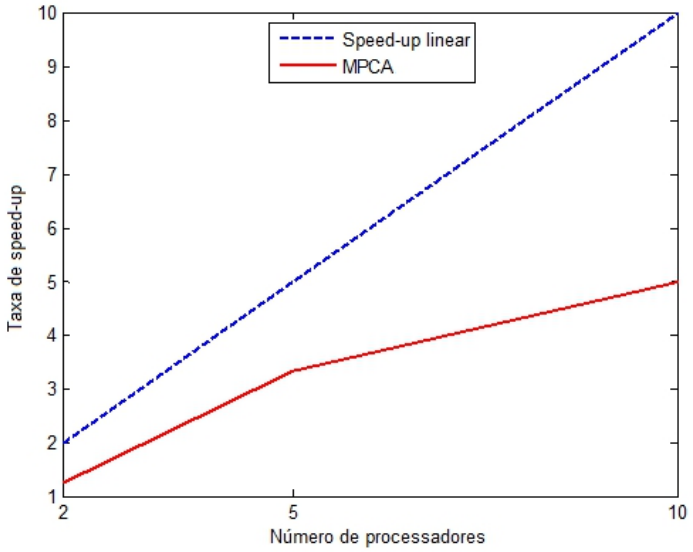


Figura 2.16 - *Speedup* do MPCA para 2, 5 e 10 processadores com atualização do *Blackboard* a cada iteração do algoritmo.

A curva de *speedup* para o MPCA apresenta resultados abaixo da linha linear, porém dentro das expectativas, tal como o ilustrado na Figura 2.6. A sublinearidade é explicada pela grande quantidade de comunicação existente entre os processadores. A responsável por esta comunicação é a nova função de atualização do *Blackboard*, usada para propagar a informação do `Best_Fitness` entre os processadores.

Uma maneira de aumentar o desempenho do MPCA é adotar um esquema de atualização da informação do `Best_Fitness` após um determinado número de iterações, i.e., implementar um ciclo intermitente de comunicação. Tal característica possibilita a redução da taxa de troca de mensagens entre os processadores, resultando em uma melhoria da performance do MPCA em ambiente paralelo.

Os resultados obtidos para o mesmo problema apresentado acima, adicionado o esquema de comunicação por ciclos a cada 1.000 iterações, i.e., execução de 1.000 iterações independentes em cada partícula até a implementação da comunicação entre os processadores, levou a uma melhora do valor de *speedup* para $S_p = 6,933$ com uma nova eficiência calculada para $E_p = 0,6933$. A Figura 2.17 apresenta a nova curva de *speedup* para 2, 5 e 10 processadores, agora adicionados de um ciclo

intermitente de comunicações.

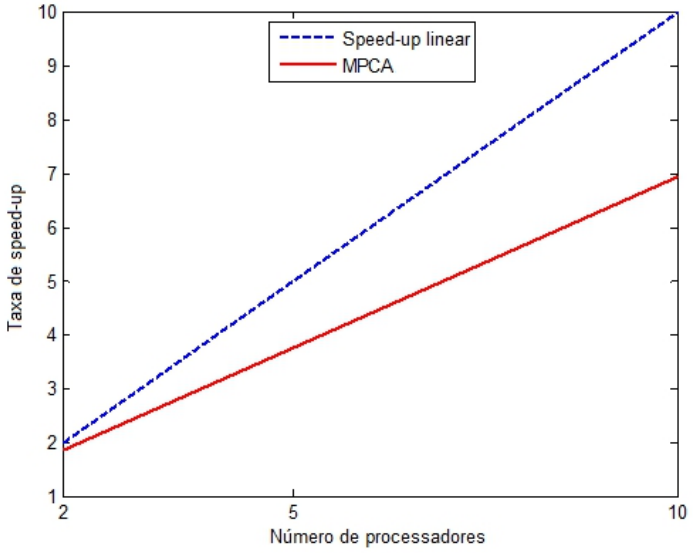


Figura 2.17 - *Speedup* do MPCA para 2, 5 e 10 processadores com atualização do *Blackboard* em ciclos intermitentes. A obtenção destes resultados utilizou um ciclo definido a cada 1.000 iterações do laço principal.

O aumento da eficiência obtida pelo ciclo intermitente de comunicação entre os processadores é obviamente afetada pelo valor escolhido para a ocorrência da sincronização. Ao mesmo tempo, este mecanismo de sincronização é essencial para o bom funcionamento do MPCA, portanto sincronizações muito espaçadas não são recomendadas. Diversos valores foram testados e este novo parâmetro é dependente do problema a ser resolvido, porém sincronizações com ocorrências entre 100 e 1.000 iterações apresentam bons resultados, tanto para eficiência paralela do algoritmo quanto para a eficiência numérica na resolução dos problemas.

Além de uma análise de *speedup* e eficiência, o MPCA também foi submetido a uma instrumentação automática de código com o uso do *Cray Performance Measurement & Analysis Tool* (CrayPat), uma ferramenta de análise de desempenho de alto nível usada para a identificação automática de oportunidades de otimização de código em uso nos sistemas Cray. A visualização dos resultados obtidos pelo CrayPat foram efetuados com o Cray Apprentice[2], uma ferramenta de visualização de dados gerados em formato XML. Maiores informações sobre o uso das ferramentas usadas nas análises podem ser obtidas na documentação oficial da Cray (CRAY, 2011).

A instrumentação automática usando a ferramenta de *profiling* do CrayPat fornece informações derivadas de uma amostragem contínua do estado do programa ao longo de sua execução. Esta amostragem permite inferir quais funções ou sub-rotinas apresentam maior carga de execução e em alguns casos sugestões de melhora são apresentadas ao usuário.

A primeira versão do MPCA foi submetida a esta instrumentação automática. O primeiro resultado, relativo à análise da quantidade de chamadas a funções, e consequentemente do tempo gasto em cada função, é apresentado na Figura 2.18. Nota-se que ao longo da amostragem do programa, mais de 95% das chamadas a sub-rotinas é direcionada à função geradora de números aleatórios. Ao mesmo tempo, $74,4\%$ do tempo de execução do programa é gasto com a geração de números aleatórios. Outros $18,7\%$ do tempo são gastos na rotina de atualização do *Blackboard*, especificamente enquanto se espera receber a mensagem sobre qual é o `Best_Fitness` atual. O tempo de cálculo da função objetivo é relativamente pequeno pois a função usada para esta análise é a função de Griewank (veja a Equação 5.4 e a Tabela 5.1 para maiores informações).

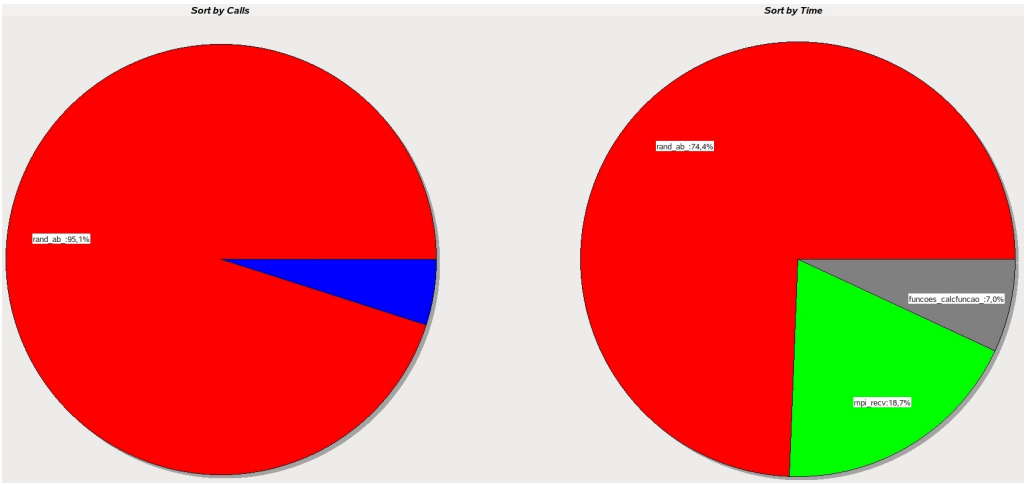


Figura 2.18 - Análise das chamadas e tempo de execução de funções do MPCA em sua primeira versão com o uso do PerfTools.

A Figura 2.19 apresenta a árvore de chamadas (*Call Tree*) do MPCA, gerado pelo CrayPat e visualizado do Apprentice[2]. A altura das caixas representa o tempo gasto em cada função, ou sub-rotina. O caminho traçado em vermelho indica um potencial gargalo identificado pelo CrayPat.

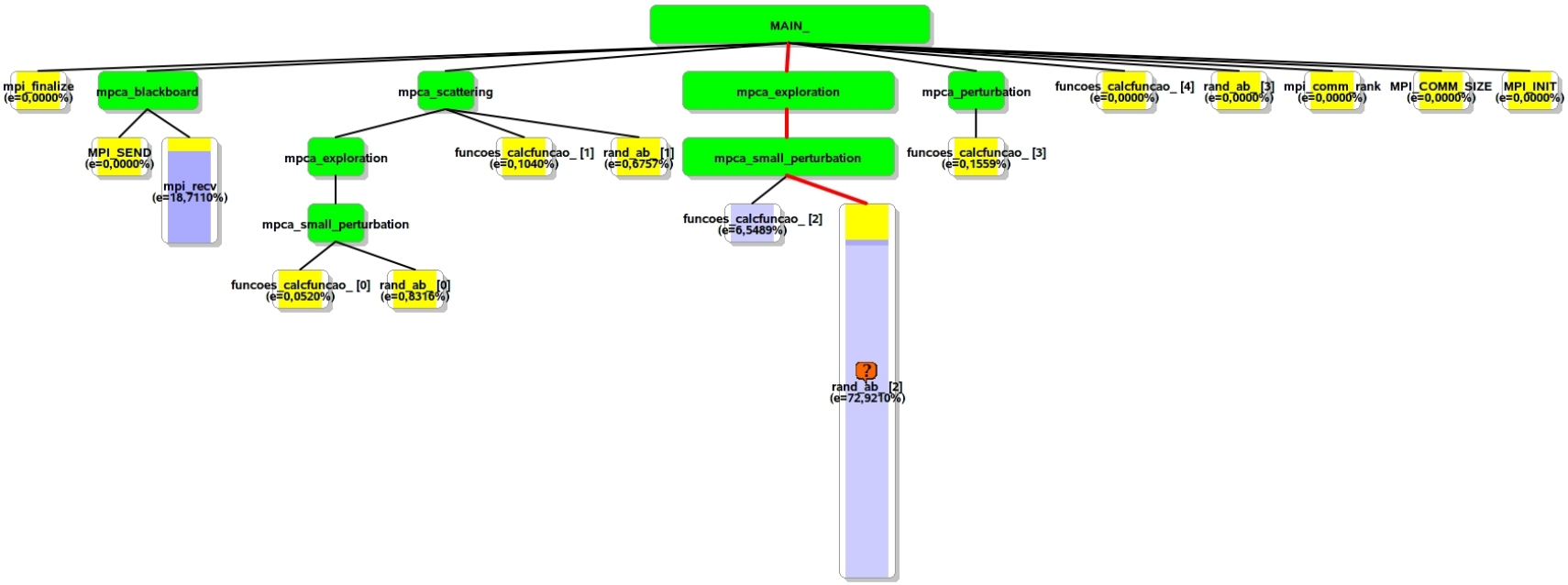


Figura 2.19 - Árvore de chamadas de funções do MPCA em sua primeira versão com o uso do PerfTools.

Duas sugestões fornecidas pelo CrayPat foram adotadas em uma nova versão do MPCA:

a) Reescrever a função `rand_ab` de maneira mais eficiente;

b) Substituir o esquema de envio e recebimento de mensagens inicialmente implementado com o binômio `mpi_send` e `mpi_recv` por uma propagação geral de mensagem com o uso de `mpi_bcast`.

A primeira sugestão foi posta em prática com a substituição da rotina de geração de números aleatórios. Na primeira versão, o gerador era baseado no algoritmo fornecido pelo *Numerical Recipes in Fortran 77* (PRESS et al., 1992). Este gerador requer uma chamada para cada número a ser gerado. Em uma nova versão, apresentada pelo *Numerical Recipes in Fortran 90* (PRESS et al., 1996), existe a possibilidade de se solicitar a geração de um único número aleatório assim como também um vetor de n números aleatórios com uma única chamada à função. Com esta nova implementação espera-se uma redução no *overhead* causado pelo seu uso constante.

A segunda sugestão também foi implementada com a simples substituição da chamada dupla de um `mpi_send` e de um `mpi_recv` dentro da sub-rotina de atualização do *Blackboard*. Na nova versão uma única chamada a um `mpi_bcast` executa a atualização do valor de `Best_Fitness` para os processadores.

Após a implementação das sugestões, uma nova instrumentação do algoritmo foi efetuada. Os resultados para o número de chamadas (torta da esquerda no Figura 2.20) mostram uma redução significativa no número de chamadas às funções geradoras de números aleatórios. Mesmo somando o número de chamadas às funções que geram escalares e vetores ($13,3\%$ e $42,7\%$ respectivamente) esta quantidade se distingue significativamente do valor inicial. O tempo gasto gerando números aleatórios manteve o mesmo patamar, com uma diferença que pode ser atribuída ao processo de amostragem aleatória utilizado. Já o tempo gasto com a comunicação via `mpi_send` e `mpi_recv` foi reduzido em mais de 10% com a adoção do novo esquema de propagação via `mpi_bcast`.

A Figura 2.21 mostra a nova árvore de chamadas do MPCA após a adoção das sugestões iniciais do CrayPat. Ainda se nota a grande parcela de tempo usado na geração de números aleatórios, porém com a nova implementação o instrumentador automático redirecionou o gargalo para o tempo gasto na comunicação.

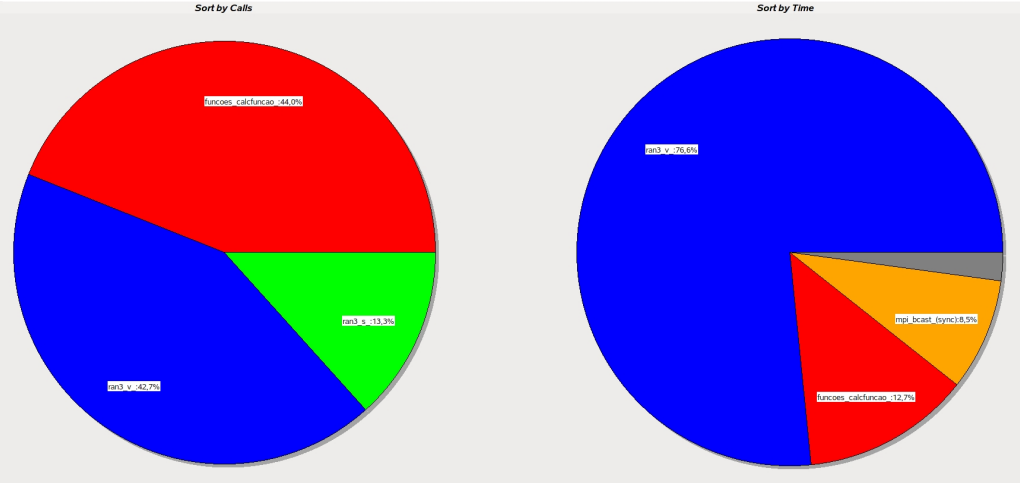


Figura 2.20 - Análise das chamadas de funções do MPCA melhorado com as sugestões advindas do uso do PerfTools.

O fato da fatia maior de tempo ser gasto na geração de números aleatórios evidencia a grande dependência dos métodos estocásticos nestes processos. A utilização de um bom gerador de números aleatórios é altamente recomendada (PRESS et al., 1996).

Em adição à nova instrumentação, que demonstrou melhorias na execução do MPCA, o *speedup* e a eficiência foram recalculados. Tendo por base os mesmos parâmetros adotados para os cálculos de *speedup* apresentados anteriormente, o algoritmo melhorado com as sugestões do CrayPat revelou uma melhora na execução para 10 processadores, agora apresentando um $S_p = 8,1096$, e consequentemente elevado a eficiência calculada para $E_p = 0,8109$. A Figura 2.22 apresenta a última curva de *speedup* obtida.

2.5 Algoritmo de vaga-lumes

O algoritmo de vaga-lumes (*Firefly Algorithm*, FA) foi proposto por Xin-She Yang na Universidade de Cambridge em 2007 (YANG, 2008). Este algoritmo é baseado na característica bioluminescente de vaga-lumes, insetos coleópteros notórios por suas emissões luminosas. Segundo Yang (2008), a biologia ainda não tem um conhecimento completo para determinar todas as utilidades que esta luminescência pode trazer ao vaga-lume, mas pelo menos três funções já foram identificadas:

a) como uma ferramenta de comunicação e atração para potenciais parceiros na reprodução;

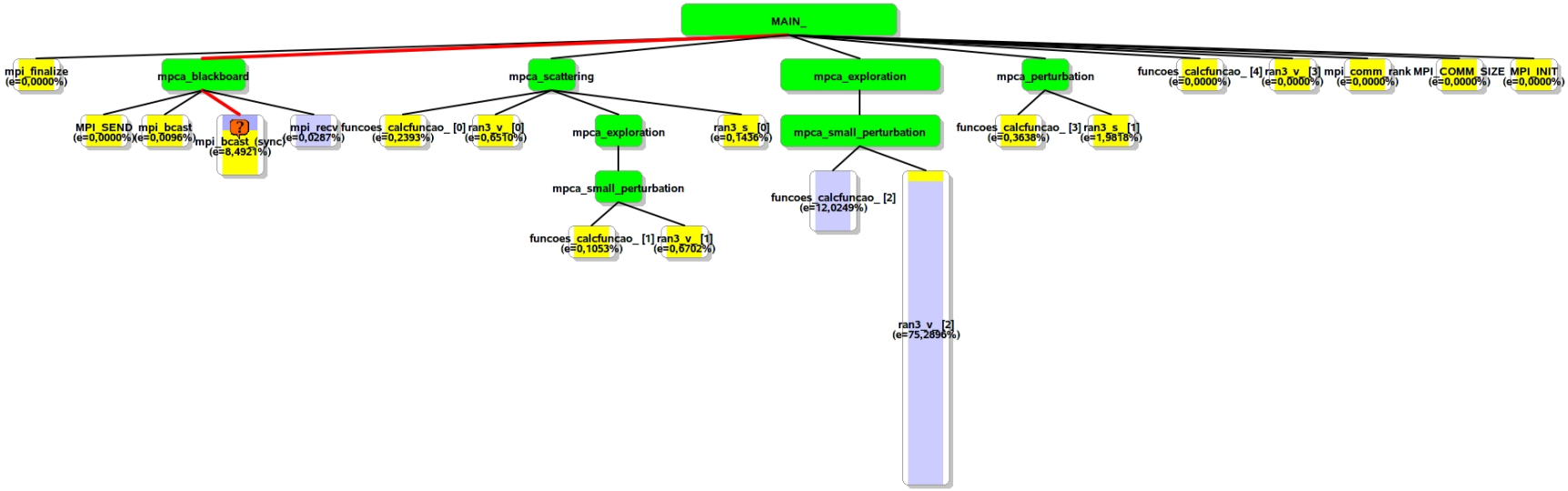


Figura 2.21 - Árvore de chamadas de funções do MPCA melhorado com as sugestões advindas do uso do PerfTools.

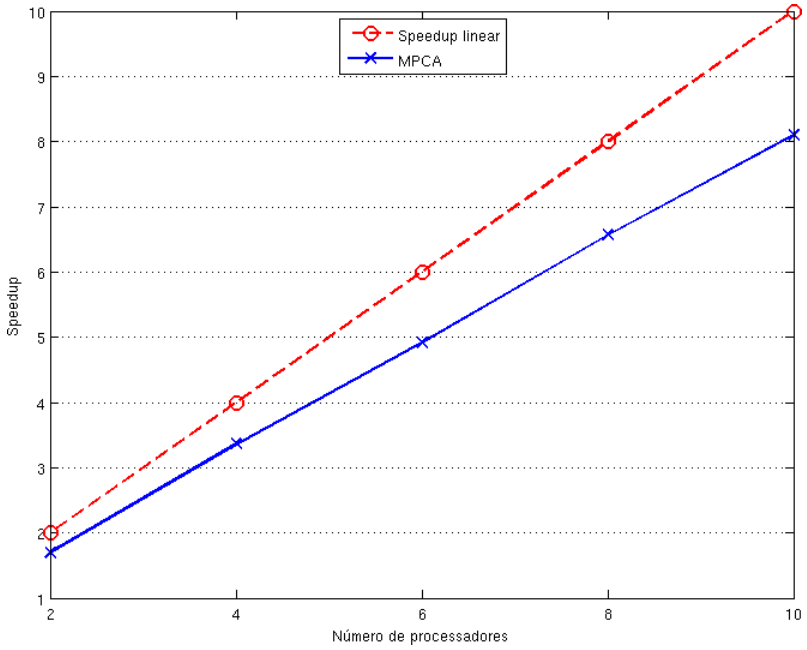


Figura 2.22 - Novo *speedup* do MPCA, após adição das sugestões do CrayPat, para 2, 4, 6, 8 e 10 processadores com atualização do *Blackboard* em um ciclo de 1.000 iterações intermitentes.

b) como uma isca para atração de eventuais presas para o vaga-lume;

c) como um mecanismo de alerta para potenciais predadores: lembrando-os de que vaga-lumes tem um "gosto amargo".

A primeira função da bioluminescência advém do fato observado de que em determinadas espécies de vaga-lumes a taxa de intermitência e a intensidade das emissões luminosas é parte essencial do mecanismo que atrai ambos os sexos para o ritual de acasalamento. Na maioria dos casos, as fêmeas são atraídas pelo brilho emitido pelos machos. Outra característica observada nos vaga-lumes, quando da existência de uma grande quantidade destes em uma área comum, é o fenômeno de sincronização para a emissão dos flashes luminosos, evidenciando uma característica de auto-organização emergente (LEWIS; CRATSLEY, 2008).

Para a implementação definida por Yang (2008), posteriormente melhorada em Yang (2010b), três regras simplificadas foram criadas para delinear o funcionamento do algoritmo:

a) os vaga-lumes não possuem sexo, portanto qualquer vaga-lume poderá

atrair ou ser atraído;

b) a atratividade é proporcional ao brilho emitido e decai conforme aumenta a distância entre os vaga-lumes (regra baseada em observação do comportamento animal);

c) o brilho emitido por um vaga-lume é determinado pela sua avaliação frente à função objetivo (i.e., quanto melhor avaliado, mais brilhante).

Para uma melhor compreensão do FA, duas características do algoritmo devem ser destacadas: como se dá a variação da intensidade da luz percebida pelo vaga-lume; e como é formulada a atratividade entre os vaga-lumes. Ainda segundo Yang (2008), para simplificação do modelo, a atratividade de um vaga-lume é determinada pela intensidade da luz emitida, e a determinação da intensidade emitida é função de sua avaliação.

Assim, a intensidade de emissão de luz por parte de um vaga-lume é proporcional à função objetivo, i.e., $I(x) \propto J(x)$, porém a intensidade de luz percebida por um vaga-lume decai em função da distância entre os vaga-lumes, dada a absorção da luz pelo meio. Logo, a intensidade percebida por um vaga-lume é dada por:

$$I(r) = I_0 e^{\gamma r^2}, \tag{2.9}$$

em que I_0 é a intensidade da luz emitida (equivalente ao valor da função objetivo); r é a distância Euclidiana entre os vaga-lumes i e j, sendo i o vaga-lume mais brilhante e j o vaga-lume menos brilhante; e γ é o parâmetro de absorção da luz pelo meio.

Desta maneira o fator de atratividade β pode ser formulado como:

$$\beta = \beta_0 e^{-\gamma r^m}, \tag{2.10}$$

em que β_0 é a atratividade para uma distância $r = 0$, e pode ser fixo em $\beta_0 = 1$; e m é o parâmetro de controle da influência da distância.

Assim, a movimentação em um dado passo de tempo t de um vaga-lume i em direção a um melhor vaga-lume j é definida como:

$$x_i^t = x_i^{t-1} + \beta\left(x_j^{t-1} - x_i^{t-1}\right) + \alpha\left(rand - \frac{1}{2}\right), \tag{2.11}$$

em que, o segundo termo do lado direito da equação insere o fator de atratividade β enquanto que o terceiro termo, regulado pelo parâmetro α, regula inserção de certa aleatoriedade no caminho percorrido pelo vaga-lume, $rand$ é um número aleatório entre 0 e 1.

O pseudocódigo do funcionamento do FA é apresentado na Figura 2.23.

Firefly Algorithm

Início
Definir a função objetivo $J(x), x = (x_1, \ldots, x_d)^T$
Definir os parâmetros n, α, γ, β_0, *MaxGeracoes*
Gerar a população inicial de vagalumes x_i $(i = 1, 2, \ldots, n)$
Para $t = 1$ até *MaxGeracoes*
Calcular a intensidade da luz I_i para x_i proporcionalmente a $J(x_i)$
Para $i = 1$ até n
Calcular o fator de atratividade β de acordo com $e^{-\gamma r^2}$
Mover o vagalume i em direção aos vagalumes mais brilhantes
Verificar se o vagalume está dentro dos limites
Fim-Para
Fim-Para
Pós-processar e visualizar os resultados
Fim

Figura 2.23 - Pseudocódigo para o *Firefly Algorithm*. Adaptado de (YANG, 2008).

2.5.1 Vaga-lumes com predação

A inspiração biológica advinda da observação do comportamento dos vaga-lumes pode ser incrementada com a adição de um mecanismo de seleção natural baseado na predação dos indivíduos menos aptos.

A seleção natural é um processo de evolução descrito pelo naturalista inglês Charles Darwin que visa explicar os fenômenos de adaptação e especiação[6] observada em fósseis.

Tomando por base a terceira função da bioluminescência descrita na seção anterior,

[6]Especiação é o processo evolutivo pelo qual as espécies vivas se formaram.

que apresenta a capacidade de alertar potenciais predadores de que vaga-lumes tem um "gosto amargo", pode-se assumir o caso em que os piores vaga-lumes são eliminados da população geral de vaga-lumes por um processo predatório qualquer, baseado no valor da sua função objetivo.

Desta maneira, este trabalho apresenta esta variante do *Firefly Algorithm* adicionado de um componente de seleção natural baseado em predação, doravante denominado *Firefly Algorithm with Predation* (FAP). Esta implementação pode seguir o esquema apresentado na Figura 2.24.

Firefly Algorithm com Seleção Natural

Início
  Definir a função objetivo $J(x)$, $x = (x_1, \ldots, x_d)^T$
  Definir os parâmetros $n, \alpha, \gamma, \beta_0, MaxGeracoes, n_{selNat}, tSelNat$
  Gerar a população inicial de vagalumes x_i $(i = 1, 2, \ldots, n)$
  Para $t = 1$ até $MaxGeracoes$
    Se $tSelNat$
      Para $i = n_{selNat} + 1$ até n
        Gera um novo vagalume
      Fim-Para
    Fim-Se
    Calcular a intensidade da luz I_i para x_i proporcionalmente a $J(x_i)$
    Para $i = 1$ até n
      Calcular o fator de atratividade β de acordo com $e^{-\gamma r^2}$
      Mover o vagalume i em direção aos vagalumes mais brilhantes
      Verificar se o vagalume está dentro dos limites
    Fim-Para
  Fim-Para
  Pós-processar e visualizar os resultados
Fim

Figura 2.24 - Pseudocódigo para o *parallel Firefly Algorithm with Predation.*

Dois novos parâmetros são necessários para a implementação deste esquema de seleção natural: a quantidade de vaga-lumes que não serão predados, representado por n_{selNat}; e a frequência com que o esquema de seleção natural será aplicada, representado por $tSelNat$.

Os novos vaga-lumes são reinicializados em posições aleatórias dentro do espaço de busca e o algoritmo itera convencionalmente até o seu critério de parada final ou até

que outro passo de seleção natural seja ativado.

O padrão OpenMP (OPENMP, 2011) foi usado para a implementação do algoritmo de vaga-lumes com predação em multi-processadores por meio de múltiplas *threads* de um único processo. As diretivas explicitaram a divisão de trabalho nos laços relacionados às iterações dos vaga-lumes, i.e., a geração e evolução das soluções candidatas foi paralelizada. Assim, a denominação do algoritmo pode ser expandida para *parallel Firefly Algorithm with Predation* (pFAP).

O desenvolvimento de algoritmos que atuam em ambientes de memória compartilhada requer atenção especial para a garantia de que o acesso a variáveis, funções e sub-rotinas por diversas *threads* simultaneamente não gere resultados inconsistentes. Para as variáveis, a definição do escopo de acesso com diretivas OpenMP (principalmente se privada ou compartilhada) é necessária para evitar maiores problemas. Funções e sub-rotinas precisam ser escritas de forma a garantir que suas chamadas sejam *thread-safe*, i.e., prontas para acesso simultâneo por várias *threads*. Meta-heurísticas estocásticas como o FAP são fortemente dependentes de geradores de números aleatórios e estas rotinas, geralmente bibliotecas prontas, precisam desta garantia para que a geração de números aleatórios ocorra sem problemas. O pFAP se valeu das rotinas fornecidas pela Intel®Math Kernel Library (MKL) para a geração, *thread-safe*, dos números aleatórios usados durante sua execução (INTEL, 2011).

Os resultados observados demonstram a viabilidade da utilização deste esquema de predação. Esta característica permite que o algoritmo escape de ótimos locais em um tempo menor do que sua versão canônica. Nos exemplos que serão apresentados na seção de validação, podemos observar que a quantidade de iterações necessárias para atingirmos um melhor ponto no espaço de buscas é uma fração da quantidade de iterações necessárias para o algoritmo canônico.

Adicionalmente às figuras que apresentam os pseudocódigos do algoritmo, apresenta-se abaixo uma descrição textual do funcionamento do *Firefly Algorithm with Predation* incrementado com sua implementação em ambiente paralelo (pFAP):

a) Inicialize as variáveis globais, a função objetivo e atribua os limites do espaço de buscas;
   - Atribua o escopo para as variáveis de memória compartilhada, para a

versão paralela (variáveis globais e locais às *threads*);

b) Inicialize a semente geradora de números aleatórios;

c) Gere a população de vaga-lumes aleatoriamente no espaço de buscas;

   – Distribua a geração inicial dos vaga-lumes entre as *threads*, para a versão paralela;

d) Para cada iteração, até que a condição de parada seja satisfeita, faça:

   – Se o mecanismo de predação for ativado, faça:

      i. Ordene os vaga-lumes de acordo com o valor da função objetivo;

      ii. Os piores vaga-lumes são predados e novos vaga-lumes são inicializados aleatoriamente no espaço de buscas;

         A. Distribua a geração de novos vaga-lumes entre as *threads*, para a versão paralela;

   – Calcule o valor da função objetivo de todos os vaga-lumes com base em suas posições no espaço de buscas;

      i. Distribua o cálculo da função objetivo entre as threads, para a versão paralela;

   – Ordene os vaga-lumes de acordo com o valor da função objetivo;

   – Verifique se o melhor vaga-lume da iteração em curso supera a melhor solução encontrada até o momento e atualize se for o caso;

   – Para cada vaga-lume, com exceção do melhor:

      i. Distribua os vaga-lumes entre as *threads*, para a versão paralela;

      ii. Movimente o vaga-lume em direção a um vaga-lume com melhor solução de acordo com a Equação 2.11;

   – Aplique um movimento aleatório ao melhor vaga-lume;

e) Imprima o melhor resultado obtido após o término do algoritmo.

2.5.1.1 Análise do desempenho do pFAP

A análise do desempenho da nova versão do algoritmo de vaga-lumes com predação, implementada em um ambiente de memória compartilhada via OpenMP, utilizou o

cálculo do *speedup* e da eficiência do novo algoritmo tal como o descrito na Subseção 2.3.1 com a Equação 2.6 e Equação 2.7, respectivamente.

O processador utilizado para a análise do desempenho deste algoritmo é um Intel®Core™i7-820QM, de quatro núcleos e equipado com tecnologia Intel®Hyper-Threading, i.e., quatro núcleos com capacidade de executar oito *threads* simultaneamente[7]. O sistema operacional adotado foi o Ubuntu Linux 10.04 LTS 64-bit.

A Tabela 2.2 apresenta o *speedup* e a eficiência obtida pela análise do algoritmo de vaga-lumes com predação para a função de Ackley, em 10 experimentos com 250.000 iterações cada e com o uso de 100 vaga-lumes. Para este exemplo, o número de vaga-lumes predados equivale a 90% da população inicial e o intervalo de predação é de 10.000 iterações.

O *speedup* é super-linear para dois e três processadores. Com quatro processadores o resultado é pouco abaixo do linear, mas especula-se que neste caso, uma vez que o processador usado possui quatro núcleos, um deles deve obrigatoriamente dividir-se entre a tarefa de executar o pFAP e controlar o sistema operacional, reduzindo desta maneira a eficiência calculada.

Tabela 2.2 - *Speedup* e eficiência do pFAP com OpenMP.

Threads	*Speedup*	Eficiência
2	2,97	1,48
3	3,09	1,03
4	3,79	0,94

A Figura 2.25 apresenta a curva de *speedup* baseada nos dados da Tabela 5.6.

A Tabela 2.3 apresenta os resultados de *speedup* e eficiência estendidos para até oito *threads*. A Figura 2.26 adiciona a informação de *speedup* ampliada na Tabela 2.3 pelo uso da tecnologia de Hyper-Threading (MARR et al., 2002).

Os valores apresentados com o uso das oito *threads* não implicam, necessariamente, em resultados definitivos, uma vez que a tecnologia de Hyper-Threading tenta ace-

[7]Maiores informações, incluindo detalhes técnicos sobre o processador utilizado podem ser obtidas em: `http://ark.intel.com/pt-br/products/43124/Intel-Core-i7-820QM-Processor-(8M-Cache-1_73-GHz)`

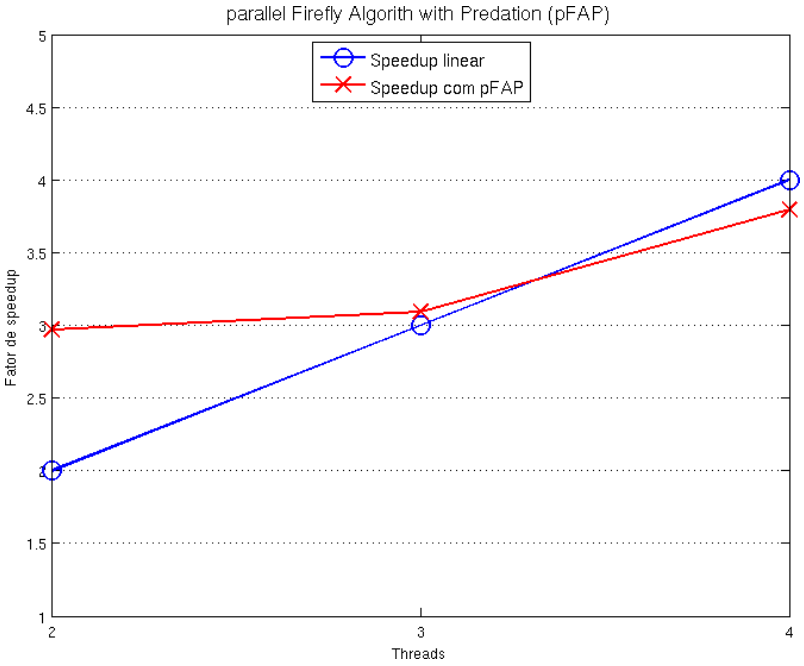


Figura 2.25 - Fator de *speedup* obtido para 2, 3 e 4 *threads* executando o pFAP.

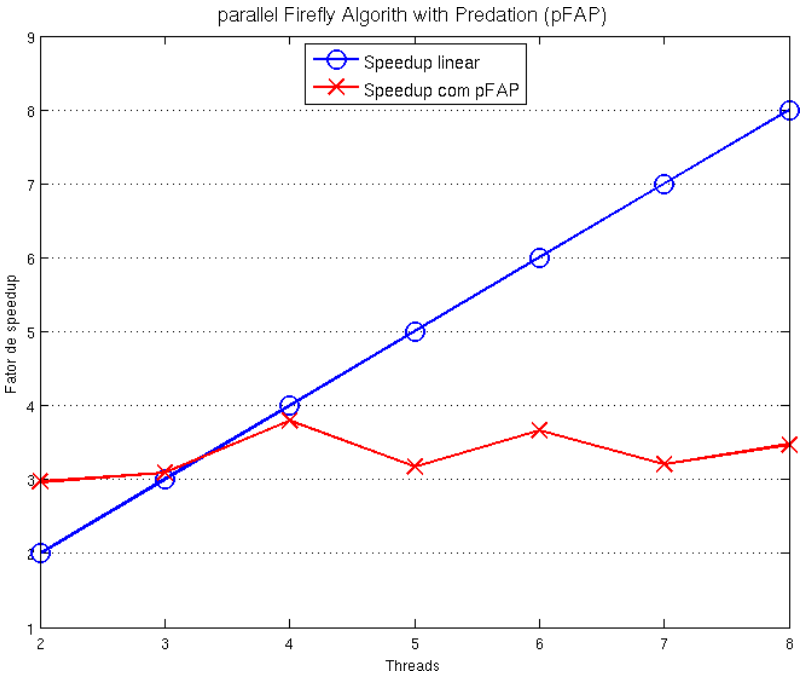


Figura 2.26 - Fator de *speedup* obtido para até 8 *threads* executando o pFAP.

Tabela 2.3 - *Speedup* e eficiência estendida com Hyper-Threading do pFAP com OpenMP.

Threads	*Speedup*	Eficiência
5	3,17	0,63
6	3,66	0,61
7	3,20	0,45
8	3,47	0,43

lerar a execução de programas via melhorias de software e não de hardware (ADVANCED MICRO DEVICES, 2010).

2.6 Seleção de parâmetros

Os métodos de otimização, em especial aqueles baseados em procedimentos estocásticos, possuem uma série de parâmetros ajustáveis por parte do Cientista Programador com o objetivo de controlar o seu comportamento ao longo do processo de busca.

Estes parâmetros definem desde as condições de parada do algoritmo, o número de indivíduos em métodos populacionais, as características da vizinhança entre os indivíduos e especialmente como o algoritmo estabelece o *trade-off* entre a exploração global e a intensificação local no espaço de busca.

Em grande parte do tempo, o ajuste destes parâmetros é feito empiricamente, com o Cientista Programador utilizando conhecimento *a priori* e *know-how* do comportamento do método e das características do problema a ser resolvido.

Foi parte integrante deste trabalho a orientação de um aluno de iniciação científica, com financiamento via Programa Institucional de Bolsas de Iniciação Científica (PIBIC), para o estudo de um método automático de ajuste de parâmetros[8].

O método, denominado *Statistical Racing*, foi aplicado no ajuste automático dos parâmetros do algoritmo de Recozimento Simulado (*Simulated Annealing*) para a resolução de funções de teste padrões. O algoritmo de Recozimento Simulado, o método *Statistical Racing* e as considerações advindas dos resultados obtidos serão apresentados nas subseções a seguir.

[8]O trabalho desenvolvido por Miranda (2011) foi agraciado com "*Menção Honrosa*" pelo excelente desempenho na apresentação durante o Seminário de Iniciação Científica do INPE (SICINPE 2011).

2.6.1 Recozimento Simulado

O algoritmo de Recozimento Simulado (*Simulated Annealing*, SA) foi descrito inicialmente por Kirkpatrick et al. (1983) tendo por base o trabalho de Metropolis et al. (1953) que provê uma solução aproximada para uma grande variedade de problemas matemáticos com técnicas de amostragem estatística de experimentos em um computador (FLOUDAS; PARDALOS, 2009).

O nome da técnica advém de sua analogia com o processo físico de têmpera (*annealing*), onde um sólido é aquecido e resfriado lentamente com o objetivo de reorganizar as moléculas em seu interior para a obtenção de uma forma livre de defeitos (HENDERSON et al., 2003).

A força de uma estrutura depende da taxa de resfriamento de um sólido, em especial metais. Se o resfriamento mencionado anteriormente for feito o suficientemente lento, a configuração resultante do rearranjo das moléculas do sólido garantirá uma melhor integridade estrutural (WEISE, 2009).

O algoritmo proposto para o SA (Figura 2.27) se baseia, portanto, em um fenômeno conhecido da termodinâmica para a solução de problemas de otimização.

Algoritmo de Recozimento Simulado

```
Início
      s := s0; e := E(s)
      sb := s; eb := e
      k := 0
      Enquanto k < kmax e e > emax
            sn := vizinho(s)
            en := E(sn)
            Se en < eb então
                  sb := sn; eb := en
            Fim-Se
            Se P(e, en, temp(k/kmax)) > random() então
                  s := sn; e := en
            Fim-Se
            k := k + 1
      Retorna sb
Fim
```

Figura 2.27 - Algoritmo de recozimento simulado.

Uma das principais características do SA, sendo inclusive seu fator de sucesso, é sua capacidade de escapar de ótimos locais. Este escape se caracteriza por sua habilidade de executar movimentos de subida (*hill-climbing*) no espaço de soluções. Estes mo-

vimentos de subida são implementados com o aceite temporário de soluções piores que a atual, que se traduz como a permissividade de deterioração do valor da função objetivo (FLOUDAS; PARDALOS, 2009)[9].

Um dos principais parâmetros do SA, a temperatura do sistema, regula o funcionamento da capacidade de escape. Ao longo do funcionamento do algoritmo, a temperatura inicial (definida pelo Cientista Programador) decai em função de um parâmetro de resfriamento e a possibilidade de aceite de uma solução pior que a atual decai da mesma forma, i.e., conforme a temperatura tente a zero os movimentos de subida se tornam mais raros.

A Tabela 2.4 expressa uma analogia entre condições existentes em um sistema físico real e estados/variáveis do algoritmo de recozimento simulado.

Tabela 2.4 - Analogia entre o sistema físico e o recozimento simulado.

Sistema Físico	Problema de Otimização
Estado do sistema	Solução
Posição molecular	Variáveis de decisão
Energia	Função objetivo
Estado estável	Solução ótima global
Estado metaestável	Ótimo local
"Quenching" rápido	Busca local
Temperatura	Parâmetro de controle T
Recozimento cuidadoso	Recozimento simulado

Fonte: Adaptado de Talbi (2009)

2.6.2 Statistical Racing

Os algoritmos de corrida, ou *racing algorithms*, foram propostos por Maron e Moore (1997) para resolver um problema de aprendizado por máquina. Este método não se vale da força bruta, mas sim de uma avaliação estatística da eficácia dos algoritmos, via método ANOVA[10], para eliminar candidatos fracos e possibilitar que algoritmos promissores sejam mais bem testados.

[9]Este movimento de subida como deterioração do valor da função objetivo é característica da "minimização" de funções. Para o caso de maximização, a deterioração do valor da função objetivo é obtida com um movimento de descida.

[10]*Analysis of Variance*.

O funcionamento dos algoritmos de corrida seguem os seguintes passos:

- Passo 1: Selecione aleatoriamente uma instância não visitada e teste todos os candidatos restantes contra ela;
- Passo 2: Armazene os resultados em populações de eficiência correspondentes;
- Passo 3: Se nenhuma diferença nas médias de populações correspondentes é detectada pelo ANOVA, continue;
- Passo 4: Conduza comparações múltiplas de média e remova candidatos se eles são significativamente piores que outros a um dado nível de significância;
- Passo 5: Retorne ao Passo 1 até que tenha somente um candidato restante ou todas as instâncias foram visitadas.

Yuan e Gallagher (2004) apresentam a aplicação do *Statistical Racing* como uma ferramenta de propósito geral a ser usada na redução do tempo necessário à realização de experimentos computacionais de larga escala com algoritmos evolucionários.

O *framefork* proposto por Yuan e Gallagher (2004) envolve a estimação da distribuição de probabilidade de alguns indivíduos da população em cada geração e itera com a geração de novos indivíduos a partir da distribuição de probabilidade calculada previamente.

Os passos básicos do algoritmo de corrida são:

- Passo 1: Inicializa uma população P gerando N indivíduos aleatoriamente;
- Passo 2: Avalia todos os indivíduos;
- Passo 3: Escolhe os M melhores indivíduos como *kernels*;
- Passo 4: Cria P' pela amostragem de N indivíduos do estimador de densidade de *kernel*;
- Passo 5: Avalia todos os novos indivíduos em P';

- Passo 6: Combina P e P' para criar uma nova população;
- Passo 7: Vá para o Passo 3 até que a condição de parada seja satisfeita.

O trabalho de iniciação científica de Miranda (2011) envolveu o desenvolvimento de um método de seleção automático de parâmetros de algoritmos evolucionários baseado no algoritmo de corrida de Maron e Moore (1997) e no *framework* de Yuan e Gallagher (2004).

O método desenvolvido alia passos dos algoritmos de corrida (como o teste estatístico via ANOVA) com o *framework* de Yuan e Gallagher (2004). Seu funcionamento básico é descrito a seguir:

- Passo 1: Sorteie np conjuntos de parâmetros, sendo np uma quantia escolhido pelo usuário;
- Passo 2: Faça uma execução rápida da meta-heurística com cada um dos np parâmetros;
- Passo 3: Agrupe os conjuntos de parâmetros usando o ANOVA, com base nos resultados obtidos;
- Passo 4: Escolha um represente para cada grupo, i.e., *kernel*;
- Passo 5: Enquanto houver mais de um grupo de parâmetros, faça:
  - Passo 5.1: Realize uma nova rodada com o *kernel* dos grupos, incrementando o tempo de execução;
  - Passo 5.2: Elimine o grupo do *kernel* com pior resultado;
  - Passo 5.3: Reduza o tamanho dos grupos e conduza um novo agrupamento com o ANOVA;
- Passo 6: Realize uma execução completa com o *kernel* do último grupo ativo.

2.6.3 Considerações sobre o ajuste automático de parâmetros

Os resultados apresentados em Miranda (2011) foram obtidos pela aplicação do método modificado com base no *Statistical Racing* sobre o algoritmo de Recozimento Simulado na minimização da função de teste de Rosenbrock (Equação 5.3).

As variáveis ajustáveis por parte do usuário (mesmo no processo de seleção automática de parâmetros) incluem:

- o número de experimentos que são executados para a geração de uma média do desempenho;
- o número de conjuntos de parâmetros que serão gerados como candidatos;
- os limites onde a geração dos candidatos ocorrerá;
- o número inicial de candidatos que serão agrupados para a análise com o ANOVA (assim como os próprios parâmetros de similaridade do ANOVA);
- e o critério de parada das execuções rápidas iterativas do método.

Dois experimentos principais foram apresentados no relatório final. O primeiro com um número reduzido de conjuntos de parâmetros candidatos (seis ao todo) e um segundo com 100 (cem) conjuntos candidatos[11].

Para ambos os casos, o conjunto de parâmetros *kernel* selecionado automaticamente ao final do processo resolveu a função de testes para uma precisão de até 8.6×10^{-4} (no caso onde um maior conjunto de parâmetros foi avaliado).

É mencionável o fato de que os parâmetros selecionados ficaram dentro de uma faixa equivalente àquela selecionada por experimentação empírica. A vantagem reside na redução do tempo e do esforço necessário para chegar a esta faixa mais promissora, onde a partir daí maiores esforços podem ser concentrados para refinar o resultado.

O número reduzido de avaliações da função objetivo ao longo do processo de seleção contribui positivamente para a adoção futura do método de *Statistical Racing* em substituição à seleção empírica por parte do Cientista Programador.

[11] Em um terceiro caso, não apresentado no relatório final, mas usado para testar a total capacidade do algoritmo, 1.000 conjuntos de parâmetros foram gerados, agrupados e sucessivamente filtrados até a obtenção de um conjunto *kernel* final.

3 PROBLEMA INVERSO FORMULADO COMO UM PROBLEMA DE OTIMIZAÇÃO

"No PD, você resolve o problema.
No PI, o problema resolve você."

(Reversal russa)

Este capítulo apresentada as definições de problema direto, inverso e regularização. Também é proposta uma representação pictórica alternativa para ilustrar a interconexão entre os componentes envolvidos na solução de problemas inversos.

3.1 Problema direto

Um sistema pode ser descrito como uma entidade capaz de produzir algum efeito (ou saída) considerando-se uma causa (ou entrada), tal como o ilustrado na Figura 3.1.

Figura 3.1 - Representação simplificada de um sistema.

Este esquema pode facilmente ser usado para representar um modelo natural ou físico, de maneira simplificada, onde dada uma condição inicial, o sistema resulta uma condição final qualquer.

De acordo com Tarantola (2005), o estudo de um modelo físico segue um procedimento científico que pode ser dividido em três passos básicos:

a) Parametrização do modelo: que envolve a descoberta de um conjunto mínimo de parâmetros do modelo que consiga caracterizar completamente o modelo (de um dado ponto de vista);

b) Modelagem direta: que envolve a descoberta das leis físicas que nos permitam, para determinados parâmetros, fazer predições nos resultados de

medida de alguns parâmetros observáveis;

c) Modelagem inversa: consiste basicamente do uso dos resultados de medidas de propriedades observáveis de um fenômeno para inferir parâmetros do modelo.

Estes passos estão intimamente correlacionados, e ainda nas palavras de Tarantola (2005), "avanços na caracterização de algum dos passos implica em avanços nos outros passos".

Os dois primeiros passos são classificados como indutivos, pois as regras de pensamento e lógica usados em suas caracterizações são implícitas. Já o terceiro passo, a modelagem inversa, é dita dedutiva, pois a teoria de lógica matemática, juntamente com a teoria de probabilidade, pode ser facilmente aplicada a este passo (TARANTOLA, 2005).

Em seu artigo clássico, Sabatier (1985) apresenta a definição de modelo físico, ou modelo direto, como sendo o mapeamento $\mathcal{M}$ de um conjunto $\mathcal{C}$ de parâmetros teóricos em um conjunto $\mathcal{E}$ de resultados. Para Tarantola (2005) a resolução do modelo direto significa predizer os valores dos parâmetros observáveis, de forma livre de erros, que correspondam a um dado modelo. Matematicamente, podemos descrever a resolução do modelo direto como:

$$A\left(u\right) = f, \tag{3.1}$$

onde A é um operador matemático, na maioria das vezes não-linear, que representa o modelo direto; u representa os parâmetros (ou dados) de entrada do modelo; e f a saída, ou parâmetros observáveis do sistema.

Resolver o problema direto equivale a descrever o estado final f do sistema A após a aplicação da condição inicial imposta pelos dados de entrada u. Desta maneira, a solução matemática da Equação 3.1 pode ser facilmente associada à Figura 3.1, onde u se associa à entrada, A se associa ao sistema e f se associa à saída.

Na maioria das vezes f é obtido ou através da simulação de modelo matemático em computador de precisão finita ou obtido por instrumentos de medida, e.g., sensores de temperatura, medidores de velocidade. Para ambos os casos, a precisão, ou a

garantia da obtenção do valor exato, não pode ser considerada, sendo que então deve-se considerar que o valor de f é aproximado, com um determinado nível de ruído inerente.

Desta forma, se considerarmos δ como um valor dependente da precisão do modelo matemático ou do instrumento de medida, deve-se trabalhar com um valor aproximado de f. A diferença de valor entre f e f_δ deve ser condizente com o nível de ruído δ, assim, chegamos à seguinte relação:

$$\|f_\delta - f\| \leq \delta \tag{3.2}$$

Para uma visão mais focada na modelagem, Goldbarg e Luna (2005) apresentam uma esquematização dos procedimentos envolvidos na construção e evolução iterativa de um modelo baseado em um fenômeno, ou sistema, real. A Figura 3.2 apresenta este esquema.

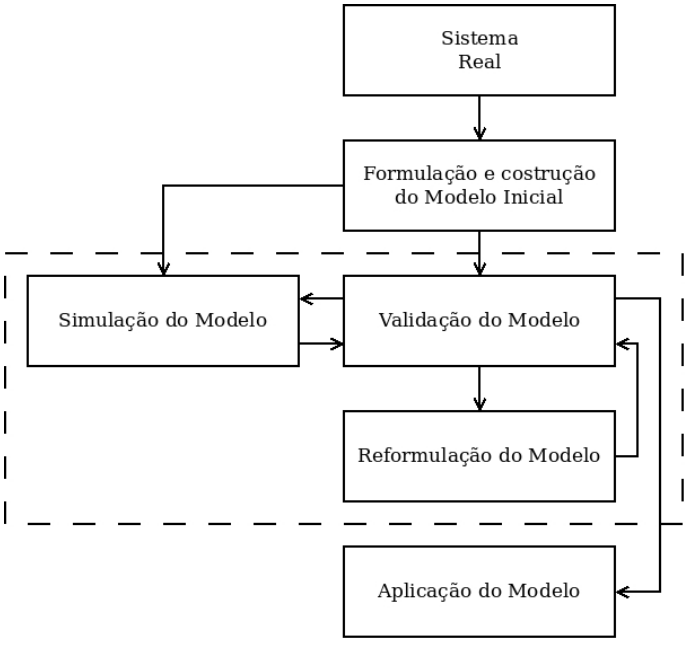


Figura 3.2 - Construção e evolução iterativa de um modelo.
Fonte: Adaptado de Goldbarg e Luna (2005).

Neste esquema, o sistema real em estudo é a base para o desenvolvimento de um

modelo inicial, que então é simulado e sua validade é testada com base na realidade. O objetivo é fazer com que a resposta do modelo formulado se aproxime da realidade. Caso esta validação não seja possível, o modelo é reformulado, com a adição/remoção de variáveis ou formulações. Este ciclo itera até que uma boa representação seja obtida e a aplicação do modelo garanta respostas condizentes com a realidade.

3.2 Problema inverso

Para uma boa definição de problema inverso, Engl et al. (1996) dizem que "resolver um problema inverso é determinar causas desconhecidas a partir de efeitos desejados ou observados". Esta definição caracteriza o fato de usarmos os resultados de observações, que podem ser obtidas também pelo modelo direto, para inferir valores dos parâmetros do sistema que está sendo investigado.

Matematicamente, problemas inversos podem ser escritos como:

$$A^{-1}(f) = u, \tag{3.3}$$

onde A^{-1} é o operador inverso; f representa os parâmetros observáveis do sistema, ou seu estado final; e u representa os parâmetros iniciais, ou de entrada do modelo, que se tenta estimar.

A relação entre problema direto e problema inverso é amplamente difundida pela representação esquemática da Figura 3.3:

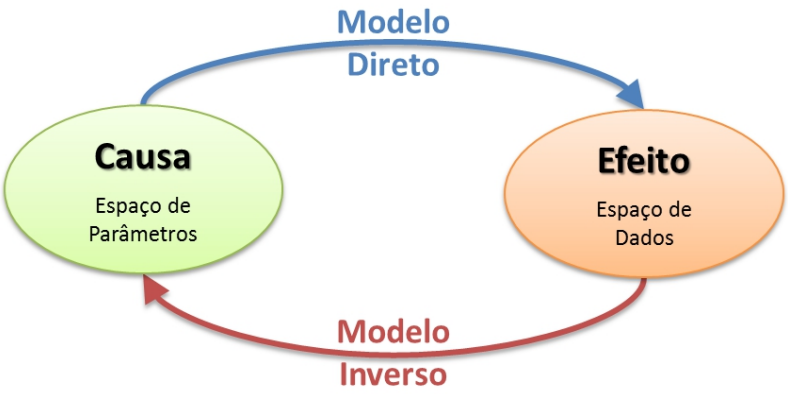


Figura 3.3 - Relação entre problema direto e problema inverso.

A solução do problema inverso envolve a construção do operador inverso A^{-1}, que dependendo de algumas condições, e.g., ser um operador linear, pode ser resolvido de maneira direta (por métodos explícitos), ou iterativamente (com métodos implícitos, como o de mínimos quadrados).

Problemas inversos podem ser classificados de várias maneiras, sendo esta ainda uma área em constante discussão entre diversos autores. Dentre as principais características que podem ser usadas para a classificação de problemas inversos pode-se citar as seguintes (CAMPOS VELHO, 2008):

- A natureza matemática do método: explícito ou implícito;
- A natureza estatística do método: determinística ou estocástica;
- A natureza da propriedade estimada: condição inicial, condição de contorno, termo de fonte/sumidouro ou propriedades do sistema;
- A natureza da solução: estimação de parâmetros ou estimação de funções;
- A natureza da dimensão do problema: finito ou infinito (SILVA NETO; MOURA NETO, 2005).

A dificuldade na solução de problemas inversos reside no fato principal de que esta classe de problemas na grande maioria das vezes viola uma das condições que caracteriza um problema como bem-posto.

A noção de problemas matematicamente bem-postos foi apresentada pelo matemático francês Jacques Hadamard no primeiro capítulo de seu livro "*Lectures on Cauchy's Problem in Linear Partial Differential Equations*" (HADAMARD, 1923) e pode-se sintetiza-las como:

a) Existência: o problema deve possuir solução, i.e., $\forall f \in F \exists u \in U | A(u) = f$;

b) Unicidade: a solução deve ser única, i.e., $\forall f \in F \perp u \in U | A(u) = f$;

c) Estabilidade: a solução (dados de saída) deve depender suavemente dos dados de entrada.

Para a solução implícita de um problema inverso, tal como será adotado para a solução dos diversos problemas inversos que serão apresentados neste trabalho, adotaremos uma formulação matemática que objetiva reduzir o erro quadrático da diferença do resultado do modelo para com resultados observacionais, admitindo-se a existência de ruído (f^{δ}). Desta maneira, a Equação 3.3 passa a ser escrita como (TIKHONOV; ARSENIN, 1977):

$$\min_{u \in U} \| A(u) - f^{\delta} \|^2, \tag{3.4}$$

sendo esta a forma prática de resolver implícita e iterativamente um problema inverso, tal como o adotado nesta tese, em consonância com as técnicas de meta-heurísticas que foram apresentadas no Capítulo 2.

3.3 Regularização

De acordo com Engl et al. (1996), a regularização pode ser definida como a aproximação de um problema mal-posto por uma família de problemas vizinhos bem-postos.

O objetivo da regularização, portanto, é encontrar a melhor solução aproximada $A^{\dagger}(u) = f^{\dagger}$ para o problema original $A(u) = f$. A solução aproximada deve ser suave, i.e., na maioria das vezes tenta-se contornar a violação da terceira condição de Hadamard.

Entende-se que a busca por uma solução mais regular, i.e., suave, envolve a adição de conhecimento *a priori* à solução do problema, de tal forma que um problema mal-posto se torne um problema bem-posto (Figura 3.4).

Este conhecimento *a priori* pode ser incluso na representação matemática do problema inverso como uma restrição. Esta restrição pode vir da realidade física a qual o problema está inserido e em consonância como o proposto por Engl et al. (1996). Esta restrição privilegiaria a família de problemas vizinhos bem-postos, solucionando a violação da terceira condição de Hadamard.

A inclusão da regularização leva à reescrita da Equação 3.4, que agora deve levar em consideração a restrição imposta pela regularização:

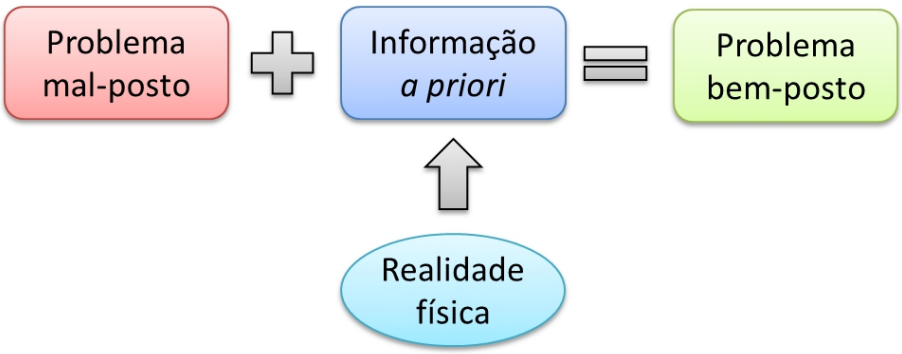


Figura 3.4 - A adição de informação *a priori* possibilita transformar um problema mal-posto em um problema bem-posto.

$$\min_{u \in U} \| A(u) - f^{\delta} \|^2 \text{ sujeito a } \Omega[u] \leq \rho \tag{3.5}$$

em que $A(u) = f^{\delta}$ representa o modelo direto e Ω representa o operador de regularização (TIKHONOV; ARSENIN, 1977).

A literatura cita três técnicas para o tratamento de restrições (YANG, 2010b): abordagem direta; método da penalidade; e a aplicação de multiplicadores de Lagrange. Esta última é a adotada por Tikhonov e Arsenin (1977), em seu clássico *Solutions of ill-posed problems*, e se tornou a principal técnica quando se resolve problemas inversos implicitamente, pois ela permite colocar em uma única função objetivo os termos de fidelidade dos parâmetros com o modelo direto e de regularidade, tal como o apresentado a seguir:

$$\min_{u \in U} \left\{ \| A(u) - f^{\delta} \|_2^2 + \alpha \Omega[u] \right\}, \tag{3.6}$$

onde $\alpha \in \Re^{+}$ é introduzido como o parâmetro de regularização, via a aplicação da técnica de multiplicadores de Lagrange.

3.3.1 Operador de Regularização

A seleção do operador de regularização equivale a informar o tipo de "realidade física" que está sendo procurado, portanto, a sua seleção varia de acordo com a forma da solução que está sendo procurada.

Serão apresentados três operadores de regularização comumente usados: o operador de Tikhonov de ordem-n; o operador de regularização entrópica; e o operador de entropia não-extensiva.

O operador de Tikhonov de ordem-n foi introduzido por Tikhonov e Arsenin na década de 1960 (TIKHONOV; ARSENIN, 1977) e é um dos mais utilizados em diversas classes de problemas inversos. Sua expressão é dada por:

$$\Omega\left(\vec{W}\right) = \sum_{n=0}^{p} \left\| \vec{W}^{(n)} \right\|_2^2, \tag{3.7}$$

em que $\vec{W}^{(n)}$ é a expressão da n-ésima derivada, ou diferença, do vetor $\vec{W}$.

O segundo operador de regularização apresentado aqui é o de regularização entrópica, especificamente o de máxima entropia, que pode ser descrito como:

$$\Omega\left(\vec{W}\right) = \sum_{n=0}^{p} S^{(n)}\left(\vec{W}\right), \tag{3.8}$$

em que $S^{(n)}\left(\vec{W}\right)$ é a entropia de n-ésima ordem de um vetor (ou função), expresso como:

$$S^{(n)}\left(\vec{W}\right) = -\int s\left(\vec{r}\right) \log\left[s\left(\vec{r}\right)\right] d\vec{r} \tag{3.9}$$

e $s\left(\vec{r}\right)$ é:

$$s\left(\vec{r}\right) = \frac{\vec{W}^{(n)}\left(\vec{r}\right)}{\int \vec{W}^{(n)}\left(\vec{r}\right) d\vec{r}} \tag{3.10}$$

onde $\vec{W}^{(n)}$ representa a n-ésima diferença, ou derivada, do vetor $\vec{W}$.

A entropia ($S^{(n)}$) apresenta o seu valor máximo quando $s\left(\vec{r}\right) = x$, com $x \in \Re$ (uma distribuição uniforme). O seu valor mínimo é obtido quando a distribuição de probabilidades de $s\left(\vec{r}\right)$ se assemelha a um delta de Dirac ($\delta\left(\vec{r}\right)$) (ROBERTI, 2005;

CAMPOS VELHO, 2008).

O terceiro operador de regularização, é baseado na forma não-extensiva da entropia, inicialmente proposto por Tsallis (1988). Sua expressão é dada por:

$$S_q(p) = \frac{k}{q-1}\left[1 - \sum_{i=1}^{N_p} p_i^q\right], \tag{3.11}$$

em que q é um parâmetro livre e é denominado como parâmetro de não-extensividade; o parâmetro k, assim como na termodinâmica, é conhecido como a constante de Boltzmann.

Campos Velho et al. (2006) se vale dos princípios apresentados pela regularização entrópica não-extensiva para propor uma teoria de regularização unificada, onde é provado que:

a) Para o valor de $q = 1$, tomando o limite da Equação 3.11, o operador de regularização converge para o operador de entropia extensiva;

b) Para o valor de $q = 2$, tomando o limite da Equação 3.11, o operador de regularização converge para o operador de Tikhonov.

Desta maneira, a teoria unificada propõe que os operadores de entropia extensiva e de Tikhonov se tornam casos particulares do operador de entropia não-extensiva.

3.3.2 Parâmetro de Regularização

A definição de um valor para o parâmetro de regularização é um dos maiores desafios quando se trabalha com problemas inversos com regularização. Deve-se levar em consideração duas condições que surgem nos extremos da possibilidade de seleção deste valor:

- $\alpha \to 0$: pouca regularização, logo o resultado final é pouco afetado. Quando $\alpha = 0$ não há inclusão da regularização no resultado da otimização do funcional, o que pode levar a um resultado com amplificação do ruído existente nos dados experimentais;

- $\alpha \to \infty$: excesso de regularização. Neste caso, o resultado perde informação da realidade física, levando a um resultado estritamente centrado em uma solução regular.

Comumente, a experimentação empírica é o método mais utilizado para a definição do valor de α. Este trabalho se valeu desta experimentação quando a regularização foi imposta à solução. Porém, existem técnicas que podem ser usadas para a determinação objetiva de α, tal como a curva-L e a discrepância de Morozov, aqui apresentadas.

Definir automaticamente o valor de α é o objetivo do método da curva-L, um critério geométrico apresentado por Hansen (1992), onde o valor de α é obtido pela análise da relação entre a diferença quadrática da fidelidade do resultado e a regularização. Este método é geométrico por se basear no resultado obtido pelo ponto indicado na máxima curvatura da curva construída para o gráfico $\Omega\left[u_\alpha\right] \times \left\|A\left(u_\alpha\right) - f^\delta\right\|_2^2$, que geralmente apresenta a forma de uma curva-L.

Outra técnica de estimação do valor de α é o critério da discrepância de Morozov, que se baseia no fato de que a diferença entre os dados do modelo e os dados observados deve ter a mesma magnitude do erro de medida. Logo, se δ representa o erro de medida, α^*, i.e., o valor ótimo, é a raiz da equação que segue:

$$\left\{\|A\left(u\right) - f^\delta\|^2\right\}_{a^*} = \delta \tag{3.12}$$

Na utilização do princípio da discrepância de Morozov, a Equação 3.12 é aplicada como:

$$\|A\left(u\right) - f^\delta\|^2 \approx N\sigma^2, \tag{3.13}$$

em que N é o número de parâmetros a serem estimados e σ^2 é a variância da distribuição gaussiana dos erros de medida (f^δ).

3.4 Representação de problemas inversos

O conceito representado pela Figura 3.3 é baseado na definição fornecida por Engl et al. (1996). Este conceito é amplamente divulgado e seu uso é constante nos mais

diversos textos relacionados a problemas inversos.

Em seu artigo "*The Anatomy of Inverse Problems*", Scales e Snieder (2000) apresentam uma proposta de ampliação da representação dada pela Figura 3.3. Sua base é a de problemas aplicados à geofísica, e sua proposta é apresentada pela Figura 3.5.

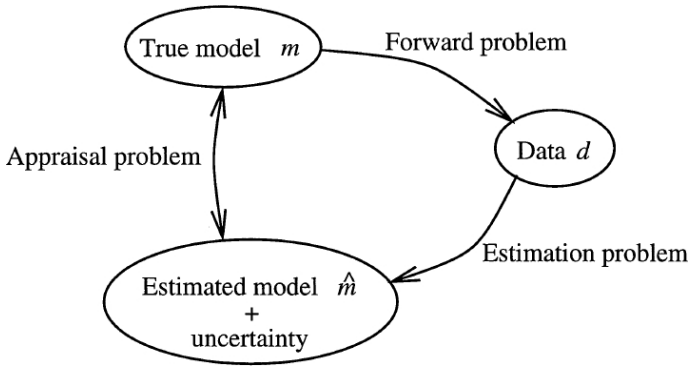


Figura 3.5 - Visão expandida de problemas inversos com foco em problemas geofísicos. Fonte: (SCALES; SNIEDER, 2000).

Esta representação introduz o conceito da existência de um "*true model*", um modelo fiel e definitivo do fenômeno físico em estudo. Este "*true model*" tem como representante um modelo estimado atualmente em uso e é parte do problema inverso obter um novo modelo estimado que melhor se aproxime da representação definitiva.

O conceito de "*true model*" pode levar a uma série de interpretações equivocadas sobre a natureza do problema, especialmente com relação à representação e implementação deste modelo em máquinas limitadas, i.e., computadores truncam e arredondam números reais.

Desta forma, este trabalho propõe uma adaptação do trabalho de expansão apresentado por Scales e Snieder (2000). Esta adaptação envolve a superposição dos conceitos apresentados pela metodologia de construção iterativa de modelos (Figura 3.2), pela representação da definição de Engl et al. (1996) (Figura 3.3) e pela adição de conhecimento *a priori* para a mitigação do mal-condicionamento do problema inverso (Figura 3.4).

Aqui, o modelo direto e inverso em uso são baseados no sistema ou fenômeno real

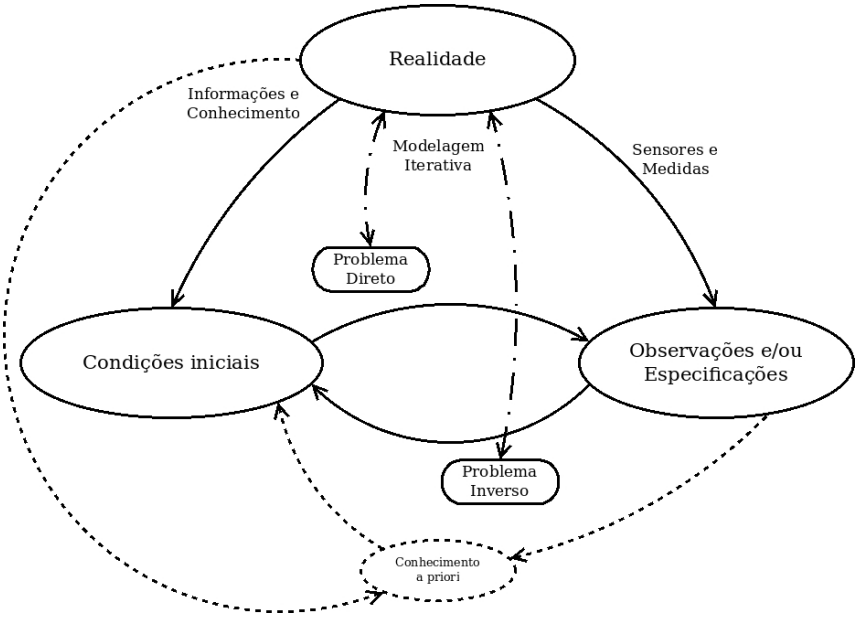


Figura 3.6 - Proposta de uma nova visão expandida para a representação de problemas inversos.

objetivo do estudo e por um processo iterativo tem seus resultados melhorados conforme novas técnicas ou ferramentas se tornem disponíveis para aumentar a precisão desta representação (este processo iterativo é representado pelas linhas com traço e ponto). As condições iniciais, observações, ou especificações desejadas, assim como o conhecimento *a priori*, também têm por base a realidade, e.g., a obtenção de valores observáveis via sensores. A linha pontilhada representa o caminho alternativo a ser seguido pela resolução de problemas inversos regularizados, i.e., os dados gerados pelo modelo, mais a informação *a priori*, devem se aproximar mais dos valores observados, e consequentemente reduzir o valor calculado pela Equação 3.6.

A visão aqui apresentada se aplica a uma classe de métodos usados na solução de problemas inversos, especificamente àquelas que se valem da minimização de resíduos quadráticos via métodos de otimização iterativos. Outras classes, tais como a solução de problemas inversos via redes neurais, podem levar à construção de uma representação própria, eliminando a necessidade de um modelo direto, por exemplo.

Esta nova proposta não tem por objetivo substituir a Figura 3.3, mas vem para expandir e complementar a interpretação das estratégias para se construir soluções de problemas diretos e inversos.

4 EXEMPLOS DE APLICAÇÕES

"A man who carries a cat by the tail learns something he can learn in no other way."

(Mark Twain)

Este capítulo apresenta a formulação matemática direta e inversa dos problemas que serão usados como exemplos de aplicações às meta-heurísticas apresentadas no Capítulo 2. Os resultados numéricos são apresentados no Capítulo 5.

4.1 Identificação de condição inicial em condução do calor

O primeiro problema direto e inverso a ser usado como exemplo neste trabalho é o de identificação de condição inicial em condução do calor, em sua forma unidimensional, para uma barra em condições adiabáticas (Figura 4.1). Especificamente, deseja-se obter a distribuição inicial de temperatura em uma barra de tamanho unitário com base em medidas das temperaturas transientes transcorrido um tempo τ do pulso inicial de calor, em pontos igualmente espaçados.

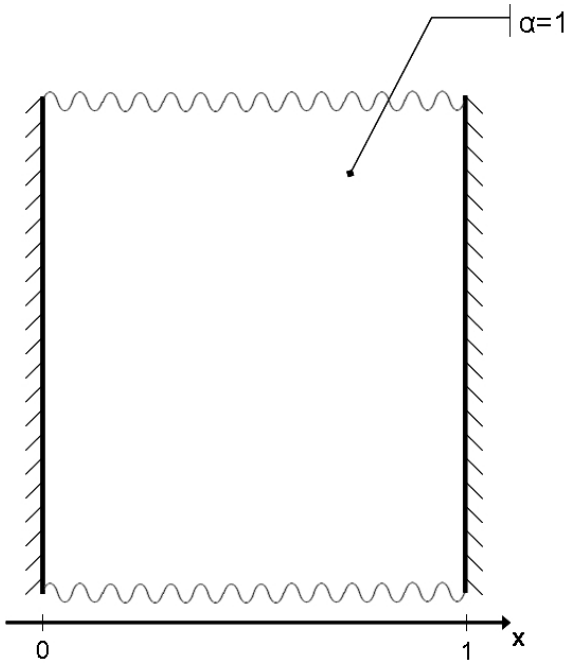


Figura 4.1 - Representação da barra unidimensional com condições adiabáticas utilizada neste problema.

4.1.1 Modelo direto

O problema direto para a determinação da condição inicial da equação do calor consiste-se de um problema de condução de calor transiente em uma barra com alto coeficiente de difusividade, condições de fronteira adiabáticas e temperatura inicial dada por $f(x)$. A formulação matemática para a solução do problema direto, dada a distribuição inicial de temperatura, se apresenta na forma (ÖZISIK; ORLANDE, 2000; MUNIZ, 1999):

$$\begin{aligned} \frac{\partial^2 T(x,t)}{\partial x^2} &= \frac{\partial T(x,t)}{\partial t}, && 0 < x < 1, t > 0, \\ \frac{\partial T(x,t)}{\partial x} &= 0, && x = 0 \text{ e } x = 1, t = 0, \\ T(x,0) &= f(x), && 0 \leq x \leq 1, t = 0, \end{aligned} \tag{4.1}$$

em que a temperatura $T(x,t)$, a condição inicial $f(x)$, a variável espacial x e a variável temporal t, são quantidades adimensionais (MUNIZ, 1999), todas no domínio $[0;1]$.

De acordo com Özisik e Orlande (2000), a solução do problema direto para a determinação de $T(x,t)$, para $0 \leq x \leq 1$ e $t > 0$, dada uma condição inicial de temperatura expressa por $f(x)$, com $0 \leq x \leq 1$, pode ser obtida utilizando-se o método espectral e superposição linear. Levando à seguinte solução exata:

$$T(x,t) = \sum_{m=0}^{+\infty} e^{-\beta_m^2 t} \frac{1}{N(\beta_m)} X(\beta_m, x) \int_0^1 X(\beta_m, x') f(x') \, dx', \tag{4.2}$$

sendo:

$$\beta_m = m\pi$$

$$X(\beta_m, x) = \cos(\beta_m x) \text{ para } m = 0, 1, 2, \ldots$$

$$N(\beta_0, x) = 1 \text{ e } N(\beta_m, x) = \frac{1}{2} \text{ para } m = 1, 2, \ldots$$

4.1.2 Modelo Inverso

O problema inverso para este caso consiste-se da determinação da distribuição inicial de temperatura ($f(x)$) dada uma medida de temperatura observada (T^{Obs}) para um tempo $t = \tau > 0$. Este tipo de problema possui uma dificuldade intrínseca dado seu mal-condicionamento, i.e., é um problema mal-posto que viola a terceira condição de Hadamard (veja a Seção 3.2).

Sua formulação se baseia na apresentada por Tikhonov e Arsenin (1977) (veja Seção 3.2), que prevê a redução do erro quadrático da diferença entre valores experimentais ou observados (T^{Obs}) em um tempo $\tau > 0$ e valores obtidos pelo modelo matemático, dada uma condição inicial $f(x)$ para a temperatura ($A(f(x))$):

$$F\left(\alpha, f(x)\right) = \left[T^{Obs} - A\left(f(x)\right)\right]^2 + \alpha\Omega\left(\vec{f(x)}\right), \tag{4.3}$$

em que α representa o valor do parâmetro de regularização (veja Subseção 3.3.2) e Ω representa o operador de regulação em uso (veja Subseção 3.3.1). A matriz A é a matriz de transição de estados com expressão obtida pelo uso, em uma mala regular, do método integral (quadratura numérica), que se baseia em uma aproximação linear de $f(x)$ nos sub-intervalos de integração e que também considera a regra do trapézio (MUNIZ, 1999).

A expressão da matriz A é dada por:

$$A_{ij} = e^{-m^2\pi^2\tau} \sum_{m=0}^{n} \frac{\cos\left(m\pi x_j\right)}{N_m} \frac{1}{2} \cos\left(m\pi x_i\right) \Delta x, \tag{4.4}$$

em que $N_0 = 1$ e $N_m = 0,5$ para $m = 1, 2, \ldots$; $x_i = (i-1)\Delta x$; e $x_j = (j-1)\Delta x$. A construção da matriz de transição de estados A ainda segue o seguinte esquema:

$$A = \begin{bmatrix} A_{1,1} & 2A_{1,2} & \cdots & 2A_{1,n} & A_{1,n+1} \\ A_{1,2} & 2A_{2,2} & \cdots & 2A_{2,n} & A_{2,n+1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ A_{n+1,1} & 2A_{n+1,2} & \cdots & 2A_{n+1,n} & A_{n+1,n+1} \end{bmatrix} \tag{4.5}$$

4.2 Termo de fonte/sumidouro em poluição atmosférica

O problema direto e inverso relacionados ao transporte e difusão de um gás na atmosfera (Figura 4.2) foi escolhido por representar um dos principais tópicos de pesquisa relacionado à nova área de ciência do sistema terrestre. O grupo de pesquisa em problemas inversos do INPE possui ampla experiência neste tipo de problema(LUZ et al., 2007).

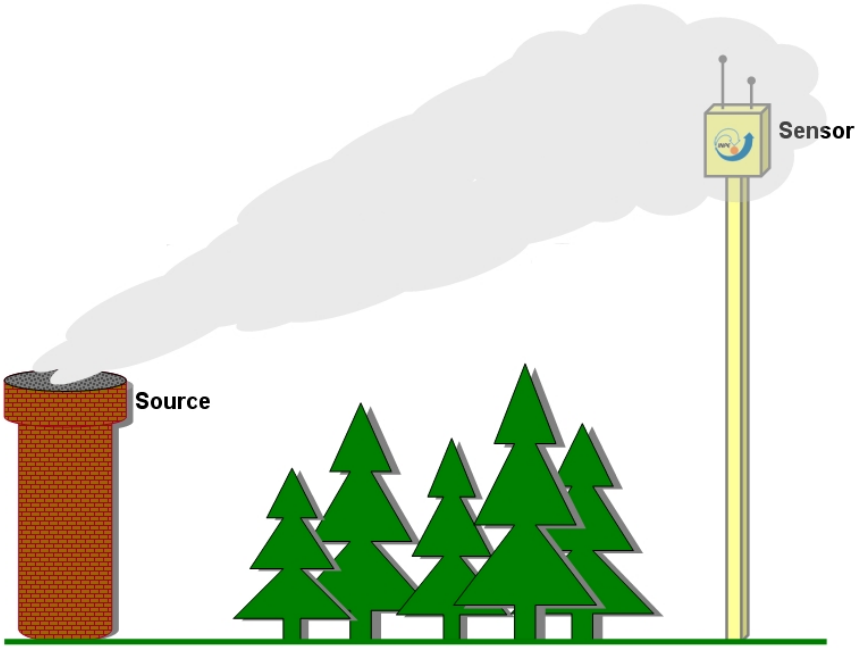


Figura 4.2 - A dispersão de poluentes atmosféricos é uma área de estudos de interesse de diversos grupos que estudam a ciência do sistema terrestre.
Fonte: Luz et al. (2007)

4.2.1 Modelo direto

O problema direto que será abordado leva em consideração o modelo de transporte e difusão de partículas na atmosfera. Este modelo de transporte pode ser representado através de uma abordagem euleriana ou lagrangiana, que são utilizadas atualmente para modelar o transporte turbulento em fluidos.

O modelo matemático para o escoamento de um fluido pode ser descrito através das equações de conservação de momento (Navier-Stokes), massa (continuidade), calor

(primeira lei da termodinâmica) e conservação de uma quantidade escalar genérica, além da equação de estado (lei do gás ideal).

A modelagem euleriana considera o comportamento das variáveis turbulentas com relação a um sistema de coordenadas fixas, sendo possível a resolução numérica do modelo direto de transporte de partículas através da Equação 4.6.

$$\frac{\partial C\left(\vec{x},t\right)}{\partial t}+u_i\frac{\partial C\left(\vec{x},t\right)}{\partial x_i}=v_c\frac{\partial^2 C\left(\vec{x},t\right)}{\partial x_i^2}+S, \tag{4.6}$$

em que $C\left(\vec{x},t\right)$ representa a concentração de partículas na posição $\vec{x}$ no instante t, u_i representa a velocidade do vento que pode ser decomposta nos três eixos (u_x, u_y, u_z, v_c) representa a difusividade molecular das partículas, S representa o termo de fonte ou sumidouro das partículas. Esta modelagem não é o foco deste trabalho.

Na modelagem lagrangiana, a posição espacial de uma partícula (x, y, z) em um determinado tempo (t), é determinada através da consideração de sua posição relativa com relação ao tempo inicial, i.e., (x_0, y_0, z_0) em t_0. Esta consideração usa uma abordagem estatística para determinar a função densidade probabilidade (FDP) associada à distribuição espacial das parcelas de fluidos que compõem o escoamento em um ambiente turbulento. Desta maneira a evolução espaço-temporal da concentração média de um conjunto de partículas em uma dada parcela de fluido é descrita como,

$$C\left(\vec{x},t\right)=\int_{-\infty}^{t}\int_{-\infty}^{\infty}S\left(\vec{x_0},t_0\right)P\left(\vec{x},t|\vec{x_0},t_0\right)d\vec{x_0}dt_0, \tag{4.7}$$

em que $P\left(\vec{x},t|\vec{x_0},t_0\right)$ é a FPD da parcela de fluido que está na posição $\vec{x_0}$ no instante t_0 passar a ocupar a posição $\vec{x}$ no tempo t, e $S\left(\vec{x_0},t_0\right)$ representa a função que descreve a distribuição espaço-temporal da fonte ou sumidouro em termos da massa das partículas, por unidade de volume, entre as posições $\vec{x_0}$ e $\vec{x}$ nos instantes t_0 e t, respectivamente.

O principal termo da Equação 4.7 é $P\left(\vec{x},t|\vec{x_0},t_0\right)$, e o cálculo deste termo pode ser feito através da determinação da trajetória de um conjunto de partículas contidas no escoamento do fluido. Este conjunto de partículas deve ser suficientemente grande

para garantir a determinação de uma trajetória relativamente precisa.

Uma boa maneira de estimar estas trajetórias é valer-se de um modelo lagrangiano de partículas baseado na equação de Langevin, onde o movimento das partículas num dado fluido de escoamento turbulento pode ser descrito de forma análoga ao movimento Browniano. A adoção da teoria relativa ao movimento Browniano permite tratar a aceleração sofrida pela partícula como sendo a soma das acelerações determinísticas e estocásticas (LUZ, 2008; ROBERTI, 2005).

Para o estabelecimento das relações entre a fonte ou sumidouro de uma dada partícula e sua concentração em dado ponto espacial, o modelo lagrangiano permite o uso de dois diferentes modelos matemáticos para a expressão da integral temporal, são eles:

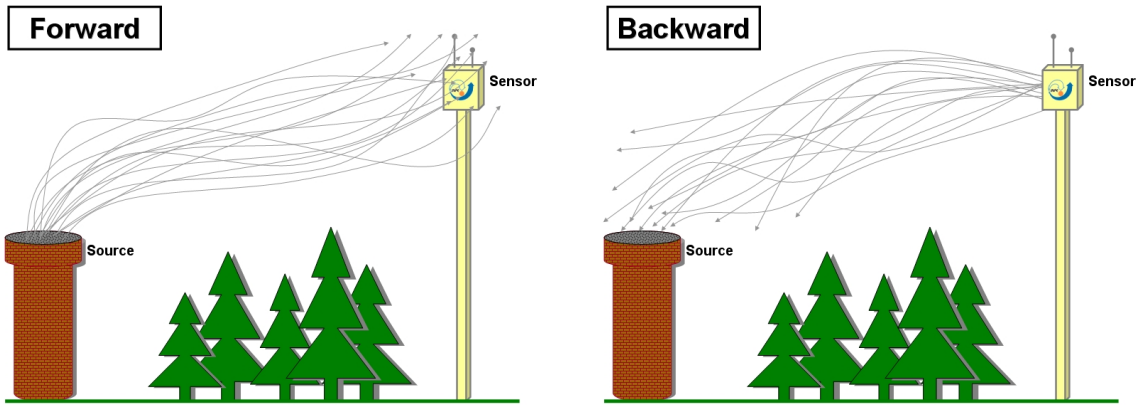
- Modelo de integração avançado (*forward*);
- Modelo de integração regressivo (*backward*).

O modelo avançado (*forward*) tem por base o preceito de que as partículas simuladas são emitidas a partir do volume relativo à sua fonte ou sumidouro e a integração é feita do tempo t_0 até o tempo t. De maneira contrária, o modelo regressivo (*backward*) simula a emissão das partículas a partir do volume do sensor em direção ao volume da fonte ou sumidouro e as trajetórias são traçadas com a integração do tempo t ao tempo t_0. O esquema é ilustrado pela Figura 4.3.

O fator que deve ser levado em consideração para a adoção de um modelo de integração avançado ou regressivo é principalmente o custo computacional associado ao cálculo destes modelos, com forte influência da quantidade de fontes, sumidouros e sensores usados.

4.2.1.1 Modelo direto de integração avançada

O modelo lagrangiano de integração avançada define que a concentração média de um contaminante na posição $\vec{x}$ e no tempo t, com uma taxa de emissão (ou absorção) na fonte (ou sumidouro) dada por $S\,(kgm^{-3}s^{-1})$, pode ser expressa por:



(a) (b)

Figura 4.3 - Conceituação gráfica para: (a) modelo de integração avançada (*forward*) onde as partículas são emitidas da fonte para o sensor, i.e., $t_0 \rightarrow t$; (b) modelo de integração regressiva (*backward*) onde as partículas são emitidas do sensor para a fonte, i.e., $t \rightarrow t_0$.

$$C\left(x,t\right) = \int_{-\infty}^{t} \int_{-\infty}^{\infty} S\left(\vec{x_0},t_0\right) P^a\left(\vec{x},t|\vec{x_0},t_0\right) d\vec{x_0} dt_0, \tag{4.8}$$

em que $P^a\left(\vec{x},t|\vec{x_0},t_0\right)$ é a densidade de probabilidade de transição avançada, definida para que $P^a\left(\vec{x},t|\vec{x_0},t_0 d\vec{x_0}\right)$ seja a probabilidade de que um elemento de fluido inicialmente posicionado em $(\vec{x_0},t_0)$ seja encontrado no volume $d\vec{x}$, centrado em $\vec{x}$ no tempo t. O cálculo de $P^a\left(\vec{x},t|\vec{x_0},t_0\right)$ é feito pelo modelo lagrangiano avançado. E ainda considerando que a taxa de emissão ou absorção, S, seja uniforme sobre o volume da fonte ou sumidouro, temos que,

$$S\left(\vec{x_0},t_0\right) = SW\left(\vec{x_0}\right), \tag{4.9}$$

em que $W\left(\vec{x_0}\right)$ é uma função de localização adimensional (0 ou 1), ou função característica, que quando fora da área de emissão ou absorção é definida como 0 (zero). Com a adição da turbulência e na condição de uma fonte estacionária, a concentração média das partículas é independente do fator tempo e P^a depende somente de $t - t_0$, desta maneira,

$$C\left(\vec{x},t\right)=S\int\limits_{t=0}^{\infty}\int\limits_{\vec{x_0}=-\infty}^{\infty}W\left(\vec{x_0}\right)P^a\left(\vec{x},t|\vec{x_0},0\right)d\vec{x_0}dt. \tag{4.10}$$

Assim, para o modelo de integração avançado, o cálculo prático da concentração média das partículas é feito sobre um sensor de volume V_S,

$$C^v\left(\vec{x}\right)=\frac{S}{V_S}\int\limits_{V_S}\int\limits_{t=0}^{\infty}\int\limits_{\vec{x_0}=-\infty}^{\infty}W\left(\vec{x_0}\right)P^a\left(\vec{x},t|\vec{x_0},0\right)d\vec{x_0}dtd\vec{x}. \tag{4.11}$$

A determinação de $C^v\left(\vec{x}\right)$ é dada pelo cálculo do tempo de residência média das partículas no volume do sensor V_S,

$$\bar{T^f}\left(\vec{x},V_S|\vec{x_0}\right)=\int\limits_{V_S}\int\limits_{t=0}^{\infty}P^a\left(\vec{x'},t|\vec{x_0},0\right)dtd\vec{x'}, \tag{4.12}$$

logo,

$$C^v\left(\vec{x}\right)=\frac{S}{V_S}\int\limits_{\vec{x_0}=-\infty}^{\infty}W\left(\vec{x_0}\right)\bar{T^a}\left(\vec{x},V_S|\vec{x_0}\right)d\vec{x_0}, \tag{4.13}$$

e ainda considerando que a fonte de emissão das partículas possui um volume V_f, a Equação 4.13 se transforma em:

$$C^v\left(\vec{x}\right)=\frac{S}{V_S}\int\limits_{V_f}\bar{T^a}\left(\vec{x},V_S|\vec{x_0}\right)d\vec{x_0}. \tag{4.14}$$

Assim, para a implementação, $C^v\left(\vec{x}\right)$ é calculado através de uma média dos tempos de residência de partículas individuais no volume do sensor, expresso por:

$$C^v\left(\vec{x}\right)=S\frac{V_f}{V_S}\frac{1}{N_{EVF}}\sum_{i=1}^{N_{PVS}}\bar{T_i^a}\left(\vec{x},V_S|V_f\right), \tag{4.15}$$

em que N_{EVF} exprime o número de partículas emitidas a cada passo de tempo a partir do volume da fonte e N_{PVS} representa o número de partículas detectadas no volume do sensor.

4.2.1.2 Modelo direto de integração regressiva

Para o modelo de integração regressivo, admite-se que as partículas são emitidas a partir de $(\vec{x}, t)$ e sua posição final é contabilizada em $(\vec{x_0}, t_0)$, levando a uma estimativa da probabilidade condicional do estado regressivo no tempo (ROBERTI, 2005).

Para facilitar a implementação, a adoção da densidade de probabilidade condicional regressiva, $P^r\left(\vec{x_0}, t_0|\vec{x}, t\right)$ pode ser definida como sendo a mesma da integração avançada:

$$P^r\left(\vec{x_0}, t_0|\vec{x}, t\right) = P^a\left(\vec{x}, t|\vec{x_0}, t_0\right), \tag{4.16}$$

desde que condições de fluxo incompressíveis e uma atmosfera com características estacionárias possa ser assumido (ROBERTI, 2005).

No modelo de integração avançado, P^a é obtido pela média sobre o *ensemble* do tempo de residência das partículas no volume de um sensor V_S, e de partículas emitidas pelo volume de uma fonte V_f.

Para o modelo de integração regressivo, o tempo correspondente à residência da partícula pode ser obtido através da contabilização do tempo gasto da partícula no volume da fonte V_f (fonte esta centrada em $\vec{x_0}$) e por partículas emitidas pelo volume do sensor V_S.

Desta forma, a Equação 4.12, que calcula o tempo de residência da partícula para o modelo avançado, pode ser reescrito para o modelo regressivo como:

$$\bar{T^r}\left(\vec{x_0}, V_f|\vec{x}\right) = \int\limits_{V_f} \int_{t'=0}^{\infty} P^r\left(\vec{x'}, t'|\vec{x}, 0\right) dt' d\vec{x'}. \tag{4.17}$$

E quando o tempo de residência do modelo regressivo é substituído na Equação 4.13,

obtem-se:

$$C^v\left(\vec{x}\right) = \frac{S}{V_S}\int\limits_{V_S} \bar{T}^r\left(\vec{x_0}, V_S|\vec{x}\right) d\vec{x}, \tag{4.18}$$

sendo que para a implementação adotamos:

$$C^v\left(\vec{x}\right) = S\frac{1}{N_{EVS}}\sum_{i=1}^{N_{PVF}} \bar{T}_i^r\left(\vec{x_0}, V_f|\vec{x}\right), \tag{4.19}$$

em que N_{EVS} representa o número de partículas emitidas a partir do volume do sensor e N_{PVF} representa o número de partículas que tiveram tempo de residência suficiente no volume da fonte ou sumidouro.

4.2.1.3 Implementação

O modelo de dispersão de partículas lagrangiano é implementado pelo *LAgrangean Model for Buoyant Dispersion in Atmosphere* (LAMBDA) que foi desenvolvido pelo grupo de física da atmosfera do *Istituto di Scienze dell'Atmosfera e del Clima* (ISAC) e usa um modelo fonte-receptor para implementar as estratégias de dispersão avançada (*forward*) e regressiva (*backward*).

Na implementação avançada no tempo, a concentração de um dado contaminante detectado no volume de um sensor é obtida pela análise do tempo de residência da partícula na posição $\vec{x}$. Para que essa partícula seja devidamente atribuída à posição em questão, o seu tempo de residência deve exceder um limite previamente estabelecido, usualmente $\Delta t = 1s$. Assim, partindo da Equação 4.15 temos:

$$C^v\left(\vec{x}\right) = C\left(\vec{x}\right) = S\frac{V_f}{V_S}\frac{1}{N_{PEF}}\sum_{i=1}^{N_{PVS}} \Delta t = S\frac{V_f}{V_S}\frac{\Delta t}{N_{PEF}}N_{PVS} \tag{4.20}$$

em que N_{PEF} é o número de partículas emitidas pela fonte em cada passo de tempo (Δt), N_{PVS} é o número de partículas captadas no sensor, V_S é o volume do sensor e V_f é o volume da fonte.

Estendendo o modelo para atuar com diversas fontes (N_f) e diversos sensores (N_S),

podemos representar a adição dos valores para a concentração calculada na posição j como:

$$C_j = \sum_{i=1}^{N_f} S_i \frac{V_{f,i}}{V_{s,j}} \frac{\Delta t}{N_{PEF,i}} N_{PVS,i,j} \tag{4.21}$$

em que S_i é a intensidade da i-ésima fonte, $V_{f,i}$ é o volume da i-ésima fonte, $V_{s,j}$ é o volume do j-ésimo sensor, $V_{PEF,i}$ é o número de partículas emitidas da i-ésima fonte e $N_{PVS,i,j}$ é o número de partículas emitidas da i-ésima fonte verificada no volume do j-ésimo sensor.

Para a implementação regressiva no tempo, a Equação 4.20 é modificada nos termos da Subsubseção 4.2.1.2 o que nos leva a seguinte formulação:

$$C^v(\vec{x}) = C(\vec{x}) = S \frac{1}{N_{PES}} \sum_{i=1}^{N_{PVF}} \Delta t = S \frac{\Delta t}{N_{PES}} N_{PVF} \tag{4.22}$$

em que N_{PES} representa o número de partículas emitidas do volume do sensor e N_{PVF} representa o número de partículas verificadas no volume da fonte.

A Equação 4.21 é então reescrita para comportar a regressão da partícula no tempo:

$$C_j = \sum_{i=1}^{N_f} S_i \frac{\Delta t}{N_{PES,j}} N_{PVF,i,j} \tag{4.23}$$

em que C_j representa a concentração medida no j-ésimo sensor, S_i a intensidade de emissão na i-ésima fonte, $N_{PES,j}$ representa o número de partículas emitidas pelo j-ésimo sensor e $N_{PVF,i,j}$ representa o número de partículas emitidas pelo j-ésimo sensor e que foram verificadas no volume da i-ésima fonte.

Para reduzir o esforço computacional necessário à solução das equações apresentadas anteriormente, o uso de um modelo fonte-receptor de solução iterativa do modelo direto é aconselhado.

Em um modelo de N_f fontes de emissão com intensidade desconhecida, a concentração medida em N_S sensores é dada por:

$$\vec{C} = M\vec{S} \tag{4.24}$$

em que $\vec{C}$ é o vetor das concentrações medidas nos N_S sensores, i.e., $\vec{C} = (C_1, C_2, \ldots, C_{N_S})$, e o vetor $\vec{S}$ representa as intensidades de emissões das N_f fontes, i.e., $\vec{S} = \left(S_1, S_2, \ldots, S_{N_f}\right)$.

A matriz M é a matriz de transição de estados denominada matriz fonte-receptor e o seu cálculo é função do modelo de dispersão atmosférico LAMBDA.

Os resultados de Roberti (2005) e de Luz (2008) foram obtidos através do uso deste programa e seu uso é validado para uma grande diversidade de cenários e de situações que variam do real ao hipotético.

4.2.2 Modelo inverso

O modelo inverso a ser resolvido para este problema se baseia na formulação apresentada por Tikhonov e Arsenin (1977) (veja Seção 3.2), que prevê a redução do erro quadrático da diferença entre valores experimentais ou observados (C^{Exp}) e valores obtidos pelo modelo matemático, dada uma condição inicial $\vec{S}$ para as emissões ($C^{Mod}(\vec{S})$):

$$F(\alpha, S) = \sum_{j=1}^{m} \left[C_j^{Exp} - C^{Mod}\left(\vec{S}\right)\right]^2 + \alpha\Omega\left(\vec{S}\right), \tag{4.25}$$

em que α representa o valor do parâmetro de regularização (veja Subseção 3.3.2) e Ω representa o operador de regulação em uso (veja Subseção 3.3.1).

Os resultados da aplicação do modelo inverso de termo de fonte/sumidouro de um gás na atmosfera estão na Subseção 5.1.2. Este problema inverso foi resolvido com o *Multiple Particle Collision Algorithm* (MPCA), descrito na Subseção 2.4.1.

4.3 Reconstrução de mapas de precipitação

Dentre as diversas saídas fornecidas pelos modelos de previsão de tempo e clima, a precipitação é uma das variáveis de maior interesse para a sociedade. Informações sobre a variabilidade espacial e temporal da distribuição da precipitação é fator essencial para diversos setores da sociedade, especialmente o econômico e recentemente

o de defesa civil.

A precipitação é o resultado de um complexo conjunto de transferências termodinâmicas de energia e massa (umidade) entre a superfície e a atmosfera. Tal processo se desenvolve em escalas de tempo e espaço relativamente pequenos.

Dentre os diversos componentes de um modelo de previsão de tempo e clima, a representação das nuvens e a realização de previsões de precipitação confiáveis ainda é um dos maiores desafios perante os pesquisadores.

Grell e Dévényi (2002) apresentam um esquema de parametrização (aqui denominado GD) de convecção baseado em um formalismo de fluxo de massa, via conjunto multi-modelos (*ensemble*) de hipóteses e fechamentos.

O conjunto de hipóteses de fechamento do esquema proposto por GD implementado no modelo de previsão regional brasileiro (*Brazilian developments on the Regional Atmospheric Modelling System*, BRAMS) é composto pelos esquemas de fechamento de Grell (GR), Arakawa e Schubert (AS), Kain e Fritsch (KF), Low-level Omega (LO) e convergência de umidade (MC) (FREITAS et al., 2009).

O esquema de média por *ensemble* coleta em cada ponto de grade do modelo os termos forçantes de radiação, fluxos de superfície e difusão turbulenta na Camada Limite Planetária (CLP) que em conjunto com a advecção na escala da grade são usados para estimar o forçante total do ponto de grade em específico. Cada um dos membros das diferentes hipóteses de fechamento utiliza sua própria função gatilho para determinar se uma coluna suportará ou não a convecção e, em caso positivo, o fluxo de massa e a precipitação resultante são calculados considerando a eficiência da precipitação da hipótese de fechamento definido para cada membro do *ensemble* (SANTOS et al., 2010a; SANTOS et al., 2010b).

O resultado obtido pelo esquema de parametrização GD é um conjunto de prognósticos de precipitação que pode ser combinado de diversas formas, gerando uma representação numérica da precipitação calculada e das taxas de aquecimento e umedecimento atmosférico (FREITAS et al., 2009; SANTOS et al., 2010a; SANTOS et al., 2010b).

4.3.1 Modelo direto

A obtenção de um campo de precipitação via modelo direto envolve a execução do modelo BRAMS, versão 4.3, para a simulação de um dia de ocorrência de intensa precipitação sobre a América do Sul, associado a um caso de aparecimento da Zona de Convergência do Atlântico Sul (ZCAS), com observação durante os dias 21 a 24 de fevereiro de 2004. O modelo foi executado por 48 horas simuladas, a partir do dia 19 de fevereiro de 2004, 12h00 UTC, com reinicialização a cada 24 horas até o término do período desejado (SANTOS et al., 2010b; SANTOS et al., 2010a).

O modelo foi configurado para uma resolução horizontal de 25 km e vertical de 100 metros no primeiro nível vertical e 38 níveis totais. A umidade do solo teve inicialização heterogênea. O esquema de parametrização convectiva adotado é o proposto por Grell e Devényi, com os fechamentos propostos de GR, LO, MC, KF e AS. O fechamento do tipo *ensamble*[1] (ENS) também foi utilizado.

As análises do Modelo de Circulação Geral da Atmosfera do Centro de Previsão de Tempo e Estudos Climáticos (MCGA/CPTEC) foram usadas como condições iniciais e de contorno para o modelo BRAMS. A resolução adotada foi a T126L28, o que equivale a um truncamento romboidal na onda de número 126 e 28 níveis do modelo. O modelo foi executado com a variação independente das cinco hipóteses de fechamento, resultando em cinco diferentes campos de precipitação para o dia 21 de fevereiro de 2004. Cada uma destas execuções equivale a um membro gerado pelo problema direto.

O resultado a ser utilizado para a análise final por parte de meteorologistas pode ser obtido via a adoção de um fechamento específico (m_{b_i}) ou pode ser composto pela média simples do *ensemble* do conjunto dos fechamentos,

$$\langle m_b \rangle = \frac{\sum_i m_{b_i}}{n}, \tag{4.26}$$

que para a simulação do dia especificado acima resulta no campo de precipitação dado pela Figura 4.4.

A Figura 4.5 apresenta a precipitação total acumulada (mm) em 24h para o dia

[1] Média simples entre o conjunto de hipóteses e fechamentos.

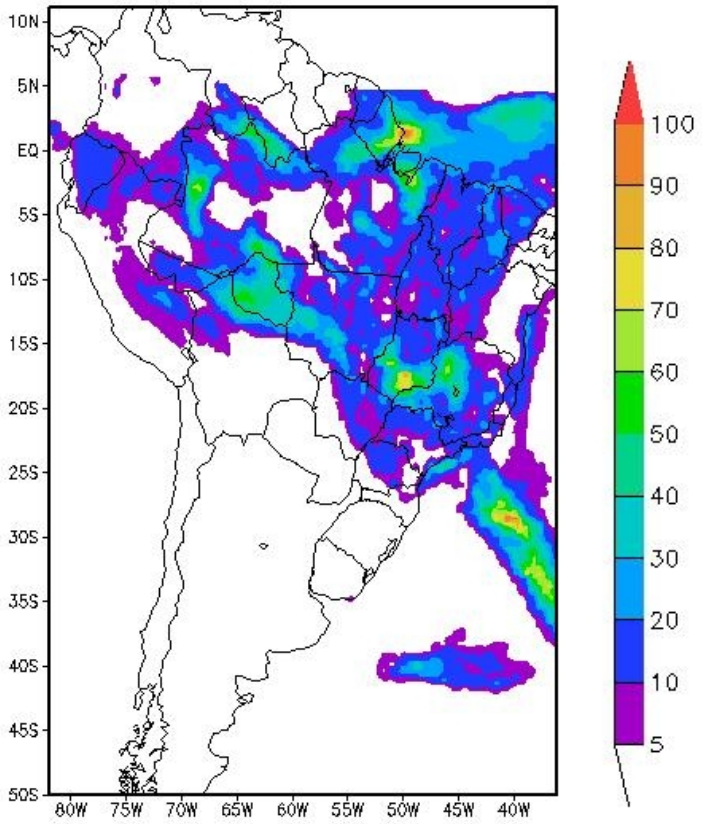


Figura 4.4 - Precipitação calculada pelo BRAMS com o uso do conjunto de hipóteses de fechamento de Grell e Dévényi (2002) para o dia 21 de fevereiro de 2004. Fonte: Santos et al. (2010b).

21 de fevereiro de 2004 estimada pelos sensores do satélite *Tropical Rainfall Measuring Mission* (TRMM), amplamente adotado por meteorologistas como fonte de observação real.

Nota-se claramente que o modelo direto atualmente usado para a estimação da precipitação via esquema de *ensemble* de diversas hipóteses de fechamento proposto por GD não possui grande similaridade com os dados observados. Este distanciamento dos resultados advém da equiprobabilidade imposta pela Equação 4.26.

Neste trabalho, desenvolvido em conjunto com a aluna de Doutorado Ariane Frassoni Santos (Programa de Pós-graduação em Meteorologia do INPE), utiliza-se da teoria de problemas inversos para a estimação dos pesos dos fechamentos usados por GD para uma melhor aproximação do campo de precipitação calculado.

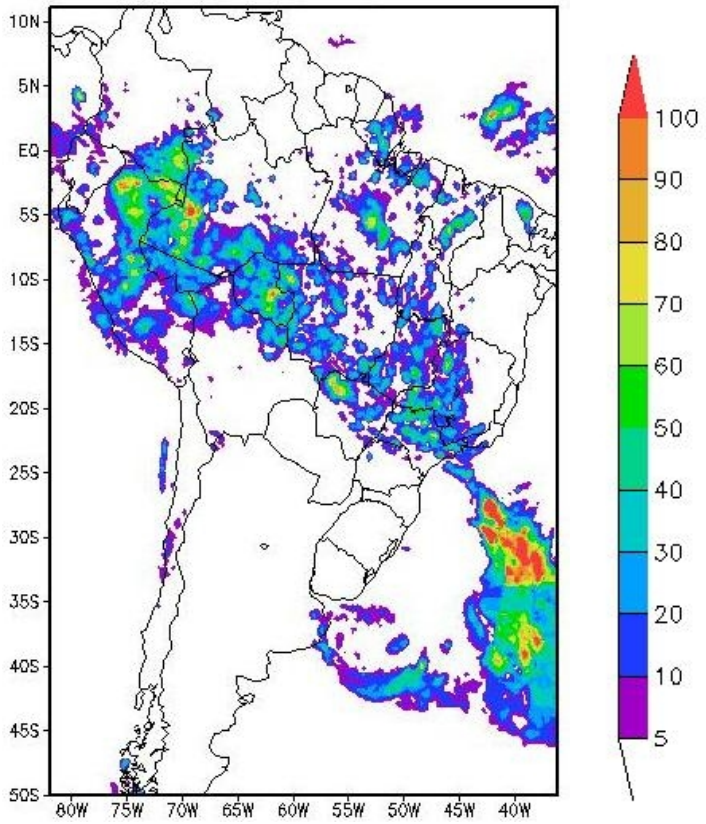


Figura 4.5 - Precipitação estimada pelo satélite TRMM para o dia 21 de fevereiro de 2004. Fonte: Santos et al. (2010a).

4.3.2 Modelo inverso

Utilizando as saídas das estimativas de precipitação fornecidas por cada hipótese de fechamento, um conjunto de dados sintéticos pode ser criado para a validação da metodologia proposta de que um problema inverso possa ser usado para estimar os pesos relativos às diferentes hipóteses de fechamento para a construção de um mapa de precipitação mais próximo da realidade.

A Equação 4.27 apresenta a equação a ser otimizada com restrição imposta pelo termo mais à direita. Esta otimização leva em consideração a diferença quadrática entre os campos calculados e os dados observados ou experimentais.

$$J\left(\vec{w},\alpha\right) = \left[\sum_{i=1}^{5} \left\| w_i P_i^{Mod} - P^{Obs} \right\|\right] + \alpha\Omega\left(\vec{w}\right) \tag{4.27}$$

A regularização de Tikhonov de ordem zero foi aplicada aos pesos dos campos de precipitação, buscando impor uma regularização de forma ao campo calculado.

Para validar o método, dados de precipitação sintéticos foram gerados tendo por base os campos de precipitação calculados pelas hipóteses de fechamento em uso pela metodologia de GD, i.e., AS, GR, KF, LO e MC. A geração dos dados constitui-se da atribuição de um conjunto de pesos de fechamento de soma igual a 1 para a construção de um campo final de precipitação (Equação 4.28).

$$P_s = w_{AS}P_{AS} + w_{GR}P_{GR} + w_{KF}P_{KF} + w_{LO}P_{LO} + w_{MC}P_{MC}, \tag{4.28}$$

onde w_i representa os pesos atribuídos a cada um dos P_i campos de precipitação calculados pelas diferentes hipóteses de fechamento em uso.

Após o cálculo do campo de precipitação sintético (P_s), um certo nível de ruído (σ) é adicionado ao campo (Equação 4.29), com o intuito de simular o erro experimental. Assume-se um erro Gaussiano de $\sigma = 2\%$.

$$P_o = Ps\,(1 + \sigma\mu) \tag{4.29}$$

Em suma, o objetivo do problema inverso expresso pela Equação 4.27 é o de estimar os valores dos w_i pesos de tal forma que se encontre um balanço entre os campos de precipitação estimados para as diferentes hipóteses de fechamento, desta maneira calculando o quão próximo um fechamento se encontra do campo de precipitação observado.

Esta técnica visa corrigir desvios causados pela a equiprobabilidade adotada na parametrização de GD e expresso na Equação 4.26. Os resultados serão apresentados na Subseção 5.2.3 e se constituem da estimativa, via problemas inversos, de dois conjuntos de dados sintéticos correspondentes aos dados de precipitação do dia 24 de fevereiro de 2004.

5 RESULTADOS

"You may never know what results come of your action,
but if you do nothing there will be no result."

(Mahatma Gandhi)

Este capítulo apresenta os resultados obtidos pelas aplicações das meta-heurísticas apresentadas no Capítulo 2 às soluções dos problemas inversos apresentados no Capítulo 4.

O foco é dado às variantes de meta-heurísticas desenvolvidas como fruto deste trabalho, especificamente o *Multiple Particle Collision Algoritm* (MPCA), descrito na Subseção 2.4.1, e o *parallel Firefly Algorithm with Predation* (pFAP), apresentado na Subseção 2.5.1.

No texto que segue, também serão apresentados resultados de validação destes algoritmos com funções de teste (*benchmark*) da literatura, além dos problemas inversos previamente descritos.

5.1 Resultados obtidos pelo algoritmo de Colisão de Múltiplas Partículas

Nesta seção serão apresentados os resultados obtidos pela aplicação do Algoritmo de Colisão de Múltiplas Partículas (*Multiple Particle Collision Algorithm*, MPCA) a funções de teste para confirmar a eficácia do seu uso e posteriormente serão apresentados os resultados obtidos pela aplicação do MPCA à solução do problema de termo de fonte/sumidouro de um gás na atmosfera apresentado em Seção 4.2.

5.1.1 Validação com funções de teste

As funções de teste, ou *benchmark*, usadas para teste foram as funções de Easom, Shekel, Rosenbrock e Griewank, sendo que Easom, Rosenbrock e Griewank são funções para minimização e a função de Shekel é de maximização.

A função Easom é expressa pela Equação 5.1, com seu espaço de buscas definido entre $-100 \leq x_i \leq 100$, $i = 1, 2$, com mínimo global em $x^* = (\pi; \pi)$ e $f(x^*) = -1$.

$$f(x) = -\cos(x_1)\cos(x_2)\exp(-(x_1-\pi)^2-(x_2-\pi)^2) \tag{5.1}$$

A função de Shekel, para maximização, está limitada entre $-65,536 \leq x_i \leq 65,536$ com máximo global em $x^* = (-32;-32)$ e $f(x^*) = 499,002$, sua expressão é dada pela Equação 5.3.

$$\begin{aligned} f(x) &= 500 - \frac{1}{0,002 + \sum_{i=0}^{24} \frac{1}{1+i+(x_1-a(i))^6+(x_2-b(i))^6}} \\ a(i) &= 16((i \bmod 5) - 2) \\ b(i) &= 16((i/5) - 2) \end{aligned} \tag{5.2}$$

A função de Rosenbrock para minimização no espaço entre $-2,048 \leq x_i \leq 2,048$ com mínimo global em $x^* = (1;1)$ e $f(x^*) = 0$ é representada pela Equação 5.3.

$$f(x) = 100(x_2 - x_1^2)^2 + (1-x_1)^2 \tag{5.3}$$

E finalmente, a função de Griewank, de topologia peculiar, que melhor demonstra a vantagem do MPCA sobre sua versão original. Tem delimitações entre $-600 \leq x_i \leq 600$ com mínimo global em $x^* = (0;0)$ e $f(x^*) = 0$, sendo representada pela Equação 5.4.

$$f(x) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) \tag{5.4}$$

A Tabela 5.1 apresenta um resumo dos mínimos e máximos globais para as funções teste usadas.

A Tabela 5.2 apresenta os resultados obtidos pelo PCA em comparação com a nova variante MPCA. Para ambos os algoritmos foram executados 10 experimentos independentes. Os parâmetros adotados foram: 100.000 iterações; 100 iterações de busca local com $LI_{SP} = 0,8$ e $LS_{SP} = 1,2$. O MPCA usou 24 partículas na busca colaborativa com ciclo de comunicação a cada 1.000 iterações.

Tabela 5.1 - Funções de teste usadas

Função	x^*	$f(x^*)$
Easom	$(\pi; \pi)$	-1
Shekel	$(-32; -32)$	$499,002$
Rosenbrock	$(1; 1)$	0
Griewank	$(0; 0)$	0

Tabela 5.2 - Comparação entre a média dos resultados obtidos com o PCA e MPCA para funções de teste. A linha ME apresenta o resultado médio para 10 experimentos independentes e a linha DP apresenta o desvio padrão.

Função		PCA	MPCA
Easom	ME	$(3,1398; 3,1415)$	$(3,1413; 3,1416)$
	DP	$(2,5246\text{E}{-3}; 2,3909\text{E}{-3})$	$(7,3212\text{E}{-4}; 8,3276\text{E}{-4})$
Rosenbrock	ME	$(0,9998; 0,9997)$	$(0,9999; 0,9999)$
	DP	$(7,6989\text{E}{-4}; 1,5595\text{E}{-3})$	$(1,4870\text{E}{-4}; 3,0400\text{E}{-4})$
Shekel	ME	$(-31,9793; -31,9788)$	$(-31,9790; -31,9787)$
	DP	$(4,5492\text{E}{-4}; 4,3423\text{E}{-4})$	$(9,1091\text{E}{-5}; 1,1009\text{E}{-4})$
Griewank	ME	$(1,2550; -0.8910)$	$(-5,9908\text{E}{-4}; 7,9774\text{E}{-3})$
	DP	$(2,1999; 3,5006)$	$(2,2532\text{E}{-2}; 2,8489\text{E}{-2})$

Aqui, os resultados demonstram que a exploração colaborativa do espaço de buscas por múltiplas partículas auxilia no escape de ótimos locais, tal como o evidenciado na minimização da função de Griewank. O tempo de execução para ambos os algoritmos é equivalente, sendo que um tempo adicional que variou de 2 a 4 segundos foi adicionado à execução do MPCA pela rotina de comunicação.

Para demonstração do uso massivo do MPCA, um caso de testes com a estimação de 100 variáveis foi executado, tendo por base a função de Griewank (Equação 5.4). O MPCA foi setado para ser executado em 384 processadores, 250.000 iterações, com ciclo de comunicações a cada 1.000 iterações, 100 iterações de busca local, $LI_{SP} = 0,5$ e $LS_{SP} = 1,5$, obtendo, para uma média de 10 experimentos, $f(x) = 6,2485\text{E}{-6}$, com um desvio padrão de $3,70506\text{E}{-6}$.

5.1.2 Aplicação ao problema de termo de fonte/sumidouro de um gás na atmosfera

O problema inverso utilizado na validação do MPCA é baseado em um estudo de caso apresentado em Roberti (2005).

Este caso estima a intensidade de emissão e absorção de duas fontes volumétricas com variação espacial e temporal. A variação espacial é representada por duas áreas de emissão/absorção horizontais próximas e a variação temporal explicita a transição entre o comportamento de emissão e absorção de um gás.

Esta característica temporal é equivalente ao comportamento de ciclo diurno de CO_2 onde, durante o processo de fotossíntese, plantas absorvem CO_2 durante o dia e emitem CO_2 durante a noite no processo de respiração.

A configuração do experimento é dada pela Figura 5.1, onde ambas as áreas A_1 e A_2 emitem gás para um tempo $t < 500$s, caracterizando a fonte de emissão, e absorvem gás para um tempo $500 < t < 1000$s, caracterizando o sumidouro do gás. A intensidade de emissão da Área 1 (A_1) é de $8g/s$ para o período de 0 a 500s de simulação e com absorção de $-4g/s$ nos 500s restantes. A Área 2 (A_2) emite $5g/s$ e absorve $-9g/s$ para os mesmos períodos de tempo.

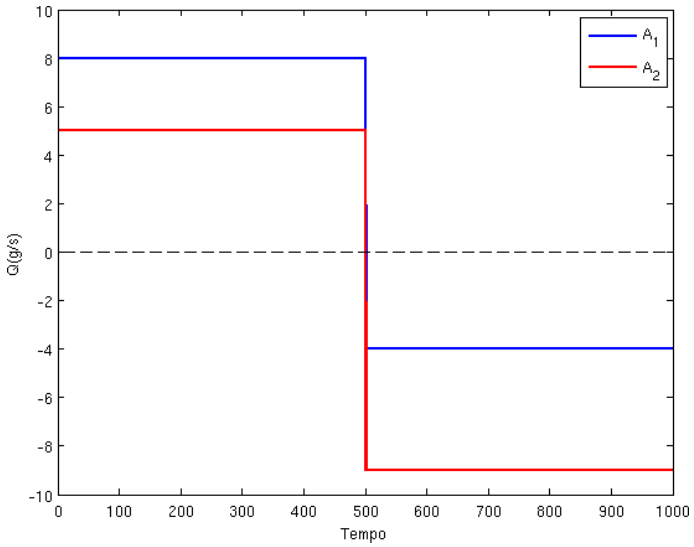


Figura 5.1 - Valores de emissão e suas variações das áreas 1 e 2 para o experimento.

Esta configuração é baseada na existência de uma área de floresta jovem (Área 2) em crescimento, que configura uma grande absorção de CO_2 durante o dia e emissão de CO_2 no período noturno para a manutenção de seu metabolismo basal. Por ser uma floresta jovem, em fase de crescimento, a quantidade de absorção de CO_2 é maior que a emissão, possibilitando o aumento de massa vegetal. A outra área representa

uma floresta senescente (Área 1), completando seu ciclo de vida natural, onde a quantidade de absorção de CO_2 durante o dia é reduzido e a emissão noturna de CO_2 é maior, ocasionando o decréscimo da massa vegetal.

O cenário computacional setado para este experimento possui altura da cobertura vegetal de 20m e área horizontal de 500m $\times$400m em terreno plano, para ambas as áreas. O centro da Área 1 está na posição $(300; 650)$ e a Área 2 está centralizada em $(300; 250)$. 56 sensores estão dispostos em 3 linhas paralelas a 40m de altura, com a primeira linha na marca de 800m da área, a segunda linha na marca de 1000m e a terceira linha na marca de 1200m. O posicionamento dos sensores se inicia na posição $y = 50$m e vai até 950m, distantes 50m entre si.

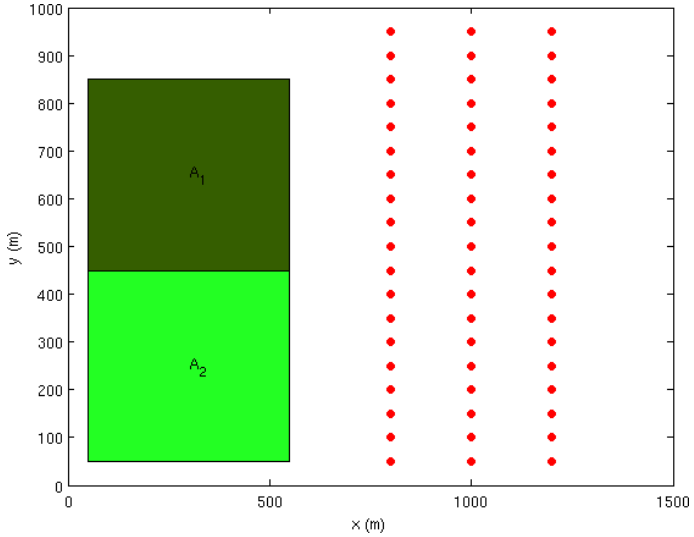


Figura 5.2 - Configuração da área usada no problema de termo de fonte em poluição atmosférica.

Os dados meteorológicos usados para a configuração deste experimento se baseiam nas características do experimento de Copenhagen para o dia 19/10/1978 das 12h00 às 12h45: $\overline{U}_{10m} = 2,6m/s$; $\overline{U}_{120m} = 5,7m/s$; $L = -71m$; e $h = 1120m$[1], com direção do vento em um ângulo de $\theta = 180°$.

A concentração do CO_2 foi calculada a cada 60s em três períodos ao longo da simulação. O primeiro período de 400 a 600s, o segundo de 600 a 800s e o terceiro de

[1]Em que $\overline{U}$ representa a velocidade do vento em 10 e 120m; L representa o comprimento de Monin-Obukhov; e h representa a altura da Camada Limite Planetária (CLP).

800 a 1000s. Uma média para cada período representa a concentração obtida pelo modelo:

$$C^{Mod}\left(x_j, p\right) = \frac{1}{n_f} \sum_{n=1}^{n_f} \sum_{i=1}^{2} \sum_{t^\star}^{2} \frac{Q_{i,t^\star}}{V_{s,j} N_{PEF,i}} \Delta t N_{PVS,i,j,n} \tag{5.5}$$

em que p é o período quando a concentração média foi calculada; n_f é o número de vezes em que a concentração foi calculada em cada período ($n_f = 3$ para o primeiro e terceiro períodos e $n_f = 4$ para o segundo período); x_j é a posição de cada um dos $j = 56$ sensores; $N_{PEF,i} = 2000$ é o número de partículas emitidas pela i-ésima fonte durante 1000s e um passo de tempo $\Delta t = 1$; $V_{s,j} = (25m \times 25m \times 10m)$ é o volume do j-ésimo sensor; $N_{PVS,i,j,n}$ é a quantidade de partículas emitidas da i-ésima fonte e detectadas no volume do j-ésimo sensor no instante n; $Q_{i,t^\star}$ é a intensidade de emissão/absorção da i-ésima fonte no intervalo de tempo $t^\star = 1$ se $t < 500$s ou $t^\star = 2$ se $t > 500$s.

De acordo com Roberti (2005), uma concentração homogênea de fundo de $0,05 g/m^3$ é considerada pré-existente no domínio, sendo acrescida nos resultados da Equação 5.5. Ruído gaussiano da ordem de 2% e 5% foram adicionados aos resultados do modelo direto para simular a obtenção de dados via sensores reais (sujeitos a erros de medida), na forma:

$$C_j\left(\vec{x}\right) = C_j\left(\vec{x}\right)\left[1 + \sigma\mu\right]. \tag{5.6}$$

A função custo de minimização para a estimação da intensidade de emissão e absorção das duas fontes neste experimento é descrita na forma:

$$F\left(\alpha, Q\right) = \sum_{i=1}^{56} \left[C_i^{Mod}\left(Q\right) + C_i^{Exp}\right]^2 + \alpha\Omega\left(Q\right), \tag{5.7}$$

em que o vetor Q representa a quantidade de emissão/absorção calculada para as duas fontes, $Q = [Q_{1A_1}, Q_{2A_1}, Q_{1A_2}, Q_{2A_2}]^T$, em que Q_{1A_1} e Q_{1A_2} representam as emissões das Áreas 1 e 2 respectivamente e Q_{2A_1} e Q_{2A_2} representam as absorções das Áreas 1 e 2 respectivamente.

A minimização do funcional expresso pela Equação 5.7, e consequentemente a solução do problema inverso de termo de fonte/sumidouro de poluição atmosférica foi conduzida pela aplicação do *Multiple Particle Collision Algorithm* (MPCA), apresentado na Subseção 2.4.1.

O MPCA foi executado em 240 processadores em uma máquina Cray XE6, sendo que cada processador executou 200 iterações no laço exterior, de busca global, e 200 iterações no laço interior, de busca local, com restrição a $\pm 20\%$ da exploração global. O processo de atualização via *blackboard* foi executado a cada 20 iterações. Foi utilizado o operador de regularização de Tikhonov de ordem zero, com $\alpha = 0,1$, atribuído empiricamente.

Os resultados médios e o desvio padrão para 25 experimentos independentes recuperados para o caso onde 2% de erro Gaussiano foi inserido nos dados é apresentado na Tabela 5.3. A Figura 5.3 plota os resultados obtidos em comparação com os valores exatos.

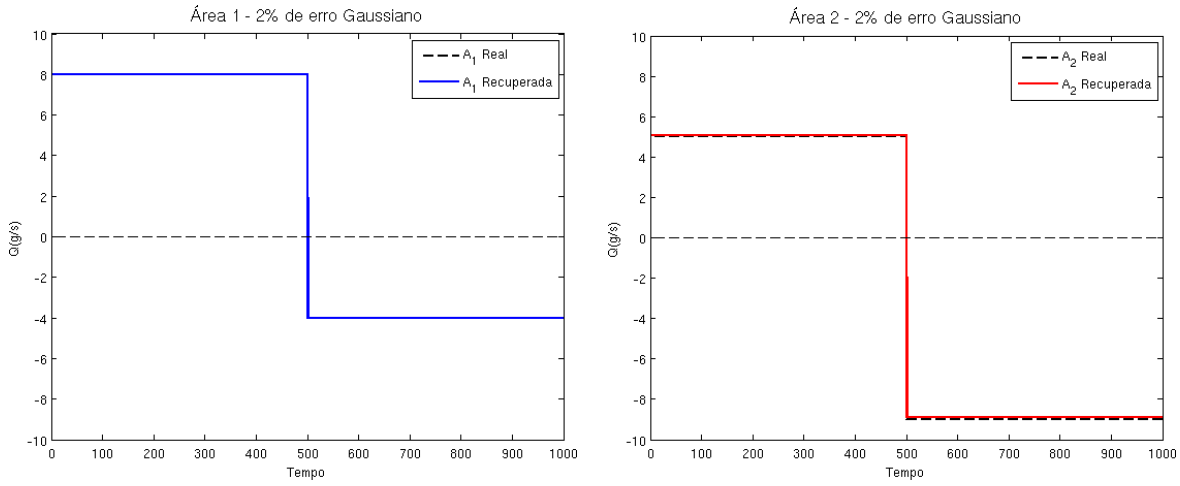
Tabela 5.3 - Resultados médios de 25 experimentos independentes estimados pelo MPCA com 2% de erro Gaussiano dos dados.

	$Q^{exato}(g/s)$	$\overline{Q^{estimado}}(g/s)$	Desvio padrão
Q_{1A_1}	$8,0$	$7,966$	$1,8\text{E}{-3}$
Q_{2A_1}	$-4,0$	$-4,029$	$1,2\text{E}{-3}$
Q_{1A_2}	$5,0$	$5,039$	$2,7\text{E}{-3}$
Q_{2A_2}	$-9,0$	$-8,896$	$1,6\text{E}{-3}$

Para comparação, a Tabela 5.4 apresenta os resultados obtidos pela aplicação da meta-heurística de Recozimento Simulado (*Simulated Annealing*) e publicados em Roberti (2005). Os resultados não apresentam desvio padrão para uma melhor comparação, mas já comprovam a eficiência equivalente do MPCA para a solução deste problema.

A Tabela 5.5 apresenta a média e desvio padrão de 25 experimentos com 5% de erro Gaussiano inserido nos dados. Os mesmos parâmetros adotados anteriormente foram usados neste caso. A Figura 5.4 apresenta os plotes em comparação com os dados exatos.

Para ambos os casos, com 2% e 5% de erro nos dados, o resultado recuperado foi



(a) (b)

Figura 5.3 - Recuperação obtida dos dados com 2% de erro pelo MPCA para as quantidades de emissão/absorção para: (a) Área 1; (b) Área 2.

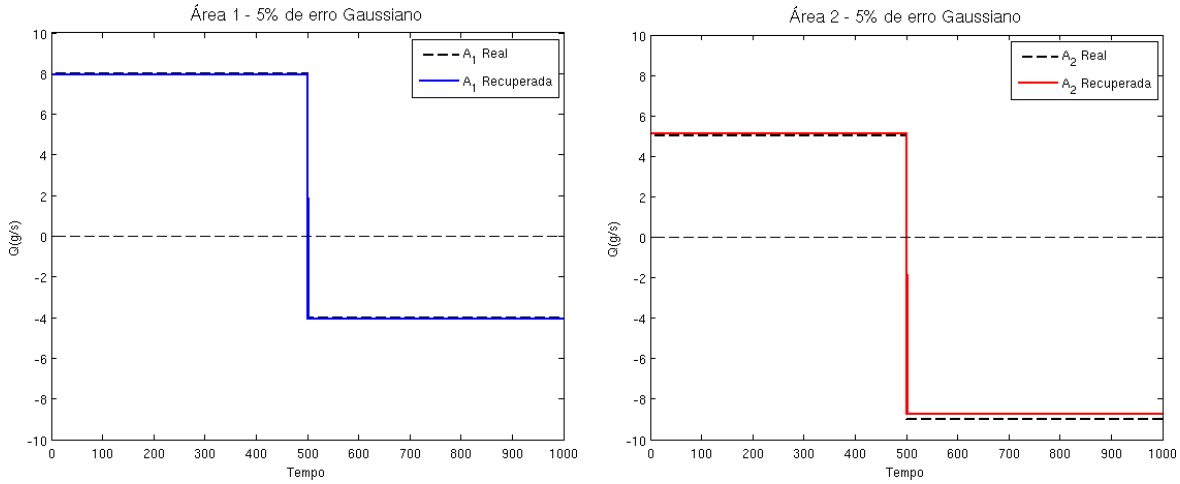
Tabela 5.4 - Resultado obtido pelo método SA. Estes resultados podem ser usados para uma comparação de equivalência dos resultados obtidos pelo MPCA (expressos na Tabela 5.3).

	$Q^{exato}(g/s)$	$Q^{estimado}(g/s)$
Q_{1A_1}	$8,0$	$7,92$
Q_{2A_1}	$-4,0$	$-4,03$
Q_{1A_2}	$5,0$	$4,93$
Q_{2A_2}	$-9,0$	$-9,02$

Fonte: Adaptado de Roberti (2005).

Tabela 5.5 - Resultados médios de 25 experimentos independentes estimados pelo MPCA com 5% de erro Gaussino dos dados.

	$Q^{exato}(g/s)$	$\overline{Q^{estimado}}(g/s)$	Desvio padrão
Q_{1A_1}	$8,0$	$7,925$	$1,8\text{E}{-3}$
Q_{2A_1}	$-4,0$	$-4,066$	$1,0\text{E}{-3}$
Q_{1A_2}	$5,0$	$5,098$	$1,7\text{E}{-3}$
Q_{2A_2}	$-9,0$	$-8,741$	$2,1\text{E}{-3}$



(a) (b)

Figura 5.4 - Recuperação obtida dos dados com 5% de erro pelo MPCA para as quantidades de emissão/absorção para: (a) Área 1; (b) Área 2.

satisfatório, sendo que somente a recuperação da quantidade de absorção para Área 2 se desviou em 2 g/s para o caso 2. Na resolução com o MPCA não foi utilizado nenhum conhecimento *a priori* e o espaço de buscas do algoritmo se estendeu de $[-10; 10]$ g/s. Uma maior quantidade de iterações e/ou processadores disponíveis pode melhorar o resultado, porém dada a alta complexidade temporal do modelo a ser resolvido para a obtenção do C^{Exp}, esta opção se torna um limitante.

5.2 Resultados obtidos pelo algoritmo de Vaga-lumes com Predação

Aqui apresentamos os resultados obtidos pela aplicação do Algoritmo paralelo de Vaga-lumes com Predação (pFAP) a funções de teste para confirmar a eficácia do seu uso. Também são apresentados os resultados obtidos pela aplicação do pFAP à solução do problema de identificação de condição inicial na equação do calor Seção 4.1 e na reconstrução de mapas de precipitação Seção 4.3.

5.2.1 Validação com funções de teste

Para a análise dos resultados, foram utilizadas as funções bidimensionais (2D) de teste, i.e., *benchmark*, de Ackley, Beale e Rastrigin.

A função de Ackley, em sua formulação n-dimensional para minimização, é expressa pela Equação 5.8 e encontra-se no domínio de $x_i \in (-32,768; 32,768)$, com ótimo

global em $x^* = \{0; \ldots; 0\}$ e $f(x^*) = 0,0$.

$$f(\vec{x}) = -20 \cdot \exp\left(-0.2\sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n} \cdot \sum_{i=1}^{n} \cos(2\pi x_i)\right) + 20 + \exp(1) \quad (5.8)$$

A função de Beale, limitada no espaço de buscas bidimensional, é expressa pela Equação 5.9 e tem seu espaço de buscas restrito a $-4,5 \leq x_i \leq 4,5$, com $i = \{1,2\}$. Seu ótimo global encontra-se em $x^* = \{3; 0,5\}$ e seu valor é de $f(x^*) = 0,0$.

$$f(\vec{x}) = (1,5 - x_1 + x_1 x_2)^2 + (2,5 - x_1 + x_1 x_2^2)^2 + (2,625 - x_1 + x_1 x_2^3)^2 \quad (5.9)$$

A função de Rastrigin é uma função n-dimensional, não-convexa, com espaço de buscas no domínio $-5,12 \leq x_i \leq 5,12$. Seu ótimo global está na posição $x^* = \{0; \ldots; 0\}$ com valor de $f(x^*) = 0,0$. Sua expressão é dada pela equação Equação 5.10.

$$f(\vec{x}) = An + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i)\right], \quad (5.10)$$

em que $A = 10,0$.

Os resultados aqui demonstrados são a média de 10 (dez) experimentos computacionais independentes. Para as funções de Ackley e Rastrigin, foram utilizados 20 vaga-lumes (soluções candidatas), destes, 5 são mantidos após o procedimento de predação, que o corre a cada 500 iterações. O critério de parada é de 100.000 iterações. Para a função de Beale, o critério de parada é o de precisão para o valor da função objetivo, neste caso 1×10^{-8}.

A Tabela 5.6 apresenta os resultados numéricos para a solução com *Firefly Algorithm* canônico e com o *parallel Firefly Algorithm with Predation*. As funções de Ackley e Rastrigin tem seu ótimo global em $\{0; 0\}$ e a função de Beale tem seu ótimo em $\{3,0; 0,5\}$. A linha ME apresenta o resultado obtido para a média da função objetivo e a linha DP apresenta o desvio padrão.

Tabela 5.6 - Resultados numéricos obtidos pelo FA canônico e o FAP com OpenMP.

Função		FA	pFAP
Ackley	ME	$(0,77; 0,21)$	$(-3,04\text{E}{-5}; -3,2\text{E}{-5})$
	DP	$(1,37; 1,53)$	$(1,17\text{E}{-4}; 1,68\text{E}{-4})$
Beale	ME	$(2,99; 0,50)$	$(3,00; 0,50)$
	DP	$(3,75\text{E}{-4}; 1,01\text{E}{-4})$	$(3,45\text{E}{-4}; 9,78\text{E}{-5})$
Rastrigin	ME	$(3,45\text{E}{-3}; -0,10)$	$(6,30\text{E}{-3}; -2,87\text{E}{-3})$
	DP	$(1,24; 1,18)$	$\{1,83\text{E}{-2}; 1,03\text{E}{-2})$

A função de Beale apresentou resultados equivalentes, porém a condição parada foi satisfeita após uma média de 239.012 avaliações para o FA e 218.570 avaliações para o pFAP.

5.2.2 Aplicação ao problema de identificação de condição inicial na equação do calor

O problema inverso de identificação da condição inicial na equação do calor foi resolvido usando o *parallel Firefly Algorithm with Predation* (pFAP), sendo que as equações que o governam foram apresentadas na Seção 4.1.

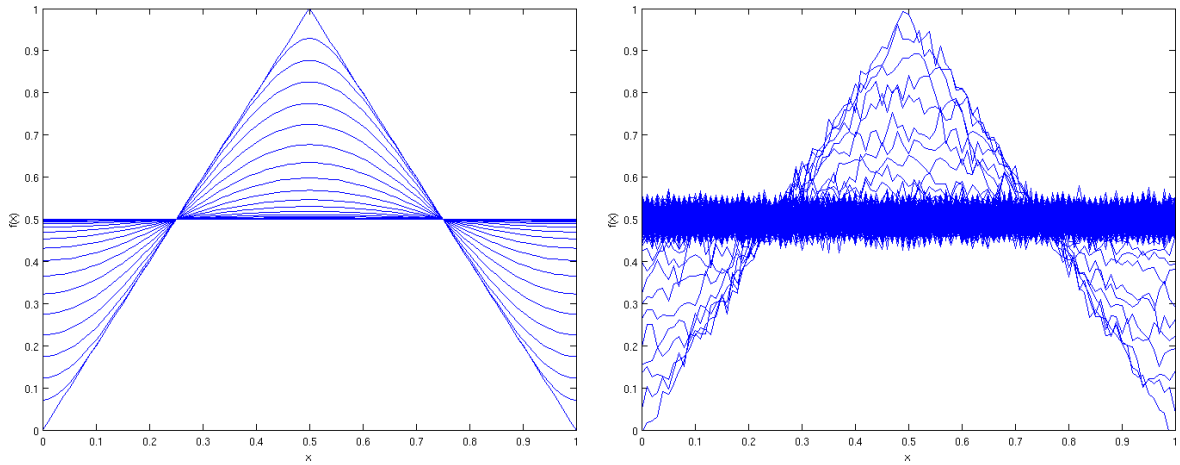
O perfil inicial de temperatura usado para os testes foi o triangular,

$$f(x) = \begin{cases} 2x & \text{se } 0 < x \leq 0,5 \\ 2(1-x) & \text{se } 0,5 < x \leq 1,0 \end{cases}, \tag{5.11}$$

com plote da evolução temporal visto na Figura 5.5(a). Na Figura 5.5(b) vemos a mesma evolução com adição de 2% de erro Gaussiano.

O número de pontos usados na discretização do problema, i.e., o número de pontos no eixo x é variável, e de ajuste por parte do usuário. Esta característica foi levada em consideração para criar 3 casos de teste usados neste trabalho.

O pFAP foi configurado para resolver o primeiro caso com 11 dimensões, i.e., 11 pontos de discretização espacial em x do problema, no tempo $t = 10^{-4}$. Foram utilizados 40 vaga-lumes, com um máximo de 1.000 iterações como condição de parada, sendo que o processo de predação é ativado a cada 200 iterações. Na ativação da predação, as 10 melhores soluções são mantidas e 30 novas soluções são criadas aleatoriamente



(a) (b)

Figura 5.5 - Evolução de um perfil triangular de temperatura inicial: (a) sem indução de erro nos dados; (b) com adição de 2% de erro nos dados.

no espaço de buscas. Outros parâmetros usados foram: $\alpha = 0,2$; $\gamma = 1,0$; $\beta_0 = 1,0$; e $m = 2$. O operador de regularização de Tikhonov de ordem 0 foi utilizado com um parâmetro de regularização no valor de $0,1$ atribuído empiricamente.

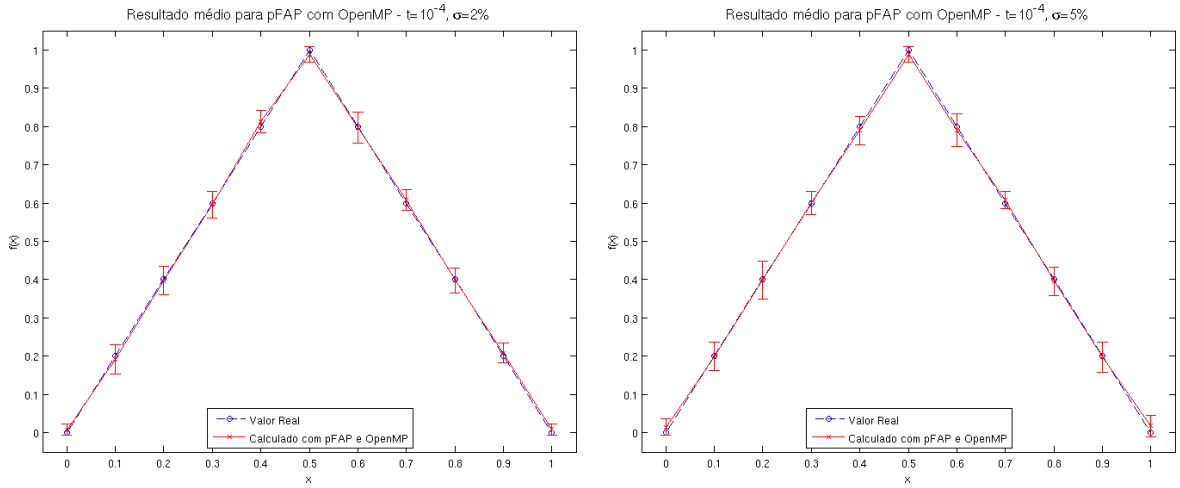
O computador usado para a execução do pFAP foi um Intel®Core™i7-820QM, de quatro núcleos, equipado com tecnologia Intel®Hyper-Threading. Foram utilizadas 4 *threads* para a execução do algoritmo, acarretando na carga de 10 vaga-lumes por *thread*.

A Figura 5.6 apresenta o resultado médio e o desvio padrão da inversão com 2% e 5% de erro respectivamente. 25 experimentos independentes, i.e., com sementes geradoras de números aleatórios distintos, foram usados para a construção do resultado médio.

O segundo caso de reconstrução da condição inicial de calor foi executado com 21 dimensões. O mesmo conjunto de parâmetros adotados para o primeiro caso foram utilizados com sucesso para esta reconstrução.

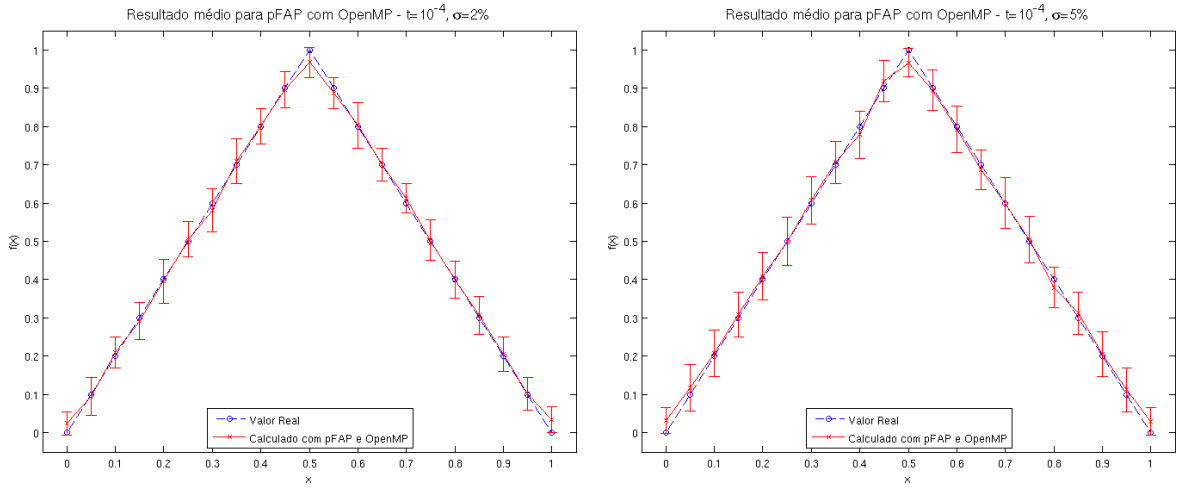
A Figura 5.7 apresenta o resultado médio e o desvio padrão da inversão com 2% e 5% de erro respectivamente. 25 experimentos independentes foram usados para a construção do resultado médio.

O terceiro caso usado neste trabalho foi configurado para a inversão em 51 dimensões,



(a) (b)

Figura 5.6 - Resultados da inversão com 11 dimensões no tempo $t = 10^{-4}$ obtida pelo pFAP com: (a) 2% de erro; (b) 5% de erro nos dados.

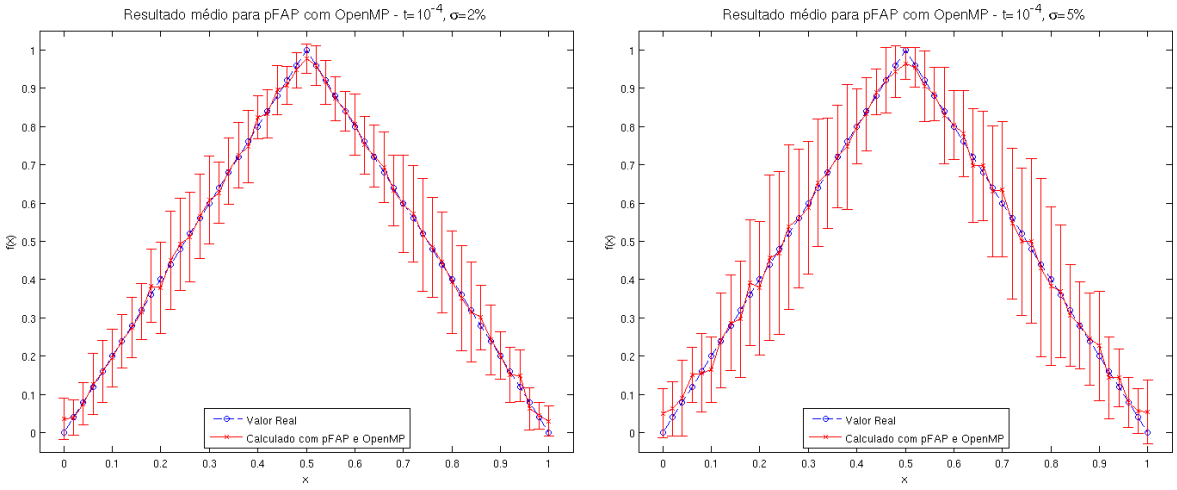


(a) (b)

Figura 5.7 - Resultados da inversão com 21 dimensões no tempo $t = 10^{-4}$ obtida pelo pFAP com: (a) 2% de erro; (b) 5% de erro nos dados.

i.e., 51 pontos de discretização no eixo x. O número de vaga-lumes, de iterações na condição de parada e passos necessários antes do processo de predação foram adaptados para refletir o aumento da complexidade requerida para uma boa inversão.

Os resultados médios (para 25 experimentos) da Figura 5.8 foram obtidos com o uso de 200 vaga-lumes sendo que a cada 500 iterações somente os 20 melhores sobrevivem ao processo de predação. A condição de parada foi aumentada para 50.000 iterações.



(a) (b)

Figura 5.8 - Resultados da inversão com 51 dimensões no tempo $t = 10^{-4}$ obtida pelo pFAP com: (a) 2% de erro; (b) 5% de erro nos dados.

Resultados obtidos com um tempo de inversão de $t = 10^{-3}$ requerem um aumento no número de vaga-lumes e iterações, consequentemente no número de avaliações da função objetivo, porém a qualidade da solução é mantida.

5.2.3 Aplicação ao problema de reconstrução de mapas de precipitação

O problema inverso de reconstrução de mapas de precipitação, tal com o descrito na Seção 4.3 foi inicialmente apresentado por Santos et al. (2010a) e Santos et al. (2010b).

Os resultados apresentados nesta seção demonstram a capacidade de inversão do *parallel Firefly Algorithm with Predation* (pFAP) para dois casos sintéticos criados especificamente para validar o modelo de inversão.

Os campos de precipitação foram gerados pelo modelo BRAMS simulando um dia de intensa precipitação na América do Sul, coincidente com um caso de ZCAS e diferente da equiprobabilidade imposta pela Equação 4.26 do método GD, os casos sintéticos aqui testados visam validar a capacidade do método de identificar diferentes pesos contribuintes para a criação de um campo de precipitação único.

As cinco matrizes geradas pelo BRAMS representam os (285×372) pontos de grade para a simulação sobre a América do Sul do dia 21 de fevereiro de 2004, tal com o descrito na Subseção 4.3.1.

A Tabela 5.7 apresenta os pesos utilizados na construção dos dois casos de testes sintéticos para os fechamentos de Arakawa e Schubert (AS), Grell (GR), Kain e Fritsch (KF), Low-level Omega (LO) e convergência de umidade (MC).

Tabela 5.7 - Pesos atribuídos aos fechamentos na criação dos casos sintéticos resolvidos neste trabalho. Adicionalmente a cada caso, dois cenários com erros Gaussianos de 2% e 5% são apresentados.

Fechamento	Caso 1	Caso 2
AS	$0,25$	$0,10$
GR	$0,35$	$0,10$
KF	$0,20$	$0,10$
LO	$0,15$	$0,60$
MC	$0,05$	$0,10$

O conjunto de pesos usado no Caso 1 foi atribuído de acordo com o conhecimento de um especialista em meteorologia. O Caso 2 foi construído a fim de testar um caso próximo ao extremo, onde um fechamento consegue representar bem a precipitação medida.

Os campos de precipitação sintético foram construídos com base na Equação 4.28, dados os pesos da Tabela 5.7 e sofreu posterior adição de erro Gaussiano, em níveis de 2% e 5%, de acordo com a Equação 4.29.

Para a solução do Caso 1, o pFAP foi configurado para executar a inversão com 40 vaga-lumes, tendo por condição de parada a execução de 200 iterações. O processo de predação foi ativado a cada 20 iterações e após este processo, somente as 10 melhores soluções eram mantidas. Outros parâmetros usados foram: $\alpha = 0,2$; $\gamma =$

$1,0$; $\beta_0 = 1,0$; e $m = 2$. Os resultados foram obtidos sem uso de regularização.

A Tabela 5.8 apresenta os resultados obtidos para o Caso 1, com adição de 2% de ruído. Nota-se a robustez do método dado o baixo desvio padrão para a execução de 25 experimentos independentes.

Tabela 5.8 - Resultado médio de 25 experimentos independentes obtido pelo pFAP na reconstrução de mapas de precipitação, para o primeiro caso sintético com $\sigma = 2\%$.

Fechamento	Exato	Média pFAP	Desvio Padrão
AS	$0,25$	$0,2483$	$3,15\text{E}{-2}$
GR	$0,35$	$0,3508$	$2,55\text{E}{-2}$
KF	$0,20$	$0,2000$	$5,40\text{E}{-3}$
LO	$0,15$	$0,1500$	$2,80\text{E}{-2}$
MC	$0,05$	$0,0540$	$2,14\text{E}{-2}$

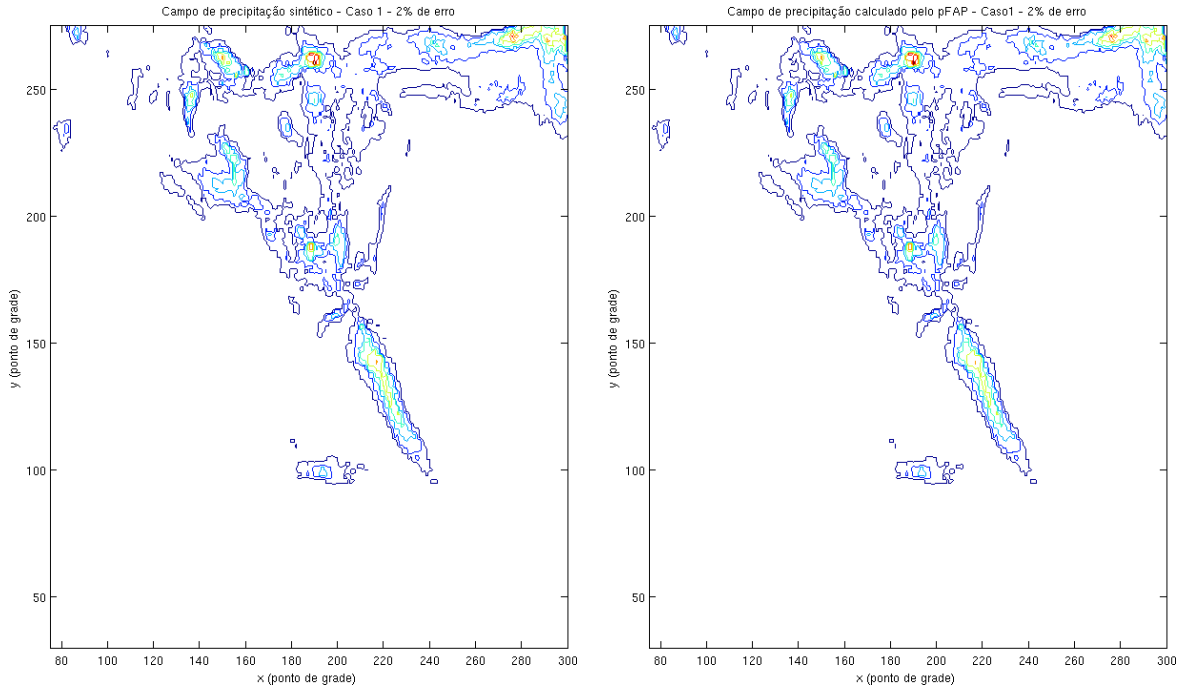
A Figura 5.9 permite a comparação dos campos de precipitação sintético e o gerado com os pesos de fechamento recuperados pelo pFAP para o Caso 1 com 2% de erro nos dados.

A Figura 5.10 mostra uma comparação entre o decaimento do valor da função objetivo para: (a) o pFAP e (b) o FA canônico. Nota-se o efeito benéfico da predação com o escape contínuo do procedimento de diversos ótimos locais. Neste caso, também é de nota que o FA canônico, dada a mesma configuração de parâmetros é incapaz de obter os mesmos resultados.

A Tabela 5.9 apresenta os resultados obtidos pelo FA canônico usados na construção da Figura 5.10. Nota-se que mesmo para 25 experimentos independentes todos os casos acabaram presos em um ponto de ótimo local, evidenciando a vantagem do mecanismo de predação utilizado no escape deste ponto.

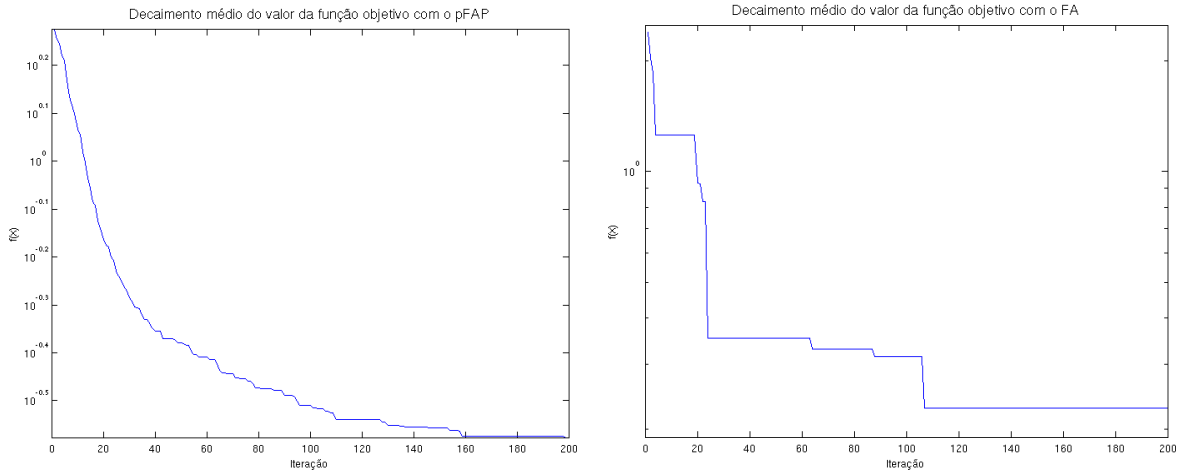
A Tabela 5.10 apresenta os resultados médios para o Caso 1 com adição de 5% de erro. Os mesmos parâmetros foram utilizados neste exemplo.

A Figura 5.11 permite a comparação dos campos de precipitação sintético e o campo gerado com os pesos de fechamento recuperados pelo pFAP para o Caso 1 com 5% de erro nos dados.



(a) (b)

Figura 5.9 - Comparação entre: (a) Campo de precipitação sintético do Caso 1 com 2% de erro; (b) Campo de precipitação reconstruído com os pesos calculados pelo pFAP.



(a) (b)

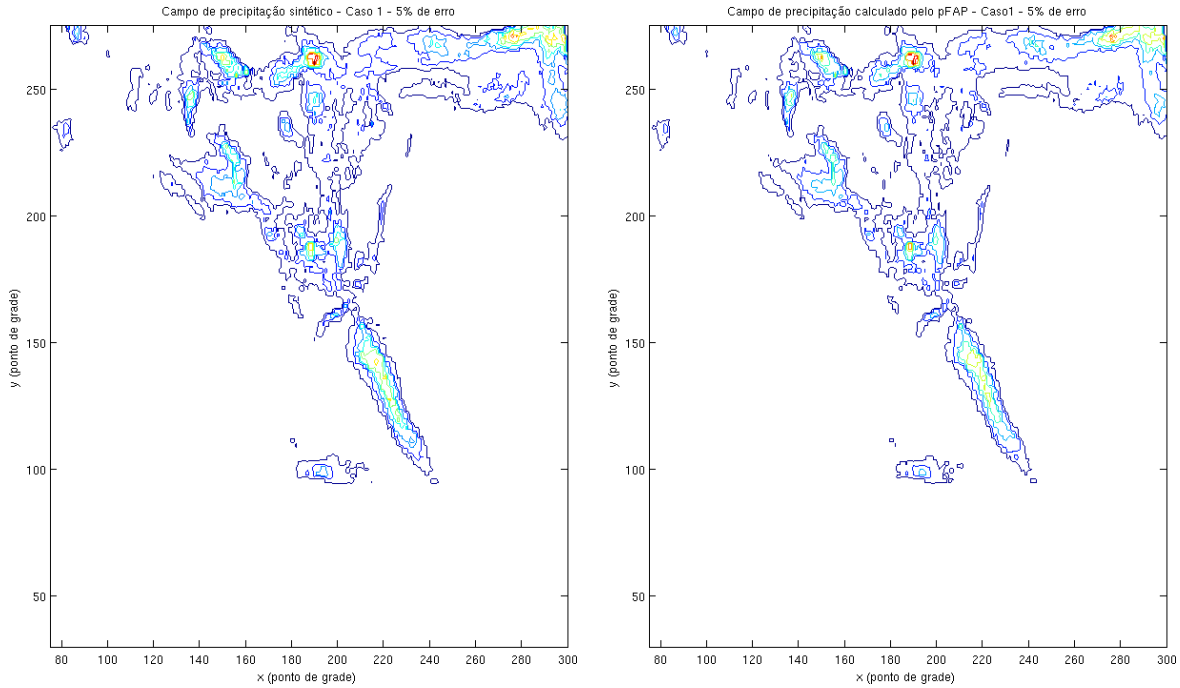
Figura 5.10 - Decaimento do valor da função objetivo para a resolução com: (a) pFAP; (b) FA canônico.

Tabela 5.9 - Resultado médio de 25 experimentos independentes obtido pelo FA canônico na reconstrução de mapas de precipitação, para o primeiro caso sintético com $\sigma = 2\%$.

Fechamento	Exato	Média FA	Desvio Padrão
AS	$0,25$	$0,2399$	$0,00E0$
GR	$0,35$	$0,3533$	$0,00E0$
KF	$0,20$	$0,2051$	$0,00E0$
LO	$0,15$	$0,1939$	$2,83E{-}17$
MC	$0,05$	$0,0619$	$7,08E{-}18$

Tabela 5.10 - Resultado médio de 25 experimentos independentes obtido pelo pFAP na reconstrução de mapas de precipitação, para o primeiro caso sintético com $\sigma = 5\%$.

Fechamento	Exato	Média pFAP	Desvio Padrão
AS	$0,25$	$0,2402$	$3,22E{-}2$
GR	$0,35$	$0,3423$	$2,12E{-}2$
KF	$0,20$	$0,2007$	$5,98E{-}3$
LO	$0,15$	$0,1561$	$3,11E{-}2$
MC	$0,05$	$0,0506$	$1,73E{-}2$



(a) (b)

Figura 5.11 - Comparação entre: (a) Campo de precipitação sintético do Caso 1 com 5% de erro; (b) Campo de precipitação reconstruído com os pesos calculados pelo pFAP.

O Caso 2 para a reconstrução do mapa de precipitação atribui maior peso a um fechamento específico. Os mesmos parâmetros utilizados para o Caso 1 foram suficientes para a recuperação dos pesos com adição de 2% e 5%.

A Tabela 5.11 contém o resultado médio e o desvio padrão de 25 experimentos independentes para o Caso 2, com 2% de erro Gaussiano nos dados. A Figura 5.12 apresenta a comparação entre o campo sintético gerado com os pesos atribuídos (Tabela 5.7) e os pesos calculados.

Tabela 5.11 - Resultado médio de 25 experimentos independentes obtido pelo pFAP na reconstrução de mapas de precipitação, para o segundo caso sintético com $\sigma = 2\%$.

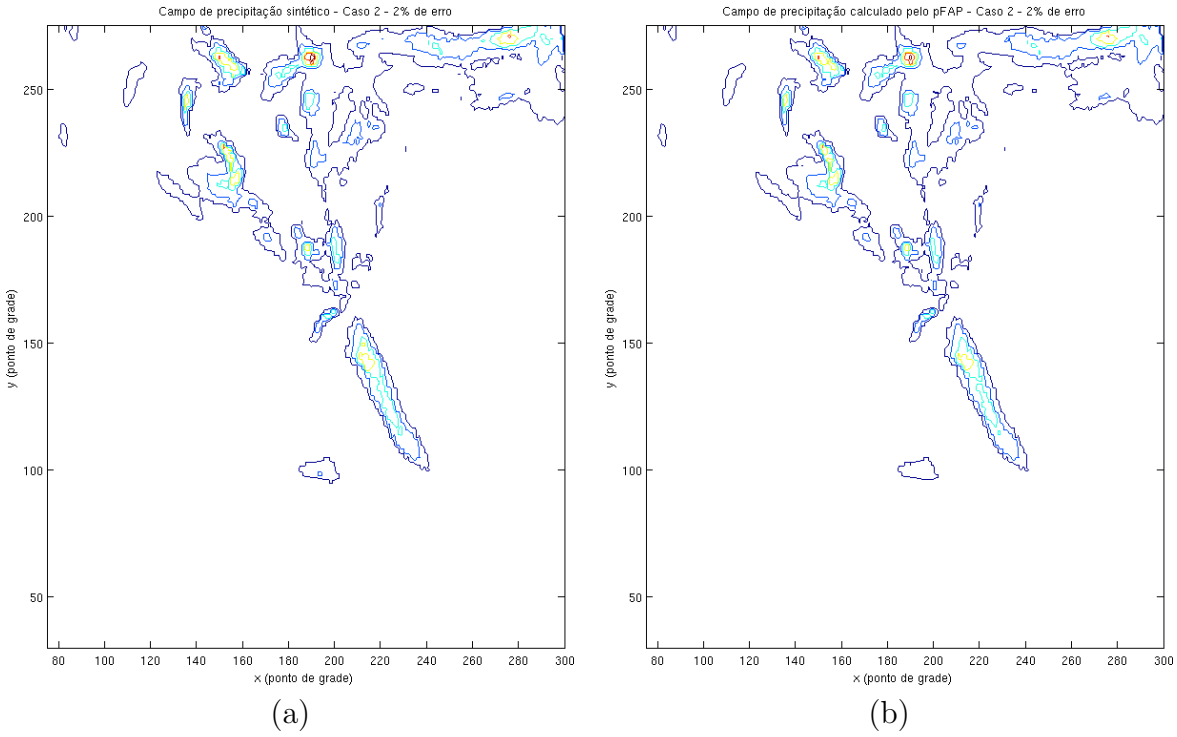
Fechamento	Exato	Média pFAP	Desvio Padrão
AS	$0,10$	$0,0942$	$2,86\text{E}{-2}$
GR	$0,10$	$0,1024$	$2,15\text{E}{-2}$
KF	$0,10$	$0,1016$	$3,99\text{E}{-3}$
LO	$0,60$	$0,5969$	$2,74\text{E}{-2}$
MC	$0,10$	$0,0997$	$1,97\text{E}{-2}$

A Tabela 5.12 apresenta o resultado médio e o desvio padrão de 25 experimentos independentes para o Caso 2, com 5% de erro Gaussiano nos dados. A Figura 5.13 apresenta o campo sintético gerado com os pesos atribuídos (Tabela 5.7) e o campo gerado com os pesos calculados.

Tabela 5.12 - Resultado médio de 25 experimentos independentes obtido pelo pFAP na reconstrução de mapas de precipitação, para o segundo caso sintético com $\sigma = 5\%$.

Fechamento	Exato	Média pFAP	Desvio Padrão
AS	$0,10$	$0,1036$	$2,56\text{E}{-2}$
GR	$0,10$	$0,1101$	$1,92\text{E}{-2}$
KF	$0,10$	$0,0966$	$7,34\text{E}{-3}$
LO	$0,60$	$0,6078$	$2,30\text{E}{-2}$
MC	$0,10$	$0,1032$	$1,95\text{E}{-2}$

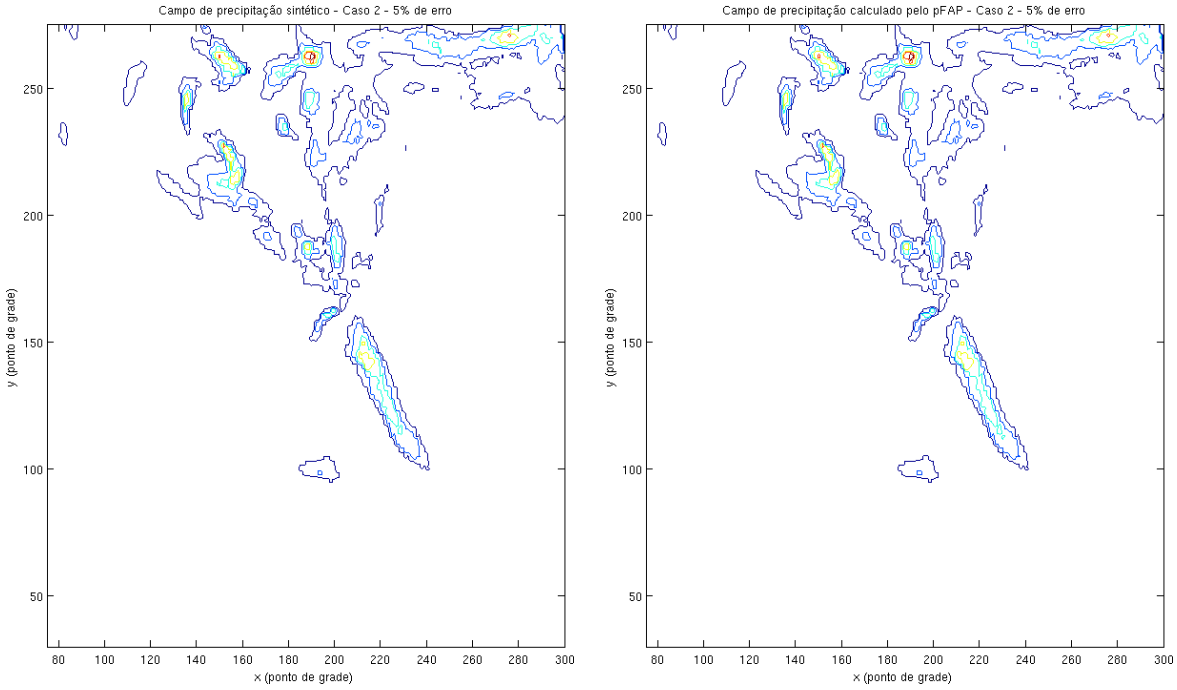
Os resultados apresentados aqui só levaram em consideração casos sintéticos. Os trabalhos de Santos et al. (2010a) e Santos et al. (2010b), assim como a tese de



(a) (b)

Figura 5.12 - Comparação entre: (a) Campo de precipitação sintético do Caso 2 com 2% de erro; (b) Campo de precipitação reconstruído com os pesos calculados pelo pFAP.

doutorado em curso estudam também a capacidade deste método em estimar pesos para um conjunto de dados reais obtidos via TRMM. As implicações meteorológicas da recuperação destes pesos ainda estão em estudo.



(a) (b)

Figura 5.13 - Comparação entre: (a) Campo de precipitação sintético do Caso 2 com 5% de erro; (b) Campo de precipitação reconstruído com os pesos calculados pelo pFAP.

6 CONCLUSÕES E CONSIDERAÇÕES FINAIS

"Omnis Ars Nature Imitio Est"

(Lucius Annaeus Seneca)

A recente popularização de computadores pessoais munidos de processadores com múltiplos núcleos tornou possível o desenvolvimento de métodos e algoritmos paralelos sem a necessidade de acesso a um sistema computacional especializado, e.g., supercomputadores. Porém, ao mesmo tempo, os sistemas especializados também têm se beneficiado da redução dos custos de produção e encontrar sistemas com 10.000 processadores ou mais tem se tornado comum. O aproveitamento deste novo conjunto de sistemas, em especial nos computadores pessoais com múltiplos núcleos, é essencial para uma melhor utilização dos recursos disponíveis, especialmente no campo da computação científica, onde a quantidade crescente de dados a serem analisados requer maior poder de processamento.

Este trabalho apresentou como produto principal duas novas meta-heurísticas paralelas, a primeira desenvolvida para operar em sistemas massivamente paralelos de memória distribuída (MPCA) e a segunda para operar em sistemas de múltiplos núcleos e memória compartilhada (pFAP). Ambas foram desenvolvidas a partir de versões canônicas seriais e em um dos casos um mecanismo adicional de escape de ótimos locais foi adicionado.

Além da característica paralela, a inspiração na natureza foi um fator determinante. A transposição e consequente utilização de estratégias de solução de problemas validadas por milhões de anos de evolução para o domínio computacional demonstra a viabilidade de sua constante mimetização na solução de problemas de interesse científico, industrial e comercial.

O primeiro conjunto de resultados apresentados, para o *Multiple Particle Collision Algorithm* (MPCA), algoritmo populacional desenvolvido para ambientes de processamento massivamente paralelos, demonstram a sua capacidade de resolver problemas com aplicações reais, tal como a estimação de fonte/sumidouro de um gás. Dada a sua capacidade de execução paralela em múltiplos processadores, seu uso pode ser recomendado para os casos onde o tempo necessário à avaliação da

função objetivo é extremamente custoso, permitindo assim, uma melhor exploração do espaço de busca. Em um caso extremo, a estimação de até 100 parâmetros foi feita com sucesso, dada uma quantidade suficiente de processadores para garantir a obtenção do resultado.

Os resultados apresentados para o *parallel Firefly Algorithm with Predation* (pFAP), desenvolvido para operar em sistemas computacionais pessoais de múltiplos núcleos com memória distribuída, chegaram a demonstrar, em um caso específico, a capacidade de *speedup* super-linear, obtido por possível efeito de cache. Além da paralelização, este algoritmo incorporou um novo mecanismo inspirado na natureza, especificamente a predação, que tem demonstrado a capacidade de agilizar o escape de ótimos locais. Com este algoritmo, problemas com até 50 variáveis a serem estimadas apresentaram resultado satisfatório e garantem a utilização deste método em diversas situações, tal como em processos de previsão do tempo.

Com os novos algoritmos aqui apresentados, a garantia da utilização efetiva dos crescentes recursos computacionais disponíveis, em especial os múltiplos núcleos que na maioria das vezes ficam ociosos, podem permitir a exploração desta nova enxurrada de dados que vêm sendo constantemente gerados.

Além dos novos algoritmos paralelos de otimização, este trabalho também gerou resultados secundários interessantes. O primeiro, fruto de uma orientação de iniciação científica, estudou a capacidade do ajuste automático de parâmetros, em detrimento do ajuste empírico comumente usado. A segunda contribuição, de caráter didático, apresentou uma visão alternativa sobre a relação dos diversos componentes envolvidos na solução de um problema inverso, desde o passo de modelagem iterativa até a possível introdução de informação *a priori* na solução do problema inverso.

Trabalhos futuros a partir dos algoritmos e resultados aqui apresentados incluem: (i) a hibridização do algoritmo MPCA com OpenMP para otimizar a exploração de espaços de buscas de altas dimensões; (ii) da mesma forma hibridizar o pFAP com MPI e implementar um possível esquema de ilhas de indivíduos com eventual migração de melhores indivíduos; (iii) avaliar possível refinamento da representação da visão alternativa para o ciclo de solução de problemas inversos; (iv) aprofundar os estudos no esquema de seleção automática de parâmetros; (v) estudar os impactos da adoção de métodos de busca local determinísticos no MPCA; (vi) aplicar os métodos a outras classes problemas de otimização.

A computação e suas aplicações estão em constante evolução. O hardware que serve de base para estes também. Da mesma forma, os algoritmos e métodos também devem evoluir e melhor aproveitar estes recursos. Este trabalho mostra que o desenvolvimento, a atualização e a melhoria de técnicas existentes para garantir a constante evolução científica é benéfica e aconselhada.

REFERÊNCIAS BIBLIOGRÁFICAS

ADVANCED MICRO DEVICES. **Physical cores v. enhanced threading software: Performance evaluation whitepaper**. 2010. On-line. Acesso em: 07/fev/2011. Disponível em: <http://www.amd.com/us/Documents/Cores_vs_Threads_Whitepaper.pdf>. 46

ANTONIOU, A.; LU, W.-S. **Practical optimization: algorithms and engineering applications**. New York: Springer, 2007. 5

BALDICK, R. **Applied optimization: Formulation and algorithms for engineering systems**. Cambridge, UK: Cambridge University Press, 2006. 824 p. 7, 8

BECCENERI, J. C. Computação e matemática aplicadas às ciências e tecnologias espaciais. In: _____. [S.l.]: LAC-INPE, 2008. cap. Meta-heurísticas e otimização combinatória: aplicações em problemas ambientais, p. 65–81. 5

BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. **ACM Computing Surveys**, v. 35, n. 3, p. 268–308, 2003. 11, 12

Campos Velho, H. F.; SHIGUEMORI, E. H.; RAMOS, F. M.; CARVALHO, J. C. A unified regularization theory: The maximum non-extensive entropy principle. **Computational & Applied Mathematics**, v. 25, n. 2–3, p. 307–330, 2006. 61

CAMPOS VELHO, H. F. de. **Problemas inversos em pesquisa espacial**. São Carlos: SBMAC, 2008. 124 p. 57, 60, 61

CRAY. **Using Cray performance analysis tools**. S-2376-52. Saint Paul, MN, USA, April 2011. Disponível em: <http://docs.cray.com/books/S-2376-52/S-2376-52.pdf>. 32

DIWEKAR, U. **Introduction to applied optimization**. 2nd ed.. ed. New York, NY: Springer, 2008. 291 p. (Springer Optimization and Its Applications, v. 22). 8

ENGL, H. W.; HANKE, M.; NEUBAUER, A. **Regularization of inverse problems**. Dordrecht: Kluwer, 1996. 56, 58, 62, 63

FLOUDAS, C. A.; PARDALOS, P. M. (Ed.). **Encyclopedia of optimization**. 2nd ed.. ed. New York: Springer, 2009. (Springer Reference). 9, 10, 11, 47, 48

FLYNN, M. J. Very high-speed computing systems. **Proceedings of IEEE**, v. 54, n. 12, p. 1901–1909, 1966. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1447203>. 13

FREITAS, S. R.; LONGO, K. M.; DIAS, M. A. F. S.; CHATFIELD, R.; DIAS, P. S.; ARTAXO, P.; ANDREAE, M. O.; GRELL, G.; RODRIGUES, L. F.; FAZENDA, A.; PANETTA, J. The coupled aerosol and tracer transport model to the brazilian developments on the regional atmospheric modeling system (catt-brams) - part 1: Model description and evaluation. **Atmospheric Chemistry and Physics**, v. 9, p. 2843–2861, 2009. Disponível em: <www.atmos-chem-phys.net/9/2843/2009/>. 77

GLOVER, F. Future paths for integer programming and links to artificial intelligence. **Comput. & Ops. Res.**, v. 13, n. 5, p. 533–549, 1986. 10

GOLDBARG, M. C.; LUNA, H. P. L. **Otimização combinatória e programação linear: modelos e algoritmos**. 2. ed. Rio de Janeiro: Elsevier, 2005. 518 p. 55

GRELL, G. A.; DéVéNYI, D. A generalized approach to parameterizing convection combining ensemble and data assimilation techniques. **Geophysical Research Letters**, v. 29, n. 14, p. 1963, 2002. xvii, 77, 79

HADAMARD, J. **Lectures on Cauchy's problem in linear partial differential equations**. New Haven: Yale University Press, 1923. 316 p. (Mrs. Hepsa Ely Silliman Memorial Lectures). 57

HANSEN, P. C. Analysis of discrete ill-posed problems by means of the l-curve. **SIAM Rev.**, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, v. 34, n. 4, p. 561–580, 1992. ISSN 0036-1445. 62

HENDERSON, D.; JACOBSON, S. H.; JOHNSON, A. W. Handbook of metaheuristic. In: _____. Dordrecht: Kluwer Academic Publishers, 2003. (International Series in Operations Research & Management Science), capítulo The theory and practice of simulated annealing, p. 287–320. 47

HEY, T.; TANSLEY, S.; TOLLE, K. (Ed.). **The fourth paradigm: Data-intensive scientific discovery**. Redmond, WA: Microsoft Research, 2009. 252 p. Disponível em:

<http://research.microsoft.com/en-us/collaboration/fourthparadigm/>. 1

HO, Y.-C.; PEPYNE, D. L. Simple explanation of the no-free-lunch theorem and its implications. **Journal of Optimization Theory and Applications**, v. 115, n. 3, p. 549–570, 2002. 10.1023/A:1021251113462. Disponível em: <http://dx.doi.org/10.1023/A:1021251113462>. 3

_____. Simple explanation of the no free lunch theorem of optimization. **Cybernetics and Systems Analysis**, v. 38, n. 2, p. 292–298, 2002. 10.1023/A:1016355715164. Disponível em: <http://dx.doi.org/10.1023/A:1016355715164>. 3

INTEL. **Intel math kernel library reference manual**. Mkl 10.3 update 8. [S.l.], 2011. Disponível em: http://software.intel.com/sites/products/documentation/hpc/mkl/mklman/mklman.pdf. 42

KAO, M.-Y. (Ed.). **Encyclopedia of algorithms**. New York, NY: Springer, 2008. 1166 p. (Springer Reference). 8

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. **Science**, v. 220, p. 671–680, 1983. 20, 47

LEWIS, S. M.; CRATSLEY, C. K. Flash signal evolution, mate choice, and predation in fireflies. **Annual Review of Entomology**, v. 53, p. 293–321, 2008. Disponível em: <http://www.annualreviews.org/doi/abs/10.1146/annurev.ento.53.103106.093346>. 38

LUKE, S. **Essentials of metaheuristics**. [S.l.: s.n.], 2010. Disponível em http://cs.gmi.edu/∼sean/book/metaheuristics/. 12

LUZ, E. F. P.; BECCENERI, J. C.; Campos Velho, H. F. Uma nova metaheurística baseada em algoritmo de colisão de múltiplas partículas. In: XI SIMPÓSIO DE PESQUISA OPERACIONAL E LOGÍSTICA DA MARINHA. **Anais...** Rio de Janeiro: CASNAV, 2008. 27

LUZ, E. F. P.; CAMPOS VELHO, H. F.; BECCENERI, J. C.; ROBERTI, D. R. Estimating atmospheric area source strengh through particle swarm optimization. inverse problems. In: INVERSE PROBLEMS, DESIGN AND OPTIMIZATION. **Proceedings...** Florida: IPDO 2007, 2007. CD-ROM. 68

LUZ, E. F. P. da. **Estimação de fonte de poluição atmosférica usando otimização por enxame de partículas**. Dissertação (Dissertação de Mestrado em Computação Aplicada) — Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2008. Disponível em: <http://urlib.net/sid.inpe.br/mtc-m17@80/2008/02.12.12.07>. 70, 76

MARON, O.; MOORE, A. W. The racing algorithm: Model selection for lazy learners. **Artificial Intelligence Review**, v. 11, p. 193–225, 1997. Disponível em: <ftp://ftp.ai.mit.edu/pub/users/oded/paper.ps>. 48, 50

MARR, D. T.; BINNS, F.; HILL, D. L.; HINTON, G.; KOUFATY, D. A.; MILLER, J. A.; UPTON, M. Hyper-threading technology architecture and microarchitecture. **Intel Technology Journal**, Q1 2002. Disponível em: <http://software.intel.com/en-us/articles/hyper-threading-technology-architecture-and-microarchitecture/>. 44

METROPOLIS, N.; ROSENBLUTH, A. W.; ROSENBLUTH, M. N.; TELLER, A. H.; TELLER, E. Equation of state calculations by fast computing machines. **The Journal of Chemical Physics**, AIP, v. 21, n. 6, p. 1087–1092, 1953. Disponível em: <http://link.aip.org/link/?JCP/21/1087/1>. 22, 47

MIRANDA, F. M. **Statistical racing em algoritmos evolucionários**. São José dos Campos/SP: [s.n.], 2011. Relatório final de projeto de iniciação científica pelo PIBIC/CNPq/INPE, orientado por Eduardo Fávero Pacheco da Luz (CAP/INPE). 46, 50

MUNIZ, W. B. **Um problema inverso em condução do calor utilizando métodos de regularização**. Dissertação (Dissertação de Mestrado em Matemática Aplicada) — Universidade Federal do Rio Grande do Sul, Porto Alegre, 1999. 66, 67

OPENMP. **OpenMP application program interface**. Version 3.1. July 2011. Disponível em: <http://www.openmp.org/mp-documents/OpenMP3.1.pdf>. 42

PRESS, W. H.; TEUKOLSKY, S. A.; VETTERLING, W. T.; FLANNERY, B. P. **Numerical recipes in Fortran 77: The art of scientific computing**. 2nd ed.. ed. New York, NY: Cambridge University Press, 1992. 1486 p. Disponível em: <http://apps.nrbook.com/fortran/index.html>. 35

_____. **Numerical recipes in Fortran 90: The art of parallel scientific computing**. 2nd ed.. ed. New York, NY: Cambridge University Press, 1996. 963 p. Disponível em: <http://apps.nrbook.com/fortran/index.html>. 35, 36

ROBERTI, D. R. **Problemas inversos em física da atmosfera**. Tese (Tese de Doutorado em Física) — Universidade Federal de Santa Maria, Santa Maria, 2005. 60, 61, 70, 73, 76, 85, 88, 89, 90

SABATIER, P. C. Inverse problems: An introduction. **Inverse Problems**, v. 1, n. 1, 1985. 54

SACCO, W. F.; FILHO, H. A.; PEREIRA, C. M. N. A. Cost-based optimization of a nuclear reactor core design: a preliminary model. In: INTERNATIONAL NUCLEAR ATLANTIC CONFERENCE. **Proceedings...** Santos: ABEN, 2007. 20

SACCO, W. F.; LAPA, C. M. F.; PEREIRA, C. M. N. A.; FILHO, H. A. A. A metropolis algorithm applied to a nuclear power plant auxiliary feedwater system surveillance tests policy optimization. **Progress in Nuclear Energy**, v. 50, p. 15–21, 2008. 20

SACCO, W. F.; OLIVEIRA, C. R. E.; PEREIRA, C. M. N. A. Two stochastic optimization algorithms applied to nuclear reactor core design. **Progress in Nuclear Energy**, v. 48, p. 525–539, 2006. 20

SACCO, W. F.; OLIVEIRA, C. R. E. A. A new stochastic optimization algorithm based on a particle collision metaheuristic. In: 6TH WORLD CONGRESS OF STRUCTURAL AND MULTIDISCIPLINARY OPTIMIZATION. **Proceedings...** Rio de Janeiro: WCSMO, 2005. 20

SAMBATTI, S. B. M. **Diferentes estratégias de paralelização de um algoritmo genético epidêmico aplicadas na solução de problemas inversos**. Dissertação (Dissertação de Mestrado em Computação Aplicada) — Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2004. 16, 30

SANTOS, A. F.; FREITAS, S. R.; LUZ, E. F. P.; VELHO, H. F. C.; GAN, M. A. Optimization firefly method for weighted ensemble of convective parametrizations. part ii: Sensitivity experiment using trmm satellite data. In: MEETING OF THE AMERICAS. **Anals...** Foz do Iguassú: AGU, 2010. 77, 78, 80, 96, 101

SANTOS, A. F. S. F.; FREITAS, S. R.; LUZ, E. F. P.; VELHO, H. F. C.; GAN, M. A. Optimization firefly method for weighted ensemble of convective parametrizations. part i: Results with a synthetic precipiration field. In: MEETING OF THE AMERICAS. **Anals...** Foz do Iguassú: AGU, 2010. 77, 78, 79, 96, 101

SCALES, J. A.; SNIEDER, R. The anatomy of inverse problems. **Geophysics**, v. 65, n. 6, p. 1708–1710, Nov-Dec 2000. 63

SILVA NETO, A. J. da; MOURA NETO, F. D. **Problemas inversos: Conceitos fundamentais e aplicações**. Rio de Janeiro: UERJ, 2005. 172 p. 57

SOTERRONI, A. C.; LUZ, E. F. P.; RAMOS, F. M.; VELHO, H. F. C.; BECCENERI, J. C.; GALSKI, R. L. Hibridização do algoritmo de colisão de partículas com o método extrem. In: CONGRESSO NACIONAL DE MATEMÁTICA APLICADA E COMPUTACIONAL 2009. **Anais...** Campinas, 2009. v. 2. ISSN 1984-920X. 25

SUTTER, H. **Going superlinear**. January 2008. On-line magazine. Disponível em: <`http://drdobbs.com/cpp/206100542`>. 19

SZWARCFITER, J. L.; MARKENZON, L. **Estruturas de dados e seus algoritmos**. 2. ed. Rio de Janeiro: LTC Editora, 1994. 320 p. 29

TALBI, E.-G. **Metaheuristics: From design to implementation**. [S.l.]: Wiley, 2009. 593 p. 10, 48

TARANTOLA, A. **Inverse problem theory and methods for parameter estimation**. Philadelphia: Siam, 2005. 358 p. 53, 54

THACKER, N. A.; COOTES, T. F. **Vision through optimization**. [s.n.], 1996. Notas de aula. Disponível em: <`http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/BMVA96Tut/BMVA96Tut.html`>. 6

TIKHONOV, A. N.; ARSENIN, V. Y. **Solutions of ill-posed problems**. New York: John Wiley & Sons, 1977. Traduzido por Fritz John. 58, 59, 60, 67, 76

TSALLIS, C. Possible generalization of boltzmann-gibbs statistics. **Journal of Statistical Physics**, v. 52, n. 1, p. 479–487, July 1988. 61

WEISE, T. **Global optimization algorithms: Theory and application**. 2009. E-book (pdf). 2nd Ed. Disponível em: <`http://www.it-weise.de/`>. 47

WIKIPEDIA. **Flynn's taxonomy**. 2010. Internet. Disponível em: http://en.wikipedia.org/wiki/FlynnFev. 2010. 15

WOLPERT, D. H.; MACREADY, W. G. No free lunch theorems for optimization. **IEEE Transactions on Evolutionary Computation**, v. 1, n. 1, p. 67–82, April 1997. 2

YANG, X.-S. **Nature-inspired metaheuristic algorithms**. Frome, UK: Luniver Press, 2008. 116 p. xvi, 36, 38, 39, 40

_____. **Engineering optimization: An introduction with metaheuristic applications**. Hoboken, NJ: John Wiley & Sons, 2010. 347 p. 7

_____. **Nature-inspired metaheuristic algorithms**. Second edition. Frome, UK: Luniver Press, 2010. 148 p. 38, 59

YUAN, B.; GALLAGHER, M. Statistical racing techniques for improved empirical evaluation of evolutionary algorithms. In: 8TH INTERNATIONAL CONFERENCE PARALLEL PROBLEM SOLVING FROM NATURE - PPSN VIII. **Proceedings...** Springer, 2004. p. 172–181. Disponível em: <`http://www.itee.uq.edu.au/~marcusg/papers/PPSNVIII-219.pdf`>. 49, 50

ZIVIANI, N. **Projeto de algoritmos: com implementações em PASCAL e C**. 2. ed. São Paulo: Pioneira Thomson Learning, 2004. 552 p. 29

ÖZISIK, M. N.; ORLANDE, H. R. B. **Inverse heat transfer: Fundamentals and applications**. New York, NY: Taylor & Francis, 2000. 330 p. 66

8 APÊNDICE A - ARTIGO EM REVISTA

Journal of Computational Interdisciplinary Sciences (2008) 1(1): 3-10

ISSN 1983-8409
http://epacis.org

A new multi-particle collision algorithm for optimization in a high performance environment

Eduardo Fávero Pacheco da Luz, José Carlos Becceneri and Haroldo Fraga de Campos Velho

Manuscript received on July 31, 2008 / accepted on October 5, 2008

ABSTRACT

A new meta-heuristics is introduced here: the Multi-Particle Collision Algorithm (M-PCA). The M-PCA is based on the implementation of a function optimization algorithm driven for a collision process of multiple particles. A parallel version for the M-PCA is also described. The complexity for PCA, M-PCA, and a parallel implementation for the MPCA is developed. The efficiency for optimization for PCA and M-PCA is evaluated for some test functions. The performance of the parallel implementation of the M-PCA is also presented. The results with M-PCA produced better optimized solutions for all test functions analyzed.

Keywords: Computational mathematics, computational complexity, high performance computing, meta-heuristics, optimization.

Associated Laboratory for Computing and Applied Mathematics, National Institute for Space Research, Av. Astronautas, 1.756, Jardim Granja, 12227-010 São José dos Campos, SP, Brazil. E-mails: [eduardo.luz, becce, haroldo]@lac.inpe.br

1 INTRODUCTION

Scientists have studied and developed new methods for resolving optimization problems [1]. Some of these methods are nature inspired techniques such as Simulated Annealing (SA) [13, 9], Genetic Algorithm (GA) [6, 5], Particle Swarm Optimization (PSO) [8] and Ant Colony System (ANS) [3] – as systems based on animal behavior, and Invasive Weed Optimization (IWO) [12] – from an emulation for the vegetal growing patterns. More recently, the canonical Particle Collision Algorithm (PCA) [17, 16] has been proposed.

The PCA implements the search for one single particle, responsible for a sequential search in the search space [17]. However, this kind of search can be improved by using many particles, dealing with in a collaborative way. The Multi-Particle Collision Algorithm (M-PCA) can better explore the search space, avoiding premature convergence to a local optimum. The M-PCA is straightforward for implementation in a parallel environment.

In the next section the PCA and M-PCA are described and the complexity for both approaches is discussed, and the algorithms are applied to several test functions. Section 3 presents the parallel implementation of the M-PCA, including the complexity for the parallel implementation, and performance results are shown.

The obtained results for the application of Multi-Particle Collision Algorithm (M-PCA) over some test functions reveals outstanding results, which together with the assurance of less computational efforts ensure the applicability of the proposed method.

2 DESCRIPTION OF THE PCA AND M-PCA SCHEMES

The theory of optimization is a branch of mathematical sciences that studies the methods for finding an optimum set for a given problem. The practical part of the theory is defined by the collection of techniques, methods, procedures and algorithms that can be used to find the optimum [1].

Optimization problems have the goal of finding the best set within a variable set to maximize or minimize a function, defined as an objective function or cost function. Optimization problems can be classified as: continuous optimization (where the variable has real or continuous values); discrete optimization (where the variable has integer or discrete values); and mixed optimization (with integer and continuous values at the same time) [2].

The best method for determining the function optimum strongly depends on the nature of the function in study. Two kinds of algorithms can be used: local optimization algorithms, that given a point in a function sub-domain are able to find the optimum point in this sub-domain (most of this algorithms are deterministic); and global optimization algorithms that seek the optimum point in the totality of the search space (those are stochastic algorithms, mostly metaheuristics).

Deterministic algorithms used in optimization can be divided into three main types:

- Zero order methods: based on the value of the objective function, e.g., Powell's conjugated directions;
- First order methods: based on the value of the objective function and its derivative regarding the project's variables, e.g., Steepest Descent;
- Second order methods: based on the value of the objective function, its derivative and the Hessian matrix, e.g., Quasi-Newton.

Stochastic methods used in optimization are those based on heuristics. Heuristics came from the Greek word *heuriskein*, meaning "to discover", and describing a method "based on the experience or judgment, that leads to a good solution of a problem, but not assuring to produce the optimum solution" [13]. Metaheuristics are those heuristics strongly based on natural processes.

The main metaheuristics are Simulated Annealing (SA), Genetic Algorithms (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), among others [9].

2.1 Particle Collision Algorithm (PCA)

The Particle Collision Algorithm (PCA) was developed by Sacco and co-authors [17, 16, 18, 19], inspired by Simulated Annealing [9].

This algorithm was also greatly inspired by two physical behaviors, namely absorption and scattering that occurs inside a nuclear reactor. The use of the PCA was effective for several test functions and real applications [19].

The PCA starts by selecting an initial solution (Old_Config), that is modified by a stochastic perturbation (Perturbation()), leading to the construction of a new solution (New_Config). The new solution is compared (by the function Fitness()), and the new solution can or cannot be accepted. The main algorithm can be seen in Figure 1.

If the new solution is not accepted, the Metropolis scheme is used (the function Scattering()). The exploration on closer positions is guaranteed by using the functions Perturbation() and Small_Perturbation(). The latter algorithms are shown by Figure 2.

```
Generate an initial solution Old_Config
Best_Fitness = Fitness(Old_Config)
For n = 0 to # of iterations
 Perturbation()
 If Fitness(New_Config) > Fitness(Old_Config)
   If Fitness(New_Config) > Best_Fitness
     Best_Fitness = Fitness(New_Config)
   End-If
   Old_Config = New_Config
   Exploration()
 Else
   Scaterring()
 End-If
End-For

Exploration()
 For n = 0 to # of iterations
 Small_Perturbation()
 If Fitness(New_Config) > Fitness(Old_Config)
   If Fitness(New_Config) > Best_Fitness
     Best_Fitness = Fitness(New_Config)
   End-If
   Old_Config = New_Config
 End-If
Return

Scattering()
 p_scattering = 1 - (Fitness(New_Config) / Best_Fitness)
 If p_scattering > random(0,1)
   Old_Config = Random Solution
 Else
   Exploration()
 End-If
Return
```

Figure 1 – Main PCA pseudo-code.

```
Perturbation()
  For i = 0 to (Dimension-1)
    Upper = Superior_Limit[i]
    Lower = Inferior_Limit[i]
    Rand = Random(0,1)
    New_Config[i] = Old_Config[i] + ((Upper - Old_Config[i]) *
    Rand) - ((Old_Config[i] - Lower)*(1-Rand))
    If (New_Config[i] > Upper)
      New_Config[i] = Superior_Limit[i]
    Else
      If (New_Config[i] < Lower)
        New_Config[i] = Inferior_Limit[i]
      End-If
    End-If
  End-For
Return

Small_Perturbation()
  For i = 0 to (Dimension-1)
    Upper = Random(1.0, 1.2) * Old_Config[i]
    If (Upper > Superior_Limit[i])
      Upper = Superior_Limit[i]
    End-If
    Lower = Random(0.8, 1.0) * Old_Config[i]
    If (Lower < Inferior_Limit[i])
      Lower = Inferior_Limit[i]
    End-If
    Rand = Random(0,1)
    New_Config[i] = Old_Config[i] + ((Upper - Old_Config[i]) *
    Rand) - ((Old_Config[i] - Lower)*(1-Rand))
  End-For
Return
```

Figure 2 – Codes for stochastic perturbation in PCA.

If a new solution is better than the previous one, this new solution is absorbed — absorption is one feature involved in the collision process. If a worse solution is found, the particle can be send to a different location of the search space, giving the algorithm the capability of escaping a local minimum, this procedure is inspired by the scattering scheme.

PCA is a robust metaheuristics, only one parameter is required from the user. This parameter is the number of iterations, and it also acts as a stop criteria for the algorithm. According to [18, 19] 10^5 iterations should be a good value for almost every application.

2.2 Complexity of PCA

Analyzing the main algorithm of PCA, presented at Figure 1, and adopting N as the number of iterations defined by the user, we can see that the first loop induces N checking operations over the objective function. At this main loop, there are calls to the Exploration() procedure, that by its turn executes another N operations through a loop inside the procedure. Therefore, $N \times N$ operations are executed by PCA, leading to a $O(N^2)$ complexity.

2.3 Multi-Particle Collision Algorithm (M-PCA)

The new Multi-Particle Algorithm (M-PCA) is based on the canonical PCA, but a new characteristic is introduced: the use of several particles, instead of only one particle to act over the search space.

Coordination between the particles was achieved through a blackboard strategy, where the Best_Fitness information is shared among all the particles in the process.

The pseudo-code for the M-PCA is presented by Figure 3, where the new loop, responsible for the control of the new particles is introduced.

Similar to the PCA, M-PCA also has only one parameter to be determined, the number of iterations. But in this case, the total number or iterations is divided by the number of particles which will be used in the process. The division of the task is the great distinction of M-PCA, which leads to a great reduction of required computing time.

The M-PCA was implemented using MPI libraries in a multiprocessor architecture with distributed memory machine.

```
Generate an initial solution Old_Config
Best_Fitness = Fitness(Old_Config)
Update Blackboard
For n=0 to # of particles
  For n=0 to # of iterations
    Update Blackboard
    Perturbation()
    If Fitness(New_Config) > Fitness(Old_Config)
      If Fitness(New_Config) > Best_Fitness
        Best_Fitness = Fitness(New_Config)
      End-If
      Old_Config = New_Config
      Exploration()
    Else
      Scattering()
    End-If
  End-For
End-For
```

Figure 3 – The new M-PCA and the loop for particle control.

2.4 Complexity of M-PCA

The code of M-PCA, presented in Figure 3, is very similar to PCA, so there are $N \times N$ checking operations in the inner loops, but due to the new loop, introduced by the multiple particle technique, that number of checking operations can be increased to a case where $N \times N \times N$ operations can occur, e.g., the number of particles is equal to the number of iterations.

So, the complexity associated to M-PCA is initially $O(N^3)$, but when we distribute the algorithm through the use of p processors, making sure that $p = n$, n as the number of particles in use, the complexity can return to $O(N^2)$, which is the same complexity of the canonical PCA.

3 HIGH PERFORMANCE ENVIRONMENTS

High performance environments are viable through the use of High Performance Computing (HPC) term that reflects the use of supercomputers or computer clusters to induce the parallel processing.

The process of computing can be defined as the execution of a sequence of instructions in a data set that can be improved by the use of a high performance environment. At this point, the computing sequence in a parallel environment consisted of one or more tasks executed in a concurrent way.

Computational systems that make a high performance environment viable are actually capable of providing teraflops, or 10^{12} floating point operations per second, to the user, enabling the solution of problems in high dimensions through new algorithms adapted to operate in this environment, as proposed in this work.

Flynn's taxonomy [4], is used to classify computers and programs capable of high speed operations in high performance environments, in the following way:

- Single Instruction – Single Data (SISD);
- Single Instruction – Multiple Data (SIMD);
- Multiple Instruction – Single Data (MISD);
- Multiple Instruction – Multiple Data (MIMD).

However, a new classification can be used, based on the analysis of hardware and software solutions adopted [20]. In this simpler scheme, only the main characteristics of each architecture are analyzed.

For the classification based on hardware, the following scheme is possible:

- Vectorial architecture: the same instruction is executed synchronously by every processor, but in distinct data sets;
- Multiprocessor architecture with shared memory: machines with shared memory, usually with a few number of processors that have access to a global memory area, where the communication between the processors occurs;
- Multiprocessor architecture with distributed memory: a set of processors and memory are connected through a communication network. The access to the remote data is done through message passing and the processors execute different sets of instructions in different moments. The synchronism is obtained only through the use of mechanisms provided by the programming language;
- Hybrid system architecture: assumes the existence of several multiprocessed nodes linked by networks similar to a distributed system architecture.

For the analysis over the software solution, the possibilities includes implementation via:

- High Performance Fortran (HPF): an explicit parallel language. Extension of Fortran 90 with the incorporation of directives for making data parallelism viable;
- OpenMP: a set of programming directives and libraries to parallel application programming, based in a shared memory model using explicit parallelism;
- Parallel Virtual Machine (PVM): a package of integrated libraries and software tools that allows serial, parallel and vectorial computers to be connected into a same network, working as a single computational resource;
- Message Passing Interface (MPI), a message trade library used in a shared memory environment, viable through explicit calls to the communication routines, where every parallelism is explicit.

Besides the classification presented, grid computing techniques, where the computational resources are scattered in a large geographical area, can be classified as a computing system solution, i.e., a software platform that manages the data traffic in a safe way, and administrates the execution of designed tasks.

To ensure the efficiency of a parallel algorithm, we need to perform some verification regarding the efficiency of the proposed algorithm.

Two related analyses can be done, the first one will check the speedup rate of the parallel algorithm in comparison to its serial version. The speedup analysis is defined, according to [20], as the ratio of the execution time of the serial algorithm (T_s) over the execution time of the parallel algorithm (T_p):

$$S_p = \frac{T_s}{T_p} \tag{1}$$

The second analysis, related to the efficiency of the algorithm, is defined as the ratio of speedup (S_p) over the number of used processors (p), trying to check how much the parallelism was explored by the algorithm:

$$E_p = \frac{S_p}{p} \leq 1 \tag{2}$$

Where, to ensure the efficiency of the proposed algorithm, the value of Equation 2 must be less or equal to one, as stated in [20].

3.1 Parallel implementation

A preliminary study of the M-PCA code has led to the chosen parallelization strategy.

Immediately some parallel strategies can be thought: (a) Coarse Grained: where the particles are divided on several clusters of particles, some particles can migrate (this is a new operator) from one cluster to another; (b) Fine Grained: such approach requires a large number of processors because the particles are addressed into a large number of particle clusters, and at each cluster a search strategy is applied, but subject to migration; (c) Global Parallelization: in this approach all search operators and the evaluation of all individuals are explicitly parallelized.

Strategy (a) admits several schemes: insulated clusters (particle migration could be implemented in several ways: particles cross from one cluster to other, or particles from one cluster could be send and/or received between the master cluster and others), cluster ring (particles travel only to the closest cluster, this may also be named as stepping-stone strategy). It is not clear which parallelization scheme is more effective for strategy (a). For a huge amount of particles, the strategy (b) could be applied in massively parallel machines. However, considering few particles (less than 100, for example) this strategy is a good option for many HPC systems, and its implementation is straightforward. For parallelizing the computation of the objective function for many particles to different processors is not a difficult challenge, but the parallelization of other aspects from the PCA is not clear.

Due to the small number of particles adopted for the present implementation of M-PCA, the parallel strategy implemented is the procedure (b).

The related code was parallelized using calls to the message passing communication library MPI [14] and executed on a distributed memory parallel machine. This machine is a HP XC cluster of 112 standard AMD-Opteron 2.2 GHz scalar architecture processors connected by a InfiniBand interconnection.

The prevailing trend in the search for high performance is the use of parallel machines due to their good cost effectiveness. Three parallel architectures are usually considered: vector processors, multiprocessors with shared memory, and distributed memory machines. In the multiprocessors shared class, all processors access a unique memory address space and there are scalability constraints. In the latter class, off-the shelf machines called nodes are interconnected by a network composing a cluster or Massive Parallel Processors (MPP), a parallel machine with hundreds or thousands of nodes using a very fast interconnection scheme. The processors of each node access only their local memories and data dependencies between node memories enforce communication by means of routines of a message passing library, like Message Passing Interface (MPI) or Parallel Virtual Machine (PVM). Data dependencies between processors require calls to the MPI library. On the other hand, a single processor/node may perform tasks like gathering partial results obtained by every node and broadcasting the global result to all other nodes, also by means of MPI calls.

An important issue is to maximize the amount of computation done by each processor and to minimize the amount of communication due to the MPI calls in order to achieve good performance. The speedup is defined as the ratio between the sequential and parallel execution times. A linear speedup denotes that processing time was decreased by a factor of p when using p processors and can be thought as a kind of nominal limit. Efficiency is defined as the ratio of the speedup by p and thus it is 1 for a linear speedup. Usually, communication penalties lead to efficiencies lesser than 1. Exceptionally, as data is partitioned among processors, cache memory access can be optimized in such way that superlinear speedup can be attained, i.e. efficiencies greater than 1.

4 TEST RESULTS

The PCA was set to execute 10^5 iterations, and the M-PCA was set to operate with 10 particles, each one executing 10^4 iterations. The test functions used were Easom, Rosenbrock and Griewank

(for minimization) and Shekel (for maximization). The optimum results for these functions are presented in Table 1.

Table 1 – Optimum results for tested functions.

Function	x^*	$f(x^*)$
Easom	(π, π)	–1
Shekel	(–32, –32)	499.002
Rosenbrock	(1, 1)	0
Griewank	(0, 0)	0

The comparative results for the Easom function can be seen in Table 2, where the great advantage of M-PCA over PCA is clear. Even if the results are similar, the time of execution demonstrates a great advantage for M-PCA. The global minimum for this function is located at $x^* = (\pi, \pi)$ with $f(x^*) = -1$.

Table 2 – Easom function results.

	PCA	M-PCA
Time	19.37s	4.29s
x^*	(3.14, 3.14)	(3.14, 3.14)
$f(x^*)$	–1.00	–1.00

Table 3 shows the results for the Shekel function, now a maximization function. Once again, both algorithms reached good results, but the time needed for M-PCA execution is significantly lower. The global minimum for this function is located at $x^* = (–32, –32)$ with $f(x^*) = 499.002$.

Table 3 – Shekel function results.

	PCA	M-PCA
Time	3099.53s	113.56s
x^*	(–32.0, –32.0)	(–32.0, –32.0)
$f(x^*)$	499.002	499.002

For the Rosenbrock function, the results are seen on Table 4. And once more, the execution time for M-PCA reveals its advantages. The global minimum for this function is located at $x^* = (1, 1)$ with $f(x^*) = 0.0$.

Table 4 – Rosenbrock function results.

	PCA	M-PCA
Time	1492.23s	21.21s
x^*	(1.00, 1.00)	(1.00, 1.00)
$f(x^*)$	0.0	0.0

And finally the results for the Griewank function are seen on Table 5, where the M-PCA surpasses its canonical version, both in time and accuracy. The global minimum for this function is located at $x^* = (0, 0)$ with $f(x^*) = 0$, and only M-PCA was able to find it with great results.

Table 5 – Griewank function results.

	PCA	M-PCA
Time	2392.05s	32.03s
x^*	(–3.14, 4.43)	$(–1.82 \times 10^{-8}, –3.25 \times 10^{-8})$
$f(x^*)$	7.39×10^{-3}	3.33×10^{-16}

A speedup and efficiency analysis considering the Easom function for M-PCA results given by Table 2 gives a speedup rate of $S_p = 4.447$ and a efficiency around $E_p = 0.4447$ for the given parameters.

The M-PCA speedup curve for the Easom function solved with 2, 5 and 10 processors is given by Figure 4.

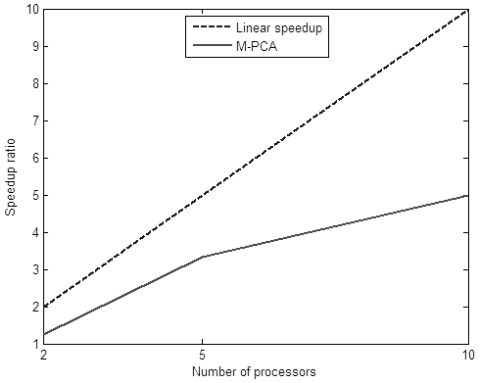


Figure 4 – Speedup curve for M-PCA with 2, 5 and 10 processors.

The speedup curve of M-PCA is bellow the desired curve (for a linear speedup) due to the high amount of communication between the processors, triggered by the update of Best_Fitness information in the blackboard.

A way to improve the performance of M-PCA is to update the Best_Fitness after a determined number of cycles of iterations. Such a feature will decrease the communication messages among processors, resulting in a better performance for the parallel M-PCA.

In preliminary tests, the application of a 1000 iteration communication cycle, i.e. the execution of 1000 iterations until the implementation of communication amongst processors, leads

to an speedup and efficiency improvement, $S_p = 6.933$ and $E_p = 0.6933$, as we can see in Figure 5.

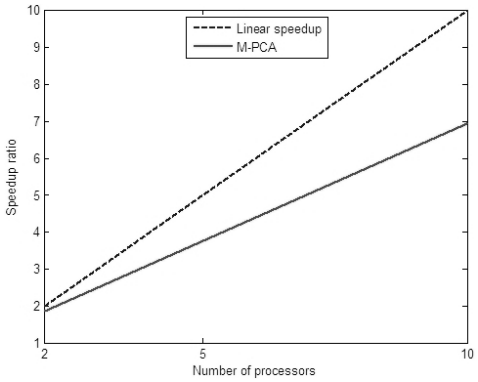


Figure 5 – Speedup curve for M-PCA with 2, 5 and 10 processors and 1000 iterations for communication cycles.

So, the M-PCA proves its advantages over the canonical version, particularly: under the Griewank test function, when analyzing its accuracy; and over all the test functions, when analyzing its execution time.

5 CONCLUSIONS

A multi-particle collision algorithm was developed, and its complexity analysis was carried out ($O(N^2)$ order complexity in a parallel implementation). Results applying the PCA and M-PCA considering several test functions were presented. The M-PCA produced better results (computational time) in all cases analyzed. Parallel versions for the M-PCA were discussed, and a global parallel approach was implemented using MPI instructions. Performance for the parallel M-PCA was also presented.

Optimization algorithms are used for a large range of applications. A very important application for the optimization process is to solve inverse problems. Bio-geo-chemical cycles for greenhouse effect gases (such as methane and carbon dioxide) can be identified using regularized inverse solutions [7], employing an implicit approach. There is work on techniques to improve inverse solutions obtained using quasi-Newton deterministic optimization [15] and PSO stochastic optimization [10, 11], for overcoming local minima and also to compute better inverse solutions. In the employed implicit methodology, the inverse problem is formulated as an optimization problem which minimizes an objective function. For each candidate solution, the value of this function is given by the square difference between experimental values and the data given by the direct model [15]. A typical estimation may demand thousands of iterations and, therefore, optimization performance is an important issue. Therefore, parallel implementation for solving implicit inverse problems is a very relevant issue to improve the performance of the inversion process.

ACKNOWLEDGMENTS

The authors wish to thank Dr. W.F. Sacco (UERJ) for the discussion about the PCA. Special thanks go to the C-PAD team at INPE/SJC. This work was partially supported by CAPES.

REFERENCES

[1] ANTONIOU A & LU W-S. 2007. Practical optimization: algorithms and engineering applications. Springer, New York.

[2] BECCENERI JC. 2008. Chapter Meta-Heurísticas e Otimização Combinatória: Aplicações em Problemas Ambientais. INPE, São José dos Campos.

[3] DORIGO M & STUTZLE T. 2004. Ant Colony Optimization. The MIT Press, Cambridge.

[4] FLYNN MJ. 1966. High-speed computing systems. Proceedings of the IEEE, 54(12): 1901–1909.

[5] GOLDBERG DE. 1989. Genetic Algorithms in search optimization and Machine Learning. Addison-Wesley, Boston, MA, USA.

[6] HOLLAND JH. 1992. Adaptation in natural and artificial systems. MIT Press, Cambridge, MA, USA.

[7] KASIBHATLA P, HEIMANN M, RAYNER P, MAHOWALD N, PRINN RG & HARTLEY DE (Eds.). 2000. Inverse Methods in Global Biogeochemical Cycles. American Geophysical Union, Washington, USA.

[8] KENNEDY J & EBERHART EC. 1995. Particle swarm optimization. IEEE Int. Conf. Neural Networks, 4: 1942–1948.

[9] KIRKPATRIK S, GELATT CD & Vecchi MP. 1983. Optimization by simulated annealing. Science, 220: 671–680.

[10] LUZ EFP. 2007. Estimação de fonte de poluição atmosférica usando otimização por enxame de partículas. Master's thesis, Computação Aplicada, INPE, São José dos Campos.

[11] LUZ EFP, VELHO HFC, BECCENERI JC & ROBERTI DR. 2007. Estimating atmospheric area source strength through particle swarm optimization. Florida: Proceedings of IPDO.

[12] MEHRABIAN AR & LUCAS C. 2006. A novel numerical optimization algorithm inspired from weed colonization. Ecological Informatics 1: 355–366.

[13] METROPOLIS N, ROSENBLUTH AW, ROSENBLUTH MN, TELLER AH & TELLER E. 1953. Equation of state calculations by fast computing machines. Journal of Chemical Physics, 21: 1087–1092.

[14] PACHECO P. 1996. Parallel programming with MPI. Morgan Kaufmann Publishers, San Francisco, USA.

[15] ROBERTI DR, ANFOSSI A, VELHO HFC & DEGRAZIA GA. 2005. Estimation of emission rate of pollutant armospheric source. Proceeding of ICIPE 2005, 3: R03-1–R03-8.

[16] SACCO WF & DE OLIVEIRA CRE. 2005. A new stochastic optimization algorithm based on a particle collision metaheuristic. Proceedings of 6th WCSMO.

[17] SACCO WF, FILHO HA & PEREIRA CMNA. 2007. Cost-based optimization of a nuclear reactor core design: a preliminary model. Proceedings of INAC.

[18] SACCO WF, LAPA CMF, PEREIRA CMNA & FILHO HAA. 2006. Two stochastic optimization algorithms applied to nuclear reactor core design. Progress in Nuclear Energy, 525–539.

[19] SACCO WF, LAPA CMF, PEREIRA CMNA & FILHO HAA. 2008. A metropolis algorithm applied to a nuclear power plant auxiliary feedwater system surveillance tests policy optimization. Progress in Nuclear Energy, 15–21.

[20] SAMBATTI SBM. 2004. Diferentes estratégias de paralelização de um algoritmo genético epidêmico aplicadas na solução de problemas inversos. Master's thesis. Computação Aplicada, INPE, São José dos Campos.

9 APÊNDICE B - ARTIGO EM REUNIÃO CIENTÍFICA

2011 IEEE International Parallel & Distributed Processing Symposium

Multiple Particle Collision Algorithm applied to Radiative Transference and Pollutant Localization Inverse Problems

Eduardo Fávero Pacheco da Luz
Post-graduation program in Applied Computing
National Institute for Space Research
São José dos Campos/SP - Brazil
eduardo.luz@lac.inpe.br

José Carlos Becceneri, Haroldo Fraga de Campos Velho
Computing and Applied Mathematics Associated Laboratory
National Institute for Space Research
São José dos Campos/SP - Brazil
{*becce, haroldo*}*@lac.inpe.br*

***Abstract*—The Multiple Particle Collision Algorithm (MPCA) is a nature-inspired stochastic optimization method developed specially for high performance computational environments. Its advantages resides in the intense use of computational power provided by multiple processors in the task of search the solution space for a near optimum solution. This work presents the application of MPCA in solving two inverse problems written as optimization problems, its advantages and disadvantages are also described, so are the obtained results.**

***Keywords*-Metaheuristic, Optimization, High Performance Computing, Inverse Problems.**

I. Introduction

The theory of optimization is a branch of the mathematical sciences that studies the methods for finding an optimum set for a given problem. The practical part of the theory is defined by the collection of techniques, methods, procedures and algorithms that can be used to find the optimum [1].

Optimization problems has the goal of finding the best set within a variable set to maximize or minimize a function, defined as an objective function or cost function. Optimization problems can be classified as [2]:

- Continuous optimization: where variable has real or continuous values;
- Discrete optimization: where variable has integer or discrete values; and
- Mixed optimization: with integer and continuous values at the same time.

The best method for determining the function optimum, strongly depends on the nature of the function in study. Two kinds of algorithms can be used: local optimization algorithms, that given a point in a function sub domain are able to find the optimum point in this sub domain (most of this kind of algorithms are deterministic); and global optimization algorithms, that seek the optimum point in the totality of the search space (those are frequently stochastic algorithms, mostly metaheuristics).

Deterministic algorithms used in optimization can be divided into three main types:

- Zero order methods: based on the value of the objective function, e.g., Powell's conjugated directions;
- First order methods: based on the value of the objective function and its derivative regarding the project's variables, e.g., Steepest Descent;
- Second order methods: based on the value of the objective function, its derivative and the Hessian matrix, e.g., Quasi-Newton.

Stochastic methods used in optimization are those based on heuristics. The word "heuristic" came from the Greek *heuriskein*, meaning "to discover", and describing a method "based on the experience or judgement, that leads to a good solution of a problem, but not assuring to produce the optimum solution" [14]. Metaheuristics can be described as those heuristics strongly based on natural processes.

The main metaheuristics are Simulated Annealing (SA), Genetic Algorithms (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), among others [10].

II. Particle Collision Algorithm (PCA)

The Particle Collision Algorithm (PCA) was developed by Sacco and co-authors [19], [18], [20], [21], inspired in some basic characteristics of Simulated Annealing [10].

This algorithm was also greatly inspired by two physical behaviours, namely absorption and scattering, that occurs inside a nuclear reactor. The use of the PCA was effective for several test functions and real applications [21].

The PCA starts by selecting an initial solution (`Old_Config`), that is modified by a stochastic perturbation (`Perturbation()`), leading to the construction of a new solution (`New_Config`). The new solution is compared to the old one (the solutions are compared by calculating the fitness of each one with function `Fitness()`), and the new solution can or cannot be accepted. The main algorithm can be seen in Figure 1.

If the new solution is not accepted, a Metropolis scheme is used (the function `Scattering()`). The exploration on closer positions is guaranteed by using the functions `Perturbation()` and `Small_Perturbation()`. The latter algorithms are shown by Figure 2.

1530-2075/11 $26.00 © 2011 IEEE
DOI 10.1109/IPDPS.2011.171

```
Generate an initial solution Old_Config
Best_Fitness = Fitness(Old_Config)
For n = 0 to # of iterations
 Perturbation()
 If Fitness(New_Config) > Fitness(Old_Config)
    If Fitness(New_Config) > Best_Fitness
       Best_Fitness = Fitness(New_Config)
    End-If
    Old_Config = New_Config
    Exploration()
 Else
    Scaterring()
 End-If
End-For

Exploration()
 For n = 0 to # of iterations
 Small_Perturbation()
 If Fitness(New_Config) > Fitness(Old_Config)
    If Fitness(New_Config) > Best_Fitness
       Best_Fitness = Fitness(New_Config)
    End-If
    Old_Config = New_Config
 End-If
Return

Scattering()
 P_Scattering = 1 - (Fitness(New_Config) / Best_Fitness)
 If P_Scattering > random(0,1)
    Old_Config = Random Solution
 Else
    Exploration()
 End-If
Return
```

Figure 1. Main PCA pseudo-code

```
Perturbation()
   For i = 0 to (Dimension-1)
      Upper = Superior_Limit[i]
      Lower = Inferior_Limit[i]
      Rand = Random(0,1)
      New_Config[i] = Old_Config[i] + ((Upper - Old_Config[i]) *
      Rand) - ((Old_Config[i] - Lower)*(1-Rand))
      If (New_Config[i] > Upper)
         New_Config[i] = Superior_Limit[i]
      Else
         If (New_Config[i] < Lower)
            New_Config[i] = Inferior_Limit[i]
         End-If
      End-If
   End-For
Return

Small_Perturbation()
   For i = 0 to (Dimension-1)
      Upper = Random(1.0, 1.2) * Old_Config[i]
      If (Upper > Superior Limit[i])
         Upper = Superior_Limit[i]
      End-If
      Lower = Random(0.8, 1.0) * Old_Config[i]
      If (Lower < Inferior_Limit[i])
         Lower = Inferior_Limit[i]
      End-If
      Rand = Random(0,1)
      New_Config[i] = Old_Config[i] + ((Upper - Old_Config[i]) *
      Rand) - ((Old_Config[i] - Lower)*(1-Rand))
   End-For
Return
```

Figure 2. Codes for stochastic perturbation in PCA

If a new solution is better than the previous one, this new solution is absorbed (absorption is one feature involved in the real collision process). If a worst solution is found, the particle can be send to a different location of the search space, giving the algorithm the capability of escaping a local minima, this procedure is inspired on the scattering scheme.

PCA is a robust metaheuristic, only a few parameters are required from the user. The main parameter is the number of iterations, and also acts as a stop criteria for the algorithm. According to [20], [21] 10^5 iterations should be a good value for almost every application.

Analysing the main algorithm of PCA, presented at Figure 1, and adopting N as the number of iterations defined by user, we can see that the first loop induces N checking operations over the objective function. At this main loop, there are calls to the `Exploration()` procedure, that by its turn executes more N operations through a loop inside the procedure. Therefore, $N \times N$ operations are executed by PCA, leading to a $O(N^2)$ complexity.

III. Multiple Particle Collision Algorithm (MPCA)

The new Multiple Particle Algorithm (MPCA) is based on the canonical PCA, but a new characteristic is introduced: the use of several particles, instead of only one particle to act over the search space.

Coordination between the particles was able through a blackboard strategy, where the `Best_Fitness` information is shared among all the particles in the process.

The pseudo-code for the MPCA is presented by Figure 3, where the new loop, responsible for the control of the new particles is introduced.

```
Generate an initial solution Old_Config
Best_Fitness = Fitness(Old_Config)
Update Blackboard
For n=0 to # of particles
   For n=0 to # of iterations
      Update Blackboard
      Perturbation()
      If Fitness(New_Config) > Fitness(Old_Config)
         If Fitness(New_Config) > Best_Fitness
            Best_Fitness = Fitness(New_Config)
         End-If
         Old_Config = New_Config
         Exploration()
      Else
         Scattering()
      End-If
   End-For
End-For
```

Figure 3. The new MPCA and the loop for particle control

Similar to the PCA, MPCA also have only few parameters to be determined, added to the number of particles to be used. But in this case, the total number or iterations is divided by the number of particles which will be used in the process. The division of task is the great distinction of MPCA, which leads to a great reduction of required computing time.

The MPCA was implemented using MPI libraries in a multiprocessor architecture with distributed memory.

The code of MPCA, presented by Figure 3, is very similar to PCA, so there are $N \times N$ checking operations in the inner loops, but due to the new loop, introduced by the multiple particle technique, that number of checking operations can be increased to a case where $N \times N \times N$ operations can occur, e.g., the number of particles is equal to the number of iterations.

So, the complexity associated to MPCA is initially $O(N^3)$, but when we distribute the algorithm through the use of p processors, making sure that $p = n$, n as the number of particles in use, the complexity can return to $O(N^2)$, which is the same complexity of the canonical PCA.

IV. Inverse problems

While dealing with the mathematical formulation of a direct problem we start with from an initial situation and came, by the application of some methodology, calculation or equation, to the effects generated by the modelled phenomenon in the direct problem.

To solve an inverse problem is to determine unknown causes taking in consideration desired or observed effects [4]. This is one of the most used definitions, leading to the conclusion that studying inverse problems is consisted of the use of observed results to estimate some parameter values that can characterize the system under investigation.

In this way, if we admit that a given mathematical model can be expressed as $A(u) = f$, the inverse model related to this problem can be expressed as $A^{-1}(f) = u$. At Fig. 4 the graphical representation of a direct problem is given by the path that connects the space of causes to the space of effects and the inverse problem connects the space of effects to the space of causes.

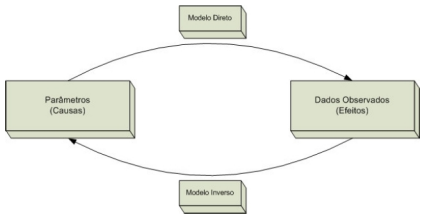


Figure 4. Direct and inverse problems connecting the cause and effect spaces.

Most of the inverse problems belongs to an ill-posed class of problems, i.e., those kind of problems that violates one or more Hadamard condition, which are:

- Existence: the problem must have a solution;
- Uniqueness: the solution must be unique;
- Stability: the dependency between data must be continuous.

Breaking the third condition, stability, leads to situation that can be classified as "ill-conditioning" that can be solved with the use of regularization. More information about the theory of regularization can be obtained in [24].

Now, we are going to present the inverse problems used in this paper in order to validate the MPCA.

A. Radiative transference

The first problem that we present is an inverse problem for radiative transference in a parallel-plane homogeneous media (Fig. 5). More information about the formulation of this problem can be obtained in [23].

The parameters to be estimated are:

$$\vec{Z} = \{\tau_0, \omega, \rho_1, \rho_2\}^T \tag{1}$$

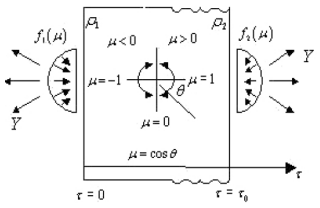


Figure 5. Schematic description of the radiative transference problem subjected to external radiance of intensities f_1 e f_2.

where τ_0 represents the optical thickness, ω represents the scattering albedo and ρ_1 e ρ_2 represents the diffuse reflectivity.

The optimization problem in this case is written as the minimization of square errors.

$$Q(\vec{Z}) = \sum_{i=1}^{N_d} \left[I_i(\vec{Z}) - Y_i \right]^2 \tag{2}$$

where I_i corresponds to the value calculated by the model for the candidate solution and Y_i represents the experimentally measured value.

This is called an implicit formulation, for the parameters does not appear directly on the formulation of the solution, but are included in the formulation of the correspondent direct problem, and its effects are perceived when the direct problem is solved and then participate in the composition of the objective function [23].

B. Localization of polluting sources

To represent the particle dispersion in the atmosphere, in this second problem, we adopted a Lagrangian Model for Buoyant Dispersion in Atmosphere (LAMBDA), which is based in the three-dimensional form of the Langevin equation for a random velocity field, according to Thomson's derivation [16].

The model for particle dispersion (LAMBDA) simulates a certain quantity of computational particles that emulates the behaviour of real particles for a generic atmospheric contaminant (Fig. 6).

The temporal integration for the Lagrangian dispersion calculates the mean concentration of a contaminant at the position $\vec{x}$ at time t, giving a certain emission rate in a source ($S(kgm^{-3}s^{-1})$), being defined as:

$$C(x,t) = \int_{-\infty}^{t} \int_{-\infty}^{\infty} S\left(\vec{x_0}, t_0\right) P^a\left(\vec{x}, t | \vec{x_0}, t_0\right) d\vec{x_0} dt_0 \tag{3}$$

where $P^a(\vec{x}, t|\vec{x_0}, t_0)$ is the time forward probability density, defined such as $P^a(\vec{x}, t|\vec{x_0}, t_0)d\vec{x_0}$ be the probability that an

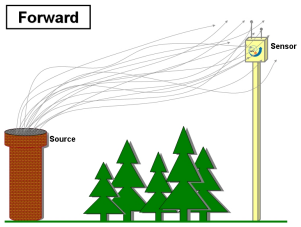


Figure 6. Lagrangian particle dispersion: particles are emitted from a polluting source and are detected by a sensor. The information from the sensor is used to locate the source and estimate its strength.

element of the fluid initially in $(\vec{x_0}, t_0)$ be found in time t and volume $d\vec{x}$ centred in $\vec{x}$.

The LAMBDA model implements the calculation of concentration in a given sensor as:

$$C_j = \sum_{i=1}^{N_f} S_i \frac{V_{f,i}}{V_{s,j}} \frac{\Delta t}{N_{PEF,i}} N_{PVS,i,j} \quad (4)$$

where C_j represents the concentration of the j-th sensor, N_f the number of sources, S_i the intensity of the i-th source, $V_{f,i}$ the volume of the i-th source, $V_{s,j}$ the volume of the j-th sensor, Δt the temporal discretization, $N_{PEF,i}$ the number of emitted particles at the i-th source and $N_{PVS,i,j}$ is the number of emitted particles at the i-th source that can be found in the j-th sensor.

To solve this problem as an optimization problem, we use a source-receptor model, that reduced the computational effort required by the iterative resolution of the direct model. Therefore, we calculate:

$$\vec{C} = M\vec{S}, \quad (5)$$

where $\vec{C}$ is a vector of elements that represents the mean concentration in the sensors, M represents the state transition matrix, e $\vec{S}$ represents the intensity at the emission sources or absorption points. The M matrix is built based on the Eq. 4, defined as:

$$M_{ij} = \frac{V_{f,i}}{V_{s,j}} \frac{\Delta t}{N_{PEF,i}} N_{PVS,i,j} \quad (6)$$

V. Results

The results presented in this section considers the mean of 10 experiments using distinct random number generation seeds and experimental data artificially generated by the use of the results from the direct model added by Gaussian noise, intending to simulate the use of real instrumentation. The adopted noise level was of 2% seeking o demonstrate the viability of MPCA in the solution of inverse problems.

The used parameter was: 8 particles; 8 processors; 10000 iterations; 1000 local search steps; Gaussian perturbation radius $N(0, 1)$; local search radius corresponding to 20% of the Gaussian perturbation radius. The algorithm was executed in a Cray XT5.

Table I presents the results for the radiative transference inverse problem. We can notice that the parameter ρ_1 had the worst estimative, but this was an expected result [23].

Table I
Results for the application of MPCA in the radiative transfer inverse problem.

	τ_0	ω	ρ_1	ρ_2
Exact result	$1,0000$	$0,5000$	$0,1000$	$0,9500$
Mean result	$0,9954$	$0,5144$	$0,1620$	$0,9540$
Standard deviation	$2,51\text{E}{-2}$	$2,42\text{E}{-2}$	$8,09\text{E}{-2}$	$2,09\text{E}{-3}$

Tables II and III presents the results for the localization of two emission/absorption polluting sources. The sources alternates its behaviour (rate) of emission to absorption and this is properly captured by this optimization algorithm.

Table II
Results for the application of MPCA in the inverse problem for locating and estimating the emission/absorption rate of the first area.

	Area 1 - Rate 1	Area 1 - Rate 2
Exact result	$0,90$	$-0,40$
Mean result	$0,8875$	$-0,4114$
Standard deviation	$0,1163$	$1,18\text{E}{-2}$

Table III
Results for the application of MPCA in the inverse problem for locating and estimating the emission/absorption rate of the second area.

	Area 2 - Rate 1	Area 2 - Rate 2
Exact result	$0,50$	$-0,80$
Mean result	$0,5127$	$-0,8820$
Standard deviation	$1,96\text{E}{-2}$	$6,59\text{E}{-2}$

VI. Conclusion

The MPCA showed itself a viable alternative to the solution of inverse problems, specially when there is high performance computers available.

The actual existence of personal computers with multicore architectures also enables the execution of an algorithm developed for high performance environments without the loss of performance, giving the user a better use of the available resources.

The results also demonstrates the convergence of MPCA for a good solution within a reasonable amount of available resources and opens the possibility of further studies, including hybridization, e.g., substitution of the actual local search method which is solely based in the generation of random neighbour solution within a given radius.

Acknowledgment

The authors would like to thank CAPES that partially funded this work.

REFERENCES

[1] A. Antoniou and W.-S. Lu. *Practical optimization: algorithms and engineering applications*. Springer, New York, 2007.

[2] J. C. Becceneri., Meta-Heuristicas e Otimizacao Combinatoria: Aplicacoes em Problemas Ambientais. INPE, Sao Jose dos Campos, 2008.

[3] M. Dorigo and T. Stutzle. *Ant Colony Optimization*. The MIT Press, Cambridge, 2004.

[4] Engl, H. W., Hanke, M., and Newbauer, A. (1996). *Regularization of inverse problems*. Kluwer, Dordrecht.

[5] M. J. Flynn. High-speed computing systems. *Proceedins of the IEEE*, 54(12):1901–1909, 1966.

[6] D. E. Goldberg. *Genetic Algorithms in search optimization and Machine Learning*. Addison-Wesley, 1989.

[7] J. H. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, 1975.

[8] P. Kasibhatla, M. Heimann, P. Rayner, N. Mahowald, R. G. Prinn, and D. E. H. (Editors). *Inverse Methods in Global Biogeochemical Cycles*. American Geophysical Union, Washington, USA, 2000.

[9] J. Kennedy and E. C. Eberhart. Particle swarm optimization. *IEEE Int. Conf. Neural Networks*, 4:1942–1948, 1995.

[10] S. Kirkpatrik, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[11] E. F. P. Luz. Estimacao de fonte de poluicao atmosferica usando otimizacao por enxame de particulas. Master's thesis, Computacao Aplicada, INPE, Sao Jose dos Campos, 2007.

[12] E. F. P. Luz, H. F. C. Velho, J. C. Becceneri, and D. R. Roberti. Estimating atmospheric area source strength through particle swarm optimization. *Florida: Proceddings of IPDO*, 2007.

[13] A. R. Mehrabian and C. Lucas. A novel numerical optimization algorithm inspired from weed colonization. *Ecological Informatics 1*, pages 355–366, 2006.

[14] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(3):1087–1092, 1953.

[15] P. Pacheco. *Parallel programming with MPI*. Morgan Kaufmann Publishers, San Francisco, USA, 1996.

[16] Roberti, D. R. (2005). *Problemas inversos em fsica da atmosfera*. PhD thesis, Universidade Federal de Santa Maria.

[17] D. R. Roberti, A. Anfossi, H. F. C. Velho, and G. A. Degrazia. Estimation of emission rate of pollutant armospheric source. *Proceeding of ICIPE 2005*, 3:R03–1–R03–8, 2005.

[18] W. F. Sacco and C. R. E. de Oliveira. A new stochastic optimization algorithm based on a particle collision metaheuristic. *Proceedins of 6th WCSMO*, 2005.

[19] W. F. Sacco, H. A. Filho, and C. M. N. A. Pereira. Cost-based optimization of a nuclear reactor core design: a preliminary model. *Proceedins of INAC*, 2007.

[20] W. F. Sacco, C. M. F. Lapa, C. M. N. A. Pereira, and H. A. A. Filho. Two stochastic optimization algorithms applied to nuclear reactor core design. *Progress in Nuclear Energy*, (48):525–539, 2006.

[21] W. F. Sacco, C. M. F. Lapa, C. M. N. A. Pereira, and H. A. A. Filho. A metropolis algorithm applied to a nuclear power plant auxiliary feedwater system surveillance tests policy optimization. *Progress in Nuclear Energy*, (50):15–21, 2008.

[22] S. B. M. Sambatti. Diferentes estrategias de paralelizacao de um algoritmo genetico epidemico aplicadas na solucao de problemas inversos. Master's thesis, Computacao Aplicada, INPE, Sao Jose dos Campos, 2004.

[23] Silva Neto, A. J. and Becceneri, J. C. (2009). *Tcnicas de inteligncia computacional inspiradas na natureza: aplicao em problemas inversos em transferncia radiativa*. Notas em Matemtica Aplicada. SBMAC, So Carlos.

[24] Tikhonov, A. N. and Arsenin, V. Y. (1977). *Solution of ill-posed problems*. John Wiley & Sons ltd., New York.

ÍNDICE

PUBLICAÇÕES TÉCNICO-CIENTÍFICAS EDITADAS PELO INPE

Teses e Dissertações (TDI)

Teses e Dissertações apresentadas nos Cursos de Pós-Graduação do INPE.

Manuais Técnicos (MAN)

São publicações de caráter técnico que incluem normas, procedimentos, instruções e orientações.

Notas Técnico-Científicas (NTC)

Incluem resultados preliminares de pesquisa, descrição de equipamentos, descrição e ou documentação de programas de computador, descrição de sistemas e experimentos, apresentação de testes, dados, atlas, e documentação de projetos de engenharia.

Relatórios de Pesquisa (RPQ)

Reportam resultados ou progressos de pesquisas tanto de natureza técnica quanto científica, cujo nível seja compatível com o de uma publicação em periódico nacional ou internacional.

Propostas e Relatórios de Projetos (PRP)

São propostas de projetos técnico-científicos e relatórios de acompanhamento de projetos, atividades e convênios.

Publicações Didáticas (PUD)

Incluem apostilas, notas de aula e manuais didáticos.

Publicações Seriadas

São os seriados técnico-científicos: boletins, periódicos, anuários e anais de eventos (simpósios e congressos). Constam destas publicações o Internacional Standard Serial Number (ISSN), que é um código único e definitivo para identificação de títulos de seriados.

Programas de Computador (PDC)

São a seqüência de instruções ou códigos, expressos em uma linguagem de programação compilada ou interpretada, a ser executada por um computador para alcançar um determinado objetivo. Aceitam-se tanto programas fonte quanto os executáveis.

Pré-publicações (PRE)

Todos os artigos publicados em periódicos, anais e como capítulos de livros.