DESENVOLVIMENTO E IMPLEMENTAÇÃO DE VERSÕES PARALELAS DO ALGORITMO DO ENXAME DE PARTÍCULAS EM CLUSTERS UTILIZANDO MPI

Antonio de Oliveira Samel Moraes

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Química, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Química.

Orientadores: Paulo Laranjeira da Cunha Lage
Argimiro Resende Secchi

Rio de Janeiro
Dezembro de 2011

DESENVOLVIMENTO E IMPLEMENTAÇÃO DE VERSÕES PARALELAS DO ALGORITMO DO ENXAME DE PARTÍCULAS EM CLUSTERS UTILIZANDO MPI

Antonio de Oliveira Samel Moraes

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA QUÍMICA.

Examinada por:

Prof. Paulo Laranjeira da Cunha Lage, D.Sc.

Prof. Argimiro Resende Secchi, D.Sc.

Prof. José Herskovits Norman, D.Sc.

Prof. Renato Nascimento Elias, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
DEZEMBRO DE 2011

Moraes, Antonio de Oliveira Samel

Desenvolvimento e Implementação de Versões Paralelas do Algoritmo do Enxame de Partículas em Clusters utilizando MPI/Antonio de Oliveira Samel Moraes. – Rio de Janeiro: UFRJ/COPPE, 2011.

XVIII, 136 p.: il.; 29, 7cm.

Orientadores: Paulo Laranjeira da Cunha Lage

Argimiro Resende Secchi

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Química, 2011.

Referências Bibliográficas: p. 113 – 118.

1. Enxame de Parículas. 2. Message Passing Interface. 3. PSO Síncrono. 4. PSO Assíncrono. I. Lage, Paulo Laranjeira da Cunha *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Química. III. Título.

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

DESENVOLVIMENTO E IMPLEMENTAÇÃO DE VERSÕES PARALELAS DO ALGORITMO DO ENXAME DE PARTÍCULAS EM CLUSTERS UTILIZANDO MPI

Antonio de Oliveira Samel Moraes

Dezembro/2011

Orientadores: Paulo Laranjeira da Cunha Lage
Argimiro Resende Secchi

Programa: Engenharia Química

Apesar da reconhecida robustez do algoritmo de enxame de partículas (PSO – *Particle Swarm Optimization*) em problemas de otimização global, o elevado número de avaliações da função objetivo limita a sua aplicação em problemas de grande porte da engenharia. Por outro lado, esse algoritmo pode ser facilmente paralelizado, o que torna a computação paralela em clusters uma alternativa atraente para utilização eficiente deste método. Com esta motivação, três versões paralelas do PSO, utilizando o conjunto de funções da biblioteca MPI (Message Passing Interface), foram desenvolvidas e implementadas neste trabalho. Os algoritmos utilizam o paradigma *mestre–escravo*, diferindo entre si pelo modo de comunicação entre os processadores (síncrona ou assíncrona) e pela forma de atualização das partículas do enxame (imediata ou por revoada). Em particular, o algoritmo assíncrono desenvolvido (AIU–PPSO) demonstrou desempenho elevado, com *speedup* linear e *eficiência* superior a 90% em todos os resultados, os quais foram obtidos em computadores paralelos MIMD com memória distribuída ligados em rede Infiniband. Ao final, um problema real de estimação de parâmetros de elevado porte foi resolvido e a solução obtida comparada à fornecida pelo otimizador ODRPACK95.

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

DEVELOPMENT AND IMPLEMENTATION OF PARALLEL VERSIONS OF THE PARTICLE SWARM ALGORITHM ON CLUSTERS USING MPI

Antonio de Oliveira Samel Moraes

December/2011

Advisors: Paulo Laranjeira da Cunha Lage
Argimiro Resende Secchi

Department: Chemical Engineering

In despite its recognized strenght in global optimization problems, the high number of objective function evaluations requested by Particle Swarm Optimization (PSO) method limits its aplication in large problems of engineering. On the other hand, this algorithm can be easily parallelized, which makes the parallel computation on clusters an attractive alternative. Therefore, three parallel versions of the PSO algorithm, using the set of functions of the MPI (Message Passing Interface) library, were developed in this work. These parallel algorithms use the master–slave paradigm, differing among themselves by the communication mode among the processors (synchronous or asynchronous) and the particles update in swarm (immediate or by swarm). In particular, the asynchronous algorithm developed (the AIU–PPSO) showed an excelent perfomance, with linear speedup and eficiency higher than 90 % in all results, which were obtained on the MIMD parallel computers with distributed memory and Infiniband network. Finally, a real and large scale parameter estimation problem was solved and the solution was compared to that provided by the ODRPACK95 optimizer.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

F_{obj}	Valor da função objetivo, p. 42, 97
Max_aval	Número máximo de avaliações da função objetivo, p. 42
N_{aval}	Número de avaliações da função objetivo, p. 42, 68, 74
$N_{escravos}$	Número de processadores escravos, p. 30, 39, 42
N_{exp}	Número de experimentos, p. 59
N_{max}	Número de permanência máximo, p. 35, 42
N_{min}	Número de permanência, p. 36, 43
N_{pass}	Número de partículas, p. 42, 68
N_{proc}	Número total de processadores, p. 30, 83
$N_{sucesso}$	Número de sucessos obtidos, p. 59
S	Speedup, p. 83, 85, 91, 93
S	speedup, p. 60
χ	Robustez, p. 59, 68, 74
ϵ_a	Tolerância absoluta, p. 43
η	Eficiência de paralelização, p. 60, 83, 85, 91, 93
ϕ_1	Número aleatório distribuído no intervalo [0,c_1], p. 17, 124, 129
ϕ_2	Número aleatório distribuído no intervalo [0,c_2], p. 17, 124, 129
$\mathbf{v}$	Vetor de velocidade das partículas, p. 15, 34, 42

$\mathbf{x}$	Vetor de coordenadas das partículas, p. 15, 34, 42
$\mathbf{x}_g$	Vetor de melhor coordenadas enxame, p. 15, 34, 42
$\mathbf{x}_m$	Vetor de melhor coordenadas associado às partículas, p. 15, 34, 42
$\mathbf{y}$	Coordenadas da posição do enxame no espaço $[\mathbf{x}\|F_{obj}]$, p. 57
c_1	Parâmetro cognitivo, p. 15, 34, 68, 124, 129
c_2	Parâmetro social, p. 15, 34, 68, 124, 129
f_g	Melhor valor da função objetivo encontrado pelo enxame, p. 42
f_m	Melhor valor da função objetivo associado às partículas, p. 42
n	Número de variáveis independentes, p. 55, 57
w	Peso de inércia, p. 16, 34, 35, 42
w_0	Peso de inércia inicial, p. 35, 68
w_f	Peso de inércia final, p. 35, 68

Lista de Abreviaturas

ARS	Adaptative Randon Search, p. 1
FIFO	First In, First Out, p. 5, 47
GA's	Genetic Algorithms, p. 1, 11, 17
MIMD	Multiple Instruction, Multiple Data, p. 2, 22
MISD	Multiple Instruction, Single Data, p. 22
MPI	Message Passing interface, p. 2, 24
PAPSO	Parallel Asynchronous PSO, p. 93, 107
PBM	Population Balance Equation, p. 133
PBM	Population Based Method, p. 1, 11, 15, 17, 22, 77
PSO	Particle Swarm Optimization, p. 1, 11, 17
PSPSO	Parallel Synchronous PSO, p. 29, 93
SA	Simulated Annealing, p. 1, 11
SIMD	Single Instruction, Multiple Data, p. 22
SISD	Single Instruction, Single Data, p. 22
AIU–PPSO	Asynchronous and Immediate Update Parallel PSO, p. 2, 5, 52, 107
IU–PPSO	Immediate Update Parallel PSO, p. 2, 5, 39, 107
IU–SPSO	Immediate Update Serial PSO, p. 5, 34, 68
SU–PPSO	Swarm Update Parallel PSO, p. 2, 5, 39, 107
SU–SPSO	Swarm Update Serial PSO, p. 5, 34

Capítulo 1

Introdução

A otimização numérica vem sendo alvo de intenso estudo em diversos campos da engenharia como, por exemplo, no projeto de equipamentos, no controle e operação de processos e na estimação de parâmetros de modelos. Dois pontos de particular interesse são o desenvolvimento de algoritmos com características globais e a otimização de problemas de grande porte e dimensão, os quais demandam um elevado custo computacional.

Os algoritmos tradicionais baseados no gradiente da função objetivo, como o método de Newton e o método dos gradientes conjugados, embora eficientes, apresentam características fundamentalmente locais, o que os tornam bastante dependentes das estimativas iniciais. Neste contexto, algoritmos globais de otimização global são alvos de intenso estudo atualmente. Esses algoritmos podem ser determinísticos ou não determinísticos, a depender se utilizam ou não números aleatórios na sua formulação. Dentre os exemplos de algoritmos globais determinísticos estão o BARON e o DIRECT e dentre os não determísticos estão os algoritmos genéticos (GA's - *Genetic Algorithms*), o algoritmo de recozimento simulado (SA - *Simulated Annealing*), os sistemas fórmicos, o algoritmo de busca randômica adaptativa (ARS - *Adaptative Randon Search*) e o algoritmo de enxame de partículas (PSO - Particle Swarm Optimization).

Particularmente, algoritmos não determinísticos baseados no comportamento de populações (PBM's - *Population Based Methods*), como os algoritmos genéticos (GA's - *Genetic Algorithms*) e o algoritmo do enxame de partículas (PSO), ganham destaque por sua reconhecida robustez. A principal deficiência desses métodos é a necessidade de um elevado número de avaliações da função objetivo durante o

processo de busca. Para a aplicação em problemas de grande porte da engenharia (problemas que, normalmente, são também de grande dimensão), nos quais o cálculo da função objetivo é a etapa que mais demanda tempo computacional, esta limitação se torna crítica.

Para contornar essa deficiência, a computação paralela em clusters aparece como uma ferramenta bastante poderosa e útil. Devido a sua natureza, esses métodos são facilmente paralelizáveis, o que torna o desenvolvimento de estratégias paralelas um campo aberto para estudos. Em particular no PSO, o cálculo da função objetivo é realizado de maneira independente por cada partícula do enxame podendo ser, dessa forma, ser eficientemente paralelizado.

Apesar da facilidade mencionada, o desenvolvimento de versões paralelas do PSO é relativamente recente quando comparada a de outros PBM's. O primeiro trabalho de paralelização desse algoritmo foi produzido por SCHUTTE *et al.* [3] que desenvolveu uma versão paralela síncrona do código e, baseado neste, KOH *et al.* [4] produziram a primeira versão assíncrona do algoritmo, ambos utilizando a estratégia *mestre–escravo*. Neste tipo de estratégia, o processador mestre (ou mais corretamente, o processo mestre) realiza os cálculos do algoritmo, enquanto os demais processadores disponíveis, os escravos, realizam paralelamente os cálculos da função objetivo associados às partículas do enxame. A maneira pela qual é realizada a troca de informações entre o *mestre* e os *escravos* caracteriza o método empregado.

Com a utilização do paradigma *mestre–escravo*, três algoritmos paralelos para o PSO foram desenvolvidos e implementados neste trabalho: o IU–PPSO (*Immediate Update Parallel PSO*), o SU–PPSO (*Swarm Update Parallel PSO*) e o AIU–PPSO (*Asynchronous and Immediate Update Parallel PSO*). Os algoritmos diferenciam-se pelo modo de comunicação entre os processos *mestre* e *escravos*, que pode ser *síncrono* (IU–PPSO e SU–PPSO) ou *assíncrono* (AIU–PPSO), e pelo modo como a atualização da velocidade e posição das partículas é realizada, o que pode ser feito imediatamente antes do envio da partícula do *mestre* para os *escravos* (IU–PPSO e AIU–PPSO) ou feita em uma única etapa para todo o enxame a cada revoada do algoritmo (SU–PPSO).

Os algoritmos foram aplicados em diversos problemas padrões de otimização para validação e análise de desempenho. Neste último, um atraso temporal foi acrescido de modo aleatório e com variabilidade máxima determinada ao cálculo da função objetivo para emular problemas reais de engenharia. Os resultados alcançados demonstraram bom desempenho dos algoritmos em situações em que o

cálculo da função objetivo apresenta custo computacional levemente ou muito superior ao custo de comunicação de dados entre os processadores. Particularmente, o AIU–PPSO demonstrou *speedup* praticamente linear e eficiência superior a 90% para até os 128 processadores do cluster ARES[1] do Laboratório de Termofluidodinâmica (LTFD) do PEQ/COPPE/UFRJ, formado por computadores paralelos MIMD com memória distribuída ligados através de rede Infiniband. Os algoritmos foram desenvolvidos em linguagem de programação C[2] e o conjunto de funções da biblioteca MPI[3] (Message Passing Interface) foi utilizada para realizar a comunicação de dados entre os processadores.

Por fim, a estimação de parâmetros de um modelo de quebra e coalescência de gotas desenvolvido em ARAÚJO [1] foi realizada, e os resultados foram comparados aos obtidos nesta mesma referência com a utilização do otimizador ODRPACK95 ZWOKAK *et al.* [5] (apud ARAÚJO [1]). O custo elevado da simulação deste modelo o torna um excelente exemplo para a aplicação dos algoritmos desenvolvidos. Novamente, o AIU–PPSO demonstrou bom desempenho, resolvendo o problema em tempo de computação aceitável (o que não seria possível com a utilização de algoritmos seriais do PSO ou outros PBM's) e com resultados superiores ao da ODRPACK95.

Concorrentemente ao desenvolvimento de estratégias paralelas, diversos trabalhos na literatura vêm estudando o PSO desde a sua formulação inicial com intuito de melhorar o seu desempenho. Poucos deles, por outro lado, investigam o comportamento das partículas ou o desempenho do método em problemas de dimensão elevada. Análises do comportamento das partículas durante a otimização podem trazer contribuições importantes para o desenvolvimento de novas versões do algoritmo, adequadas à resolução eficiente de problemas de alta dimensão. Desse modo, concomitantemente ao desenvolvimento das estratégias paralelas, algumas análises sobre o PSO foram realizadas neste trabalho. Novos critérios de convergência para o enxame foram definidos e implementados, sendo comparados entre si em termos de *robustez* e número de avaliações da função objetivo, medida indireta de sua *perfomance*. As duas formas de atualização das posições e velocidades das partículas (*Immediate Update* e *Swarm Update*) também foram comparadas sob esses mesmos critérios. Por fim, um estudo de escalabilidade do número de partículas do enxame com a dimensão do espaço de busca foi realizado.

[1]Este cluster foi reconfigurado e atualmente se encontra sob o nome MARTE. Vide Seção 4.4.
[2]Compilador gcc da GNU, versões 4.1.2 (Red Hat 4.1.2-48) e 4.4.3
[3]Distribuição OpenMPI, versão 1.4.2 e 1.2.6

1.1 Objetivos

O objetivo principal deste trabalho é o desenvolvimento e a implementação de estratégias paralelas do algoritmo do enxame de partículas para a aplicação em problemas de grande porte da engenharia.

Os objetivos específicos deste trabalho, divididos entre *desenvolvimento e análise das características do método PSO* e *desenvolvimento e análise das estratégias paralelas do algoritmo PSO*, são listados a seguir.

Sobre o desenvolvimento e análise das características do método PSO

1. Desenvolver e implementar novos critérios de convergência para o enxame;
2. Analisar as duas formas de atualização de velocidade e posição das partículas (*Immediate Update* e *Swarm Update*);
3. Analisar a escalabilidade do número de partículas do enxame com a dimensão do espaço de busca.

Sobre o desenvolvimento e análise das estratégias paralelas do algoritmo PSO

1. Desenvolver e implementar estratégias paralelas do algoritmo do enxame de partículas (PSO);
2. Analisar a escalabilidade dos algoritmos paralelos com o número de processos do domínio de comunicação;
3. Aplicar os algoritmos paralelos desenvolvidos em problemas de grande porte da engenharia química.

1.2 Organização do texto

No Capítulo 2 é realizada a revisão bibliográfica desta dissertação. O mesmo é iniciado com uma descrição sucinta de problemas gerais de otimização na Seção 2.1

e os métodos de otimização global existentes para a resolução de tais problemas na Seção 2.2. Segue, na Seção 2.3, uma discussão detalhada do algoritmo de Enxame de Partículas e algumas de suas muitas variantes disponíveis na literatura. Previamente a revisão sobre estratégias paralelas do PSO, que é realizada na Seção 2.6, nas Seções 2.4 e 2.5 são discutidos, respectivamente, alguns conceitos ligados à computação paralela e ao MPI (*Message Passing Interface*), o padrão para comunicação de dados em computadores paralelos utilizado no desenvolvimento dos algoritmos paralelos deste trabalho.

No Capítulo 3, apresentam–se os algoritmos desenvolvidos e/ou implementados neste trabalho. Na Seção 3.1, são apresentadas as implementações das versões seriais dos algoritmos, o IU–SPSO e o SU–SPSO. Na Seção 3.2, são apresentadas e discutidas as implementações dos algoritmos paralelos síncronos, extensões paralelas dos algoritmos seriais, o IU–PPSO (Seção 3.2.1) e o SU–PPSO (Seção 3.2.2). Na Seção 3.3 discute–se o algoritmo paralelo assíncrono (o AIU–PPSO). Antes de detalhar a implementação deste algoritmo em si, é feita uma revisão na Sec cão 3.3.1 sobre *listas encadeadas*, estrutura de dados utilizadas para a criação de filas *FIFO*, ferramenta chave na implementação deste código.

No Capítulo 4, são apresentadas as definições e funções utilizadas no desenvolvimento e análise dos códigos. Na Seção 4.1 são definidas as métricas no espaço euclidiano utilizadas para o desenvolvimento dos critérios de convergência do enxame, os quais são também definidos nesta seção. Na Seção 4.2 é discutida a forma de caracterização de sucesso em uma determinada busca sobre um problema de teste padrão, com qual se define a *robustez* e a *perfomance* dos algoritmos, esta última sendo medida usando o número de avaliações da função objetivo. Na Seção 4.3 são definidos o *Speedup* e a *Eficiência de paralelização*, conceitos utilizados para a análise de desempenho dos algoritmos paralelos desenvolvidos. Por fim, nas Seções 4.4 e 4.5 são apresentados os recursos computacionais disponíveis e as funções teste utilizadas para a análise dos algoritmos desenvolvidos.

No Capítulo 5 apresentam–se os resultados obtidos e suas discussões. Este capítulo é dividido em duas partes que analisam as características específicas dos algoritmos PSO implementados (Seção 5.1) e o desempenho das versões paralelas desenvolvidas (Seção 5.2). Nesta última, são apresentados ainda os resultados da aplicação do AIU–PPSO a um problema de estimação de parâmetros de porte significativamente elevado, cujo custo médio de avaliação da função objetivo é cerca de *18 minutos* nos computadores paralelos de ARES[4] (LTFD/PEQ/COPPE/UFRJ).

[4]Ver Seção 4.4

Nos Capítulos 6 e 7 estabelecem–se, respectivamente, as principais conclusões e sugestões sobre os estudos realizados, os códigos desenvolvidos e os resultados obtidos nesta dissertação.

Capítulo 2

Revisão Bibliográfica

Neste capítulo é feita uma revisão da literatura a respeito do algoritmo de enxame de partículas (PSO) e suas versões para computação paralela. Inicialmente, nas Seções 2.1 e 2.2, o problema geral de otimização e alguns métodos ditos globais para a sua resolução são discutidos. Na Seção 2.3, faz-se a revisão sobre o algoritmo de enxame de partículas em si, onde discute-se o método clássico e algumas de suas variantes. Nas Seções 2.4 e 2.5, é feita uma breve revisão sobre computação paralela e sobre MPI (*Message Passing Interface*) para então, na Seção 2.6, ser feita a revisão sobre os algoritmos paralelos do enxame de partículas.

2.1 Otimização

De maneira simples e direta, um problema de otimização consiste em encontrar o valor mínimo ou máximo de uma função (chamada função objetivo) em um dado espaço de busca que pode ser ou não restrito. A função objetivo é uma aplicação que associa a qualquer ponto do espaço de busca um determinado valor. Em particular, no caso em que se procura o *"melhor"* dentre todos os pontos no espaço de busca, ou seja, o ótimo global, tem-se o problema de *Otimização Global*. Na prática, para funções de elevada complexidade, este *"melhor ponto"* pode não ser de fato encontrado. Portanto, métodos de otimização definidos como *"Globais"* são assim chamados por tentarem encontrar o ponto de ótimo global da função objetivo.

Formalmente, o ponto $x^* \in X$ de uma dada função objetivo $S : X \subseteq \Re^n \rightarrow \Re$

é dito mínimo global de S se $S(x^*) \leq S(x)$ para qualquer $x \in X$. Por outro lado, se existe $\epsilon > 0$, tal que $S(x^*) \leq S(x)$ para todo x, tal que $\| x - x^* \| < \epsilon$, x^* é dito mínimo local de S (EDGAR *et al.* [6]). No caso em que não há restrições sobre o conjunto X, um problema de otimização é definido como:

$$x^* = \arg \Big[\min_{x \in X} S(x) \Big] \tag{2.1}$$

O conjunto X é chamado *domínio de busca*, podendo ser todo o espaço $\Re^n$, ou apenas parte dele.

No caso da otimização com restrição, apenas parte dos pontos $x \in X$ podem ser assumidos como possíveis soluções do problema, os quais devem satisfazer a restrições de igualdade ou desigualdade. De maneira geral, o problema de otimização com restrição pode ser definido como:

$$\begin{gathered} x^* = \arg \Big[\min_{x \in X} S(x) \\ \text{sujeito à} : h_j(x) = 0, \ j = 1, ...m \\ g_j(x) \leq 0, \ j = 1, ...p \\ x \in X \subseteq \Re^n \Big] \end{gathered} \tag{2.2}$$

onde $h_j(x)$ são as m restrições de igualdade e $g_j(x)$ são as p restrições de desigualdade.

As restrições definem o conjunto $D \subseteq X$, tal que a solução do problema é $x \in D$. Neste caso, o conjunto D é chamado de conjunto dos pontos viáveis de otimização, definido como:

$$D = \{x \in X \subseteq \Re^n | h(x) = 0, \ g(x) \leq 0\} \tag{2.3}$$

Neste trabalho, apenas o problema de otimização sem restrição foi considerado. Contudo, os métodos desenvolvidos podem ser aplicados à problemas de otimização com restrição que possam ser reformulados, através do uso de funções penalidades, em problemas de otimização sem restrição.

2.2 Métodos de Otimização Global

Existem diversos métodos para a solução dos problemas definidos na seção anterior. De modo geral, para problemas sem restrições no espaço de busca, esses métodos podem ser agrupados em:

1. Método que utilizam derivadas da função objetivo (métodos analíticos ou métodos indiretos);
2. Métodos que não utilizam as derivadas da função objetivo (métodos de busca ou diretos);

Os métodos que utilizam derivadas (métodos indiretos) baseiam-se no fato de que no ponto crítico (um ótimo local da função e potencial ótimo global) as derivadas da função objetivo em relação as variáveis de busca são nulas. Esta é chamada de condição necessária de otimalidade de 1^a ordem. Ainda, se a matriz Hessiana da função (que seja duas vezes diferenciável no ponto) for positiva definida, a condição é suficiente para que o ponto seja um ótimo local estrito da função (SECCHI e BISCAIA [7]). Devido as suas características matemáticas, os métodos que utilizam derivadas são fundamentalmente locais. Exemplos clássicos são os *método do gradiente descendente*, o *método dos gradientes conjugados* e o *método de Newton*, este último utilizando ainda a segunda derivada da função (para funções monovariadas) ou a matriz Hessiana da mesma (funções multivariadas). Há ainda os métodos que se baseiam em aproximações da matriz Hessiana do sistema (métodos de métrica variável) podendo-se citar os métodos *Broyden-Fletcher-Goldfard-Shanno* (BFGS), dentre vários outros. (EDGAR *et al.* [6]).

Os métodos que não utilizam derivadas (métodos diretos) não assumem as características matemáticas supracitadas. Nesses métodos, apenas o valor da função objetivo é utilizado como informação para a busca, o que torna intrínseca a necessidade de procedimentos de comparação de seus valores. Podem ser *determinísticos* ou *não determinísticos*, a depender se utilizam ou não números aleatórios na sua formulação. Em comparação aos métodos que utilizam derivadas, os métodos diretos são mais robustos para problemas de otimização global sendo, por outro lado, menos eficientes computacionalmente, necessitando de uma maior quantidade de avaliações da função objetivo.

As Seções 2.2.1 e 2.2.2 discutem alguns métodos diretos determinísticos e não determinísticos, utilizados para o problema de otimização global.

2.2.1 Métodos determinísticos

Um método é chamado determinístico se for possível predizer todos os seus passos a partir de um determinado ponto inicial, levando sempre ao mesmo resultado para execuções que partam desse mesmo ponto. Por outro lado, nos métodos *não determinísticos*, devido ao uso de números aleatórios, cada execução do código, partindo de mesmo ponto ou população inicial, pode conduzir a resultados diferentes. Dentre alguns exemplos de métodos determinísticos para otimização global estão o BARON (SAHINIDIS e TAWARMALANI [8]) e o DIRECT (BJORKMAN e HOLMSTROM [9]; FINKEL [10]).

O BARON (Branch and Reduce Optimization Navigator) é um programa que implementa métodos determinísticos para resolução de problemas de Programação Não-Linear (PNL) e Programação Não-Linear Inteira Mista (PNLIM). O programa é baseado na técnica *branch-and-bound*, originalmente proposta por LAND e DOIG [11], que consiste na enumeração sistemática dos candidatos à solução dentro de intervalos viáveis dispostos em uma árvore de busca. Esta árvore é construída pela repetição recursiva do procedimento *branching*, que consiste de ramificações de uma região viável em sub-regiões viáveis menores. Em uma segunda etapa é aplicado o procedimento de *bound*, para a seleção dos limites inferior e superior das sub-regiões viáveis. As sub-regiões inviáveis são descartadas pela aplicação do procedimento de *poda* (*pruning*), o qual considera (para um problema de minimização) que se o valor da função objetivo no limite inferior de uma dada sub-região é maior do que o valor do ótimo já encontrado até o momento, então esta sub-região pode ser descartada (BOSSOIS *et al.* [12]).

O DIRECT é um algoritmo do tipo busca por *coleta* que utiliza a informação obtida nos pontos de amostragem do domínio para decidir sobre a direção de busca a ser utilizada, sem a necessidade de utilizar informações do gradiente da função objetivo. Basicamente o algoritmo trabalha dividindo o domínio de busca em *hipercubos* em cujo centro se armazena o valor da função objetivo. A seleção dos *hipercubos* ótimos potenciais e a sua posterior divisão em *hipercubos* menores constituem os pontos chaves do algoritmo. Os algoritmos pelos quais esses passos são realizados, assim como as idéias básicas do algoritmo são descritas em FINKEL [10].

2.2.2 Métodos não determinísticos

Os métodos baseados em buscas aleatórias sobre o espaço, por utilizarem apenas o valor da função objetivo nos pontos do espaço de busca e não o valor de suas derivadas, não requerem a regularidade e/ou continuidade da função objetivo. Dentre os principais exemplos dessa classe de métodos estão os algoritmos genéticos (GA's), o algoritmo de recozimento simulado (SA- Simulated Annealing), os algoritmos fórmicos e o algoritmo do enxame de partículas (PSO- Particle Swarm Optimization).

A principal desvantagem desses métodos é o elevado número de avaliações da função objetivo, o que compromete drasticamente a sua aplicação em problemas de grande porte da engenharia. Por outro lado, eles são também facilmente paralelizáveis, o que faz da computação paralela em clusters extremamente atrativa para a aplicação desses métodos a esses tipos de problemas.

Algoritmos genéticos

A formulação original dos algoritmos genéticos foi desenvolvida por HOLLAND [13]. O algoritmo pertence à classe de métodos baseados no comportamento de populações (PBM's) e está fundamentado nos príncipios básicos da *Genética Populacional*, que afirma que a variabilidade entre os indíviduos de uma população que se reproduzem sexualmente é resultado de mutações aleatórias nos indivíduos e da recombinação genética entre os mesmos na reprodução. Esses princípios são traduzidos no método, de maneira simplificada, da seguinte forma:

1. As características dos indíviduos estão registradas nos seus genes de maneira codificada;
2. Cada geração de indivíduos em uma população é formada pela combinação do material genético de indivíduos da geração anterior;
3. Os indivíduos podem sofrer mutações aleatórias em uma dada geração tornando-se mais ou menos aptos;
4. Os indivíduos mais aptos têm maior probabilidade de se reproduzirem e, por conseguinte, transferirem seu material genético;
5. Todos os indivíduos têm a mesma probabilidade de sofrer mutação.

Os passos fundamentais dos algoritmos genéticos podem ser descritos de forma simplificada, como:

1. Gerar população inicial;
2. Selecionar pares para efetuar o cruzamento;
3. Selecionar aleatoriamente os indivíduos para sofrer mutação;
4. Avaliar a aptidão dos indivíduos da nova população;
5. Testar o critério de convergência do algoritmo.

A aptidão do indivíduo está relacionada ao valor da função objetivo que ele associa. Assim, indivíduos mais aptos são aqueles que apresentam os melhores valores da função objetivo. Os indivíduos podem ser codificados de várias maneiras. No caso de funções contínuas definidas no espaço dos números reais, a codificação real é a mais adequada, pois evita o processo de decodificação para avaliar a aptidão do indivíduo.

O cruzamento entre os indivíduos é realizado através do operador de cruzamento. Previamente a sua aplicação, deve ser realizada a seleção dos indivíduos (os pais) a cruzarem. Esta pode ser feita através de algum algoritmo de seleção, citando-se aqui a seleção por roleta e a seleção por torneio. Esses algoritmos devem levar em consideração a aptidão dos indivíduos, sendo que os mais aptos devem ter maior probabilidade de serem selecionados. Efetuada a seleção, é feito o cruzamento para a geração dos filhos. No caso da codificação real, estes podem ser gerados, por exemplo, pela média ponderada entre as características dos pais (no caso a sua posição no espaço de busca).

A mutação é realizada através do operador mutação que pode ser aplicado indiferentemente a qualquer indivíduo da população. Este operador será aplicado de acordo com o sorteio de um número aleatório que deve ser menor do que uma probabilidade pré-estabelecida de mutação.

Em adição a esses operadores básicos há ainda algumas variantes de operadores introduzidas com o intuito de melhorar o desempenho e a eficiência do algoritmo. Dentre elas tem-se o elitismo, que consiste em replicar o indivíduo de uma dada geração inalterado na geração seguinte, a reinicialização periódica da população, com o objetivo de aumentar a velocidade de convergência do algoritmo

devido a eventual tendência dos indivíduos para mínimos locais e a estratégia de *niching* que objetiva afastar a população de regiões próximas a mínimos locais.

O algoritmo de recozimento simulado

Este método baseia-se no processo de reorganização da estrutura cristalina de metais submetidos a um processo de recozimento (elevação da temperatura até próximo à temperatura de fusão e resfriamento gradual para cristalização) para remover defeitos nesta estrutura. Neste processo físico, METROPOLIS *et al.* [14] perceberam que a direção natural do processo de recozimento é a de minimização da energia dos átomos de modo a organizá-los na estrutura cristalina de menor energia possível em uma dada temperatura.

Em analogia a este processo físico, o algoritmo busca a minimização da função objetivo, que neste caso corresponde à energia do sistema termodinâmico, enquanto que o estado termodinâmico do sistema é uma solução viável do problema e o estado de referência é o *"mínimo global"* da função (menor valor da função que se conhece até o estado atual do sistema). Para evitar que o algoritmo convirja para um mínimo local, a busca pode prosseguir em direções que resultem em elevação da energia do sistema (valor da função objetivo). Este movimento é, entretanto, controlado pelo parâmetro "temperatura (T)" do sistema, sendo aceito apenas com uma determinada probabilidade $exp(\frac{-\delta}{T})$, onde $\delta = S(x^{k+1}) - S(x^k)$. O parâmetro $T \in (0, \infty)$ é dado pelo processo de resfriamento, devendo ser uma função decrescente com as iterações do algoritmo, sendo que altos valores favorecem a busca global, enquanto baixos valores a busca local (SECCHI e BISCAIA [7]).

Sistemas fórmicos

É um método originalmente formulado para otimização combinatorial estando baseado no comportamento natural de formigas para o desenvolvimento de mecanismos de cooperação e aprendizado (BONABEAU *et al.* [15]). Neste algoritmo, um conjunto de formigas artificiais cooperam entre si através da utilização de feromônio digitais que são depositados nas arestas de um grafo. As formigas podem se movimentar através desse grafo deixando um rastro de feromônio cuja intensidade é função do valor da função objetivo naquela determinada aresta. Esse feromônio age como uma memória das formigas sendo utilizado para que as mesmas caminhem na

direção de melhora da sua solução. Para evitar a convergência para ótimos locais é realizada ainda, uma escolha probalística com a utilização de números aleatórios de cada passo da solução (SECCHI e BISCAIA [7]).

Busca randômica adaptativa

Formulado por SECCHI e PERLINGEIRO [16], é um algoritmo com características globais que se baseia em uma busca aleatória memorizada, com distribuição assimétrica e redução sistemática da hiperelipse de amostragem. Os pontos de memorização são selecionados no domínio de acordo com a equação 2.4.

$$x_i^{k+1} = x_i^k + \frac{R_i}{\delta}[b_i(a_i\phi_i - 1)]^\delta \tag{2.4}$$

Nesta equação, $\mathbf{x}^k$ é o melhor ponto da última amostragem, ϕ é um vetor de números aleatórios sorteados no intervalo [0,1], $\mathbf{R}$ é o vetor dos eixos da hiperelipse, δ é o parâmetro de forma da distribuição das amostras, $\mathbf{a}$ é o vetor de assimetria da distribuição e $\mathbf{b}$ é o vetor de direção da assimetria.

As principais etapas do algoritmo são descritas abaixo:

1. Amostragem de pontos aleatórios com a equação 2.4;
2. Ordenação das amostras com base no valor da função objetivo;
3. Memorização dos melhores pontos das amostras;
4. Verificação dos critérios de convergência;
5. Analise de redundância dos pontos (baseado em uma métrica de análise);
6. Resgate de um ponto da memória;
7. Verificação da tendência da busca;
8. Ajuste dos parâmetros da busca ($\mathbf{a}, \mathbf{b}$ e δ);
9. Análise do critério de parada;
10. Ordenação dos ótimos;

Algoritmo de enxame de partículas

O algoritmo do enxame de partículas, assim como os algoritmos genéticos, pertence à classe de algoritmos baseado no comportamento de populações (PBM's). Como é o algoritmo foco desta dissertação, na Seção 2.3 é realizada uma revisão detalhada deste algoritmo.

2.3 O Algoritmo do Enxame de Partículas

O Algoritmo do Enxame de Partículas (PSO) é um método não determinístico de otimização que tem como motivação a simulação do comportamento gregário encontrado entre diversas espécies na natureza. Formulado inicialmente por EBERHART e KENNEDY [17], ele pertence à classe dos algoritmos baseados em inteligência de enxames, que partem do pressuposto de que o comportamento individual de cada elemento depende fortemente de uma componente social. Essa componente social está relacionada à capacidade de interação entre os elementos do enxame, através de mecanismos de troca de informação entre os indivíduos que podem ser explícitos ou implícitos.

No algoritmo, a cada partícula, que corresponde a uma solução potencial do problema, estão associadas uma velocidade e uma posição no espaço n-dimensional formado pelo produto cartesiano das variáveis de otimização. O algoritmo consiste em, a partir de uma população inicial, que pode ser gerada aleatoriamente no domínio de busca, atualizar a velocidade e a posição das partículas segundo equações que caracterizam o comportamento das mesmas.

Na sua forma original, a atualização da velocidade das partículas é influenciada por três termos: o *termo de inércia*, o *termo de cognição* e o *termo social*. O primeiro está relacionado à influência da velocidade da partícula sobre o seu próprio movimento. O *termo de cognição* está relacionando à influência da melhor posição já registrada pela partícula sobre o seu próprio movimento, enquanto que o *termo social* relaciona a influência da melhor posição registrada pelo enxame sobre o movimento de cada partícula.

As equações 2.5 e 2.6 mostram as formas de atualização da velocidade e da posição das partículas originalmente proposta por EBERHART e KENNEDY [17].

$$\mathbf{v}_i^{k+1} = \mathbf{v}_i^k + c_1 r_1[\mathbf{x}_{m,i}^k - \mathbf{x}_i^k] + c_2 r_2[\mathbf{x}_g^k - \mathbf{x}_i^k] \tag{2.5}$$

$$\mathbf{x}_i^{k+1} = \mathbf{x}_i^k + \mathbf{v}_i^k \tag{2.6}$$

onde $\mathbf{x}_i$ e $\mathbf{v}_i$[1] são a posição e a velocidade da partícula i, $\mathbf{x}_{m,i}$ é a coordenada da melhor posição já encontrada pela partícula i, $\mathbf{x}_g$ é a coordenada da melhor posição já encontrada pelo enxame e o índice k é o contador de revoadas (ou iterações) do algoritmo. Os parâmetros c_1 e c_2 são os parâmetros cognitivo e social, enquanto que r_1 e r_2 são dois números sorteados aleatoriamente no intervalo $[0, 1]$ a cada *revoada* do algoritmo.

A primeira variante importante do algoritmo foi proposta por SHI e EBERHART [18]. Essa modificação consistiu na introdução do *peso de inércia* (w), parâmetro multiplicativo do primeiro termo ao lado direito da equação 2.5. Esse termo pondera a influência da velocidade atual da partícula sobre o seu próprio movimento, sendo o seu principal efeito o de promover um balanceamento entre o caráter local e global da busca. Como observado pelos autores, altos valores do peso de inércia tornam o enxame mais exploratório, enquanto que pequenos valores favorecem a busca mais localizada, aumentando a velocidade de convergência das partículas. Dessa maneira, os autores propuseram diminuir linearmente o valor de w com o número de iterações (ou revoadas) do algoritmo na medida em que a busca se processa.

A equação 2.7 mostra a forma de atualização da velocidade das partículas com a inserção do peso de inércia. A partir deste momento, a forma de atualização da velocidade com peso de inércia passará ser denominada de *PSO-Clássico*.

$$\mathbf{v}_i^{k+1} = w\mathbf{v}_i^k + c_1 r_1[\mathbf{x}_{m,i}^k - \mathbf{x}_i^k] + c_2 r_2[\mathbf{x}_g^k - \mathbf{x}_i^k] \tag{2.7}$$

CLERC [19] propôs o uso do fator de constrição (K) à equação original de atualização da velocidade com o intuito de assegurar a convergência das partículas. A equação 2.8 mostra a nova forma de atualização da velocidade proposta.

$$\mathbf{v}_i^{k+1} = K[\mathbf{v}_i^k + c_1 r_1(\mathbf{x}_{m,i}^k - \mathbf{x}_i^k) + c_2 r_2(\mathbf{x}_g^k - \mathbf{x}_i^k)] \tag{2.8}$$

[1]Dimensionalmente a variável $\mathbf{v}_i$ corresponde a um deslocamento da partícula i. Contudo, assim como referenciado na maioria dos textos sobre o PSO, ela é chamada velocidade.

$$K = \frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|} \tag{2.9}$$

$$\varphi = c_1 + c_2 \qquad \text{e} \qquad \varphi > 4 \tag{2.10}$$

SHI e EBERHART [18] observaram que o fator de constrição não altera a forma básica da equação 2.7, mas faz uma redefinição dos parâmetros da mesma (SCHWAAB [20]). A vantagem é a autocorreção dos parâmetros promovida por este fator que pode assegurar a estabilidade do método, garantindo a convergência das partículas.

BISCAIA *et al.* [21] propuseram uma nova versão do algoritmo no qual o movimento das partículas é descrito segundo a solução de um sistema dinâmico de segunda ordem subamortecido. A natureza fundamental do algoritmo é mantida, mas a forma de movimentação das partículas passa a ser diferente.

As equações 2.11 e 2.12 mostram as formas de atualização da posição e velocidade das partículas nesta nova versão.

$$\mathbf{x}_i^{k+1} = \mathbf{X}_i^k + \exp(\frac{w-1}{2})\Big\{(\mathbf{x}_i^k - \mathbf{X}_i^k)\cos(\theta_k) + \Big[\frac{1-w}{2}(\mathbf{x}_i^k - \mathbf{X}_i^k) + \mathbf{v}_i^k\Big]\frac{\sin(\theta_k)}{\theta_k}\Big\} \tag{2.11}$$

$$\mathbf{v}_i^{k+1} = \exp(\frac{w-1}{2})\Big\{\mathbf{v}_i^k\cos(\theta_k) - \Big[(\phi_1+\phi_2)(\mathbf{x}_i^k - \mathbf{X}_i^k) + \frac{1-w}{2}\mathbf{v}_i^k\Big]\frac{\sin(\theta_k)}{\theta_k}\Big\} \tag{2.12}$$

em que:

$$\theta_k = \sqrt{(\phi_1+\phi_2) - \frac{(1-w)^2}{4}} \tag{2.13}$$

$$\mathbf{X}_i^k = \frac{\phi_1\mathbf{x}_{m,i}^k + \phi_2\mathbf{x}_g^k}{\phi_1+\phi_2} \tag{2.14}$$

A principal vantagem dessa nova versão é a sua estabilidade incondicional para valores do peso de inércia menores do que 1 ($w < 1$), que é uma região maior do que a do PSO-Clássico (ver Apêndice C). Os autores classificaram esta nova forma do algoritmo como *PSO-Modificado*. No Apêndice B este *PSO-Modificado* é melhor estudado, mostrando que essa forma de atualização da velocidade é idêntica à do PSO-Clássico (equação 2.7) com parâmetros *"sintonizados"* de modo a garantir a convergência das partículas.

O PSO vem se consolidando desde a sua formulação inicial como uma boa opção para problemas de *otimização global*, principalmente devido a sua notória robustez e a sua aparente superioridade em eficiência em relação a outros PBM's. Outras vantagens frequentemente citadas são a fácil implementação e a pequena quantidade de parâmetros de ajuste em relação aos mesmos.

Para ratificar a questão da superioridade em eficiência, HASSAN *et al.* [22] apresentaram um trabalho de comparação entre o PSO e os GA's. Seus resultados indicaram que os dois métodos apresentaram a mesma efetividade para encontrar o mínimo global, mas o PSO foi significativamente mais eficiente em termos de número de avaliações da função objetivo. Dos oito problemas testados, em cinco deles o PSO foi estatisticamente superior ao AG com intervalo de confiança de 99 % e superior em média nos outros três problemas.

Uma outra conclusão importante deste trabalho, foi a de que o nível da superioridade do PSO foi dependente do problema. Pelos resultados apresentados, a superioridade do PSO parece ser mais pronunciada nos casos sem restrições no espaço de variáveis contínuas e em menor grau quando o espaço é restrito, seja com variáveis contínuas ou discretas.

Apesar dessa aparente superioridade em eficiência do PSO, esse algoritmo apresenta como principal deficiência o alto custo computacional, resultado do elevado número de avaliações da função objetivo requerido no processo de busca. Motivados por essa limitação, muitos dos estudos sobre esse algoritmo concentram-se na melhora do seu desempenho. Os mesmos são bastante diversificados e, em sua maioria, consistem na proposição de novas estratégias heurísticas para modificar o comportamento do enxame. Essas estratégias passam pelo estudo da influência dos parâmetros, pela adição de novos termos às equações básicas de movimentação das partículas ou mesmo pela incorporação ao algoritmo básico de conceitos advindos de outros PBM's.

Particularmente em relação aos GA's, o PSO guarda uma diferença muito importante: *os mecanismos de compartilhamento de informações.* Nos GA's, os cromossomos de cada indíviduo podem compartilhar as informações nele contida com quaisquer outros indíviduos da população. Dessa forma, a população se movimenta entre as gerações como um grupo na busca do ótimo. Por outro lado, no PSO, as partículas têm apenas memória individual da sua velocidade, da sua melhor posição e da melhor posição do enxame.

Possivelmente baseado nesta diferença entre mecanismos de troca de informação, ARUMUGAM e RAO [23] propuseram a formulação de um PSO *"híbrido"*, incluindo o operador típico dos GA's, o *cross-over*, como mecanismo de troca de informações entre as partículas. Neste mesmo trabalho, os autores propuseram o uso adaptativo dos parâmetros do PSO, o *peso de inércia* e os parâmetros *cognitivo e social*, levando em consideração a informação dos ótimos locais e global das partículas em contraposição ao uso clássico, que considera apenas o decaimento linear do *peso de inércia*. Seus resultados demonstram que as variantes introduzidas melhoram o desempenho do enxame nos problemas testados.

CHEN e ZHAO [24] propuseram a adaptação do tamanho da população ao longo do processo de busca baseado no conceito de *diversidade da população*, definida pelos autores através de métricas sobre disposição das partículas no espaço de busca. O trabalho é baseado na ideia de que perto do final do processo de busca, quando as partículas estão tendendo à convergência, uma população grande não é mais necessária, pois a diversidade é pequena. Por outro lado, quando a diversidade é grande, uma população maior pode conferir maiores chances de alcançar o ótimo global.

HE *et al.* [25] introduziram um novo termo à equação de velocidade das partículas, nomeado por eles de *congregação passiva*. Por congregação passiva, depreende-se a atração sobre uma partícula exercida por outros membros do enxame sem que haja a manifestação de um comportamento social típico. A despeito de quaisquer interpretações físicas fornecidas, o PSO com congregação passiva pode ser representado pela seguite equação 2.15 de velocidade das partículas.

$$\mathbf{v}_i^{k+1} = w\mathbf{v}_i^k + c_1 r_1[\mathbf{x}_{m,i}^k - \mathbf{x}_i^k] + c_2 r_2[\mathbf{x}_g^k - \mathbf{x}_i^k] + c_3 r_3[\mathbf{R}_i^k - \mathbf{x}_i^k] \tag{2.15}$$

em que $\mathbf{R}_i$ são as coordenadas de posição de uma partícula selecionada aleatoriamente no enxame, c_3 é o coeficiente de congregação passiva e r_3 é um número aleatório no intervalo $(0, 1)$.

O termo de congregação passiva age, portanto, como uma atração aleatória exercida por um membro qualquer do enxame sobre as demais partículas. Este termo pode ser visto como uma perturbação estocástica com efeito de possibilitar a troca de informação entre as partículas. Em termos práticos, sua ação parece aumentar

a diversidade do enxame, eventualmente auxiliando o escape de regiões próximas a mínimos locais. Os resultados apresentados mostram uma melhora no desempenho do enxame com a presença deste termo. Ainda, em diversos gráficos da história do valor da função objetivo com as gerações do algoritmo, podem ser observados claramente a alteração do valor ótimo do enxame. Esse comportamento, que na maioria dos casos é acompanhado de ruídos no gráfico histórico, pode representar o escape de mínimos locais.

De modo semelhante à proposta de congregação passiva, KALIVARAPU *et al.* [26] propuseram o uso de feromônios digitais que são adicionados como um termo extra à equação de atualização da velocidade. O objetivo deste termo é simular a ação dos feromônios liberados por insetos para encontrar caminhos adequados na busca de alimentos. A equação 2.16 mostra como é introduzida a ação dos feromônios digitais no enxame. Nesta equação, r_3 é um número aleatório sorteado no intervalo $[0, 1]$, c_3 é o parâmetro de confiança no feromônio (com ação similar a dos parâmetros c_1 e c_2) e $\mathbf{P}_i^k$ é o feromônio alvo da partícula i na revoada (iteração) k do algoritmo.

$$\mathbf{v}_i^{k+1} = w\mathbf{v}_i^k + c_1 r_1[\mathbf{x}_{m,i}^k - \mathbf{x}_i^k] + c_2 r_2[\mathbf{x}_g^k - \mathbf{x}_i^k] + c_3 r_3[\mathbf{P}_i^k - \mathbf{x}_i^k] \tag{2.16}$$

No início do algoritmo, 50 % da população de partículas são sorteadas aleatoriamente para deixar feromônios no espaço de busca, independente do seu valor de função objetivo. Para as revoadas subsequentes, somente as partículas que melhorarem o seu valor de função objetivo estarão habilitadas para depositá-lo. O nível do feromônio é normalizado entre 0 e 1. Cada feromônio depositado tem nível 1, sendo que este decai[2] com o tempo (iterações ou revoadas do algoritmo) com uma taxa definida pela usuário (valor padrão é igual a 0.95). Para evitar que o número de feromônios no espaço torne-se demasiadamente grande, os mesmos podem ser *fundidos* aos pares para formar um novo feromônio, cujo nível e posição é o ponto médio dos feromônios fundidos. A possibilidade de fusão é verificada em termos do raio de influência de cada feromônio, sendo que aqueles que estiverem dentro, mutualmente, do raio de influência do outro são fundidos. Por fim, o feromônio alvo de um partícula i é determinado com base na sua distância normalizada para essa partícula e no seu nível, sendo aquele para o qual o produto de sua distância para a mesma e o seu nível for o maior dentre os outros feromônios no espaço.

[2]Não é dito no artigo qual a forma desse decaimento. Uma das possibilidades é de que seja um decaimento linear

ENGELBRECHT e VAN DEN BERGH [27] utilizam o conceito de aprendizado cooperativo, ideia extraída de POTTER e JONG [28] que implementaram esse conceito na sua versão dos algoritmos genéticos. Em POTTER e JONG [28], ao invés de um AG simples ser resolvido, a população é dividida em múltiplas outras que podem cooperar através da troca de informação, pelo intercâmbio de agentes ou pela migração de indivíduos.

De modo semelhante ao anterior, JIANG *et al.* [29] apresentam, conforme designado por eles, um PSO melhorado (Improved PSO - IPSO). No seu algoritmo, o enxame é particionado em sub-enxames, os quais executam de maneira independente o algoritmo PSO ou seus variantes. Em *"gerações"* ou *"revoadas"* especificadas, os sub-enxames são forçados a se misturar para a troca de informações. Novamente, os resultados apresentados mostram a superioridade do novo método em relação ao PSO original. Este tipo de estratégia também foi adotada por WAINTRAUB *et al.* [30] com seu modelo de ilhas vizinhas. Por se tratar de um modelo em computação paralela a discussão desta última referência será retomada na Seção 2.6.

Inúmeras referências sobre o PSO estão disponíveis na literatura. São muitos os tipos de estratégias utilizadas como tentativa de melhorar o seu desempenho. A tarefa de avaliação das estratégias já estudadas, por si só, poderia fornecer conhecimento substancial para a implementação de versões mais eficientes deste algoritmo. Para não estender muito a discussão e perder o foco do trabalho, sugere-se, adicionalmente às referências já discutidas, a leitura dos trabalhos de PEDERSEN e CHIPPERFIELD [31], COELHO [32], HE e WANG [33], JIE *et al.* [34] e HU *et al.* [35].

Muitos trabalhos de aplicação do PSO em problemas práticos de engenharia também estão disponíveis. Dentre eles podem-se citar o trabalho de REINBOLT *et al.* [36], de MENESES *et al.* [37] e o de PATEL e RAO [38].

Apesar das várias referências citadas que buscam melhorar o desempenho (em termos de número de avaliações da função objetivo) do PSO, a grande limitação deste algoritmo, assim como dos outros PBM's, é o elevado número de avaliações da função objetivo requerido. Como já discutido, aplicações em problemas práticos de engenharia tornam-se demasiadamente custosas ou mesmo inviáveis com a aplicação destes métodos devido ao alto custo computacional associado ao cálculo da função objetivo. Para contornar esse problema, o uso de computação paralela aparece como a alternativa mais viável atualmente.

2.4 Computação Paralela

A grande limitação para a utilização dos métodos não determinísticos, em particular os algoritmos baseados em comportamento de populações (PBM's) como os algoritmos genéticos e o de enxame de partículas, é, como já mencionado, o elevado número de avaliações da função objetivo. Quando o custo de avaliação desta função é elevado, a aplicação destes métodos pode conduzir a tempos de processamento extremamente elevados e no caso das funções que envolvem a simulação de problemas reais de engenharia a sua aplicação pode ser inviável. Exemplos desses problemas são o projeto de equipamentos que envolvem a simulação de sistemas fluidodinâmicos ou de processos químicos, problemas de estimação de parâmetros, controle e operação ótimo de plantas industriais, dentre diversos outros. Assim, de modo a utilizar esses métodos nestes tipos de problemas, a computação paralela ou processamento paralelo aparece entre as alternativas mais viáveis atualmente.

De acordo com SECCHI [39] o termo processamento paralelo inclui qualquer tipo de computador ou organização de sistema no qual operações múltiplas são executadas em paralelo. Essas organizações são comumente referenciadas pelos termos sistemas de multicomputadores, sistemas de multiprocessadores, processadores paralelos, processadores matriciais e processadores associativos.

Atualmente, com a disponibilidade cada vez maior de plataformas para computação paralela, a resolução desses problemas de otimização, assim como a resolução de muitos outros problemas de larga escala da engenharia, tornam-se viáveis e com custos relativamente baixos, incentivando o desenvolvimento de algorimtos e métodos adequados à utilização nestas plataformas.

Ratificando o descrito em SECCHI [39], a principal motivação para a utilização da computação paralela é a resolução de uma dada tarefa em P processadores utilizando $\frac{T_1}{P}$ unidades de tempo, a qual levaria T_1 unidades de tempo se fosse resolvida em um único processador. Essa escalabilidade perfeita é, entretanto, impossível[3] de ser alcançada devido ao desbalanceamento de carga entre os processadores, aos conflitos de acesso à memória em máquinas com memória compartilhada somado ao custo de transferência de mensagem via rede em máquinas com memória distribuída e à fração serial de processamento existente no código (SECCHI [39]; AMDAHL [40]; GUSTAFSON [41]).

[3]É impossível se os códigos paralelos são extensões paralelas dos seriais. Nos casos em que a implementação paralela altera a natureza do método, a escalabilidade perfeita, ou mesmo a sua superação, é possível

Os computadores paralelos, segundo FLYNN [42] (apud SECCHI [39]), podem ser classificados, baseados no número de execuções concorrentes, como:

1. SISD (Single Instruction, Single Data)
2. SIMD (Single Instruction, Multiple Data)
3. MISD (Multiple Instruction, Single Dat)
4. MIMD (Multiple Instruction, Mutliple Data)

Os computadores SISD são os computadores seriais convencionais, com fluxo único de instruções para operar em dados em memória, podendo ser superpostas nos seus estágios de execução.

Os computadores SIMD ou SPMD (Single Program, Multiple Data) apresentam várias unidades de operação que excutam uma única instrução por vez, mas esta opera sobre vários dados distintos, produzindo diversos resultados simultaneamente. São adequados para operação em conjunto de dados regulares como vetores e matrizes, operando simultaneamente e independentemente a mesma instrução sobre cada parte destas estruturas. Devido a isto, são chamados também de *Computadores Matriciais* ou *Computadores Vetoriais*. Já os computadores MISD ou MPSD fazem exatamente o contrário dos computares SIMD, ou seja, executam diversas instruções (ou programas) diferentes sobre o mesmo conjunto de dados.

Os computadores MIMD (ou MPMD) unem as características dos computadores SIMD e MISD. Consistem em um conjunto de unidades de processamento que executam programas de modo assíncrono e independente. Em qualquer instante da execução, processadores distintos podem executar instruções diferentes sobre diferentes conjuntos de dados. Os processadores (unidades de processamento) estão interligados para permitir a troca de informação e sincronismos das suas atividades (SECCHI [39]). A interconexão entre os processadores pode ser realizada através de memória compartilhada ou através de redes de comunicação de dados quando a memória é distribuída.

2.5 MPI - Message Passing Interface

A passagem de mensagens é um paradigma de programação para computação paralela. MPI fundamenta-se neste paradigma, constituindo-se de um conjunto de regras, especificações e funções escritas em Fortran, C e C++ para permitir que processos comuniquem-se pelo envio e pelo recebimento de mensagens em sistemas de computadores paralelos com memória compartilhada ou distribuída. Desde o primeiro fórum, iniciado em 1992, os principais objetivos têm sido tornar este padrão de elevado desempenho, escalável e portável sobre as diferentes plataformas computacionais. Em 1994, quando a primeira versão foi concluída, o conjunto de especificações e funções tornou-se um padrão para a comunição entre processos em computação paralela.

Muitas referências estão disponíveis. Nesta dissertação, SNIR *et al.* [43] foi a principal utilizada, a qual recomenda-se para qualquer tipo de usuário, desde iniciante até avançados. Há ainda referências disponíveis na internet como BARNEY [44], PACHECO [45] e PACHECO e MING [46].

A forma geral de um programa em linguagem C utilizando MPI pode ser sintetizada de acordo com a figura 2.1. O arquivo de cabeçalho *"mpi.h"* deve estar presente em todas as construções com MPI. Este arquivo contém todas as definições, macros e protótipos das funções necessárias para a sua compilação. Os processos em execução no programa MPI executam instruções de maneira independente no estilo *MIMD*, transferindo informações entre si através das funções de troca de mensagem.

O padrão pode utilizar comunicação de ponto a ponto (*point-to-point communication*) e/ou comunicação coletiva (*collective communication*). No primeiro caso, a comunicação envolve a transferência de mensagem entre um par de processos, um que envia a mensagem e o outro que a recebe. Essa comunicação pode ser feita entre processos dentro de um *comunicador*[4] específico (intra-comunicador) ou entre processos em diferentes *comunicadores* (inter-comunicadores). No caso da comunicação coletiva, a mensagem é transmitida entre todos os processos dentro de um único comunicador (intra-comunicador).

Nos algoritmos desenvolvidos neste trabalho somente funções de comunicação ponto a ponto (*point-to-point communication*) foram utilizadas. Este tipo de comu-

[4]Um comunicador especifica um domínio de comunicação paralelo. Define uma rede de processadores que poderão ser contactados durante a execução do programa paralelo.

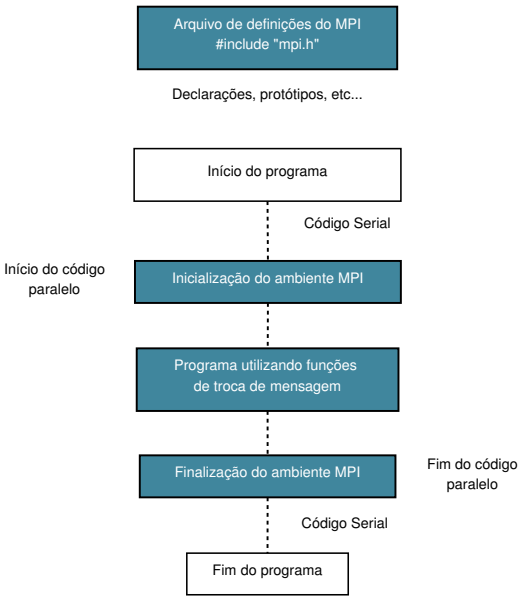

Figura 2.1: *Estrutura geral de programas utilizando MPI.*

nicação pode ser feito de dois modos distintos: a comunicação bloqueada (*blocking communication*) e a não bloqueada (*non blocking communication*).

Uma comunicação é dita bloqueada se o retorno da função permite que os recursos da sua chamada (dados em seus argumentos) podem ser re-utilizados. Em outras palavras, o programa só pode prosseguir quando as operações de troca de mensagem estão concluídas. No caso das funções de envio e recebimento de mensagens *bloqueadas*, o retorno da função ocorre apenas quando o conteúdo da mensagem em trânsito foi devidamente copiado para ou da memória intermediária do sistema (*"buffer"*)[5].

Em comunicações *não bloqueadas*, a função pode retornar antes que os procedimentos de comunicação estejam terminados. Após o retorno, é possível a utilização dos recursos da chamada, os quais podem ainda não terem sido completamente alterados. São necessários funções para verificar a conclusão da comunicação.

As chamadas de funções em MPI podem ser ainda *locais* ou *não locais*. No primeiro caso, a conclusão do procedimento depende apenas da execução no processo local, donde a função foi chamada. Em chamadas *não locais*, a finalização do procedimento em um processo requer algum procedimento do MPI sendo executado em outro processo.

[5]O *buffer* corresponde a uma memória intermediária do sistema que é alocada ou desalocada em tempo de execução do programa paralelo.

A seguir são detalhadas as principais funções do MPI (em linguagem C) utilizadas na implementação dos códigos deste trabalho.

void MPI_Init(int *argc, char *argv)

argc	**Entrada**. Número de entradas na linha de comando;
argv	**Entrada**. Vetor com as entradas da linha de comando.

Esta função inicia o programa a ser realizado em computação paralela. A partir de sua chamada, todas as definições, macros e funções do MPI poderão ser utilizadas. Os dois argumentos de entrada para a função de inicialização, são as variáveis de entrada por linha de comando da função principal do programa em C.

int MPI_Comm_rank(MPI_Comm Comm, int *rank)

Comm	**Entrada**. Comunicador (MPI_COMM_WORLD);
rank	**Saída**. Rank (identificação) dos processadores no comunicador.

A função MPI_Comm_rank() fornece no argumento *rank* a identificação de cada processador no comunicador *Comm*. Neste trabalho, o comunicador padrão do MPI, dado pela macro pré-definida MPI_COMM_WORLD, foi utilizado. Este comunicador, disponível após a inicialização do MPI, é constituído de todos os processadores que participam da computação paralela, os quais são definidos nas opções de execução do comando *mpirun*[6]. A função retorna um número inteiro, com um código de erro associado (SNIR *et al.* [43]).

[6] *mpirun* é o comando para execução de programas paralelos utilizando o MPI, em particular, na distribuições do OpenMPI utilizadas (versões 1.2.6 e 1.4.2).

int MPI_Send(void *msg, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm Comm)

msg	**Entrada**. Endereço inicial do *"buffer"* de envio;
count	**Entrada**. Número de *"entradas"* para enviar;
datatype	**Entrada**. Tipo de dado de cada entrada enviada para o *"buffer"*;
dest	**Entrada**. Processador de destino da mensagem;
tag	**Entrada**. Código de identificação da mensagem;
Comm	**Entrada**. Comunicador (MPI_COMM_WORLD);

A função MPI_Send() é *bloqueada* e potencialmente *não local*[7], retornando apenas quando o conteúdo da mensagem é completamente copiado no *buffer* de envio. Ela envia para o processador de *rank* igual a *dest* (pertencente ao comunicador MPI_COMM_WORLD) *count* dados do tipo *datatype* contido no argumento *msg*. A função guarda em um *"buffer"* do sistema o endereço do primeiro elemento da variável *msg* e aloca espaço na memória para *count* dados do tipo *datatype*. O *"buffer"* de envio consiste, portanto, de um espaço na memória do sistema suficiente para armazenar *count* dados do tipo *datatype*.

A mensagem é enviada com um respectivo *tag*, o qual fornece um código de identificação para a mesma. A função MPI_Send() retorna um número inteiro com o código de erro associado.

[7]Ver capítulo 2, página 32 de (SNIR *et al.* [43])

int MPI_Recv(void *msg, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm Comm, MPI_Status *status)

msg	**Saída**. Endereço inicial do *"buffer"* de recebimento;
count	**Entrada**. Número de *"entradas"* para receber;
datatype	**Entrada**. Tipo de dado de cada entrada recebida para o *"buffer"*;
source	**Entrada**. Processador de origem da mensagem;
tag	**Entrada**. Código de identificação da mensagem;
Comm	**Entrada**. Comunicador (MPI_COMM_WORLD);
status	**Saída**. Status do processador de origem.

A função MPI_Recv() recebe do processador de *rank* igual a *source*, pertencente ao comunicador MPI_COMM_WORLD, *count* dados do tipo *datatype* contido no *"buffer"* de recebimento. O endereço inicial do *"buffer"* é armazenado na variável *msg*. A mensagem é recebida com um determinado *tag*, que a caracteriza. O argumento *source* pode ser especificado como MPI_ANY_SOURCE, uma macro do MPI que permite que a função receba a mensagem de qualquer processo contido no comunicador.

O Status do processo é retornado na estrutura *status*, de tipo MPI_Status. Essa estrutura contém três campos: o MPI_SOURCE, o MPI_TAG e o MPI_-ERROR, os quais contém, respectivamente, a *fonte*, o *tag* e o *código de erro* da função.

A função MPI_Recv() também é uma função *bloqueada*. Ela retorna apenas quando o conteúdo da mensagem foi completamente copiado no *"buffer"* de recebimento do sistema.

void MPI_Finalize()

A função MPI_Finalize() finaliza a execução em paralelo. Após a sua chamada, nenhuma macro, operação e/ou quaisquer outros procedimentos definidos em *"mpi.h"* poderão ser utilizados.

2.6 Algoritmos Paralelos do Enxame de Partículas

Embora versões paralelas de alguns algoritmos de otimização como os algoritmos genéticos (GA's), o Simulated Annealing (SA) e os algoritmos baseados em gradiente já estejam bastante popularizadas e com uma vasta bibliografia, a paralelização do PSO é relativamente recente e ainda pouca explorada na literatura. De acordo com WAINTRAUB *et al.* [30], os algoritmos genéticos são os métodos baseados em população mais explorados em termos de computação paralela. Com relação a estes, alguns trabalhos podem ser citados, destacando-se aqui os trabalhos de ALBA e TROYA [47], ALBA *et al.* [48] e ALBA *et al.* [49]

O primeiro trabalho de paralelização do PSO foi desenvolvido por SCHUTTE *et al.* [3] que propõem uma implementação síncrona com a utilização da estratégia *mestre-escravo*, denominada por eles de PSPSO (Parallel Syncronous Particle Swarm Optimization). O desempenho do algoritmo é testado na otimização de um sistema biomecânico definido pelos mesmo autores em REINBOLT *et al.* [36]. Previamente à formulação do código, os autores discutem três questões que são consideradas por eles para o desenvolvimento do código paralelo: (i) *a simultaneidade das operações* e a *escalabilidade do algoritmo*, (ii) *a sincronia* e a *assincronia do algoritmo* e (iii) *a coerência da implementação*.

Com relação à *simultaneidade das operações*, elas devem ser facilmente decompostas em operações sobre as máquinas disponíveis no domínio de computação paralela. É desejável também que o algoritmo seja altamente escalável, ou seja, a sua natureza não deve limitar o número de nós computacionais que podem ser utilizados. No caso do PSO, as duas características são satisfeitas. O cálculo da função objetivo é realizado independentemente pelas partículas do enxame podendo, portanto, ser realizado de maneira paralela. Além disso, o número de nós computacionais que podem ser utilizados é uma unidade maior do que o tamanho da população de partículas ($N_{escravos}+1$). Assim sendo, o limite de escalabilidade é igual ao tamanho da população (apenas no caso em que o cálculo da função objetivo não é também paralelizado).

Por *sincronia* e *assincronia* das atualizações, os autores argumentam sobre a especulação de que atualizações assíncronas da velocidade e posição, as quais são realizadas imediatamente após o cálculo da função objetivo por cada partícula, tendem a fazer com que o enxame responda mais rapidamente a mudanças nos melhores

valores locais e globais encontrados pelas partículas e pelo enxame, diminuindo o número total de avaliações da função objetivo necessário para encontrar o ótimo. Assim sendo, esse modo de atualização deve ser preferido ao modo síncrono, no qual as atualizações são realizadas apenas uma vez a cada *revoada* ou *geração* do enxame. (CARLISLE e DOZIER [50]).

Ressalta-se desde já que, nos códigos desenvolvidos para este trabalho, essas duas formas de atualização supracitadas serão referenciadas, respectivamente, como *Immediate Update* e *Swarm Update*, de modo a diferenciar dos modos de troca de mensagens entre processadores mestre e escravos, os quais utilizarão a nomenclatura assíncrono ou síncrono.

Sobre a *coerência da implementação* os autores argumentam que a paralelização não deve afetar o resultado final obtido quando uma mesma semente de números aleatórios for utilizada. Ou seja, as operações do algoritmo devem ocorrer na mesma ordem da implementação do PSO serial original. Essa característica é preservada no código apresentado pelos autores, mas, como será visto mais adiante, ela só pode ser alcançada com um código paralelo síncrono com atualizações por revoada do enxame (*Swarm Update*), o que não corresponde ao algoritmo mais eficiente.

No algoritmo proposto pelos autores (SCHUTTE *et al.* [3]), o cálculo da função objetivo é realizado simultaneamente por $N_{escravos}$ partículas do enxame, sendo $N_{escravos}$ igual a uma unidade inferior ao total de processadores disponíveis no domínio de computação paralela, que deve ser no máximo igual ao número de partículas do enxame. Os passos básicos do algoritmo conforme discutido nesta referência são:

1. Inicialização
    a Especificar parâmetros do algoritmo;
    b Inicializar aleatoriamente as posições e as velocidades no espaço de busca;
    c Especificar k = 1.
2. Otimização
    a Calcular a função objetivo para todas partículas em paralelo utilizando suas posiçẽs atuais;
    b Criar a barreira de sincronização;
    c Atualizar os melhores valores das partículas;

d Atualizar o melhor valor do enxame;

e Testar o critério de parada;

f Se o critério de parada não foi satisfeito, atualizar as posições e velocidades de todas as partículas;

g Incrementar k e voltar para 2.

De acordo com o apresentado, o código desenvolvido por esses autores utiliza funções de troca de mensagens coletivas do MPI (collective communications) e uma função para criar uma barreira de sincronização das partículas. Neste ponto já pode ser observado um ponto fraco do modo de comunicação síncrono. *a necessidade de sincronização torna o código limitado pelo escravo mais lento.* Assim, o escravo que calcula a função objetivo nas coordenadas da partícula que levaria maior tempo de cálculo é o que fornece o tempo de cômputo da revoada.

De acordo com KOH *et al.* [4], a estratégia síncrona de paralelização utiliza de modo mais eficiente os recursos computacionais quando três condições são satisfeitas:

1. Acesso total e não dividido a máquinas em um cluster homogêneo;
2. Tempo de cálculo da função objetivo é constante e independente das coordenadas no espaço de busca;
3. Número de tarefas paralelas é igualmente dividida entre os processadores.

Segundo esses autores nenhuma dessas condições é, em geral, satisfeita, comprometendo o desempenho dos algoritmos síncronos. Principalmente a segunda condição, como mencionado anteriormente, faz com que os escravos que calculam mais lentamente a função objetivo limitem o tempo de processamento nessas estratégias.

Baseado nestas limitações de desempenho, KOH *et al.* [4] propuseram a primeira estratégia assíncrona de paralelização do PSO, denominada por eles de *PAPSO - Parallel Asynchronous Particle Swarm Optimization.* Nesta estratégia não há um ponto de sincronização das partículas. Ela se beneficia de uma característica do PSO para conseguir a assincronia do código: *o algoritmo não impõe a ordem e a quantidade de vezes que as partículas calculam a função objetivo.*

De posse desta característica é criada uma fila que armazena as partículas em ordem de chegada para posterior envio aos processadores escravos. Dentro de um loop de recebimento, atualização e envio, o processador mestre recebe o valor da função objetivo de um escravo, envia essa partícula recém chegada para o final da fila, atualiza as informações dos ótimos locais e do enxame e envia a próxima partícula da lista, com velocidade e posição atualizadas com os valores atuais dos ótimos, para o escravo que acabou de ficar disponível para cálculo. Os passos básicos do algoritmo executados pelos processadores mestre e escravo, conforme descrito na referência, são:

1. ***Processador Mestre***

    a Inicialização dos parâmetros do algoritmo e das velocidades e posições das partículas;

    b Criação de uma fila de partículas para serem calculadas nos processadores escravos;

    c Atualização das posições e das velocidades da partícula i baseado nas informações atuais dos ótimos locais e global do enxame;

    d Envio da posição atualizada ($\mathbf{x}_i$) de uma partícula i para cálculo em um processador escravo disponível;

    e Recebimento da informação (valor da função objetivo) de um processador escravo;

    f Teste do critério de convergência.

2. ***Processadores Escravos***

    a Recebimento do processador mestre da nova posição da partícula a ser calculada;

    b Cálculo da função objetivo nesta posição;

    c Envio para o mestre do valor calculado da função objetivo;

Essa implementação guarda uma diferença significativa em relação às implementações seriais e paralelas síncronas. Como a ordem de retorno das partículas depende do tempo que os processadores escravos levam para calcular a função objetivo, as partículas podem não pertencer, a princípio, à mesma *revoada* ou *geração*, eliminando, portanto, esse conceito do método. Isso faz com que a *coerência* destacada anteriormente não seja respeitada, ou seja, mesmo utilizando a mesma semente

de números aleatórios, no algoritmo assíncrono as partículas podem tomar caminhos diferentes, a depender da ordem de retorno das partículas pelos escravos.

Na Seção 2.3 foram discutidas algumas variantes do algoritmo de enxame de partículas, desenvolvidas com o principal intuito de melhorar o desempenho deste algoritmo, construindo algoritmos mais robustos e mais eficientes. Alguns desses trabalhos começam a desenvolver essas variantes para computação paralela. Neste sentido, KALIVARAPU *et al.* [26] desenvolveram uma estratégia síncrona de paralelização (semelhante à estratégia proposta por SCHUTTE *et al.* [3]) para um algoritmo PSO com feromônios digitais. Esses feromônios são adicionados como um termo extra ao lado direito da equação de atualização da velocidade das partículas (equação 2.7) e tem a função de emular a ação de agentes químicos deixados por insetos (os feromônios) no seu caminho à procura de alimentos. Segundo esses autores, a adição desses feromônios consegue melhorar o desempenho do enxame, diminuindo o número de avaliações da função objetivo na busca pelo ótimo global.

WAINTRAUB *et al.* [30] utilizaram a ideia de enxames cooperativos e desenvolveram uma estratégia de ilhas com vizinhança. Nesta estratégia, alternativa ao modelo de ilhas descrito no mesmo trabalho, é criada uma topologia de ilhas que comunicam-se e permitem a migração de indíviduos entre as mesmas. Cada ilha desenvolve um população independente de partículas e executam o algoritmo clássico do PSO (equação 2.7). Depois de um dado número de iterações do algoritmo, as melhores partículas de cada ilha migram para as suas vizinhas levando consigo toda a sua informação.

Os resultados apresentados mostram um substancial ganho de perfomance dessa estratégia em relação à *mestre-escravo*. Contudo, não é dito explicitamente que tipo de comunicação é realizada nesta última (se síncrona ou assíncrona) assim como não é mencionado qual o tipo de comunicação realizado entre as partículas em uma dada ilha do algoritmo. De uma leitura cuidadosa do artigo, as ilhas parecem ser construídas através de *inter-comunicadores* no ambiente MPI, os quais permitem a comunicação entre as ilhas através de função de troca de mensagens *ponto a ponto*. (Seção 2.5). Dentro de cada ilha (um *intra-comunicador*), funções de troca de mensagens *ponto a ponto* ou *coletivas* seriam possíveis, o que permitiria a implementação de qualquer variante do PSO. A despeito dessa observação, parece evidente que a ideia de enxames cooperativos melhora substancialmente a perfomance do PSO e deve, portanto, ser melhor estudada.

Capítulo 3

Otimização por enxame de partículas

Nas Seções 2.3 e 2.6 foi realizada uma revisão detalhada do algoritmo do enxame de partículas e das suas versões paralelas. Neste capítulo, será feito o detalhamento dos códigos implementados neste trabalho, com particular ênfase aos códigos desenvolvidos para a computação paralela. Na Seção 3.1 a implementação dos algoritmos seriais IU–SPSO e o SU–SPSO são detalhadas. Na Seção 3.2 são apersentados os algoritmos paralelos síncronos (IU–PPSO e SU–PPSO) e, na Seção 3.3, o algoritmo paralelo assíncrono (AIU–PPSO).

3.1 Algoritmos seriais: IU–SPSO e SU–SPSO

Os algoritmos implementados neste trabalho utilizam a forma de SHI e EBERHART [18] para a atualização da velocidade e posição das partículas. Essa forma, chamada neste trabalho de PSO-Clássico, é dada pelas equações 3.1 e 3.2 de atualização da velocidade e posição das partículas.

$$\mathbf{v}_i^{k+1} = w\mathbf{v}_i^k + c_1 r_1[\mathbf{x}_{m,i}^k - \mathbf{x}_i^k] + c_2 r_2[\mathbf{x}_g^k - \mathbf{x}_i^k] \tag{3.1}$$

$$\mathbf{x}_i^{k+1} = \mathbf{x}_i^k + \mathbf{v}_i^k \tag{3.2}$$

O peso de inércia é atualizado de maneira decrescente com o número de *revoadas* já realizadas pelo algoritmo, conforme mostra a equação 3.3. Quando o número de revoadas é pequeno, o peso de inércia é próximo à w_0. Na medida que o número de revoadas cresce, o peso de inércia tende hiperbolicamente para w_f.

$$w = w_0 + (w_f - w_0)\frac{\alpha}{N_{max} + \alpha} \tag{3.3}$$

N_{max} é o número máximo de permanência no ótimo (valor padrão igual a $4N_{min}$, onde N_{min} é o número de permanência no ótimo) e α é igual a zero no início do processo de busca, sendo incrementado de uma unidade ao final de cada revoada se o critério de permanência do ótimo for satisfeito. Este critério, o *Critério 1* de convergência, será devidamente definido na Seção 4.1.

Os parâmetros cognitivo (c_1) e social (c_2) são considerados constantes. Sempre que não for dito explicitamente, o valor padrão de 1,5 é utilizado para esses parâmetros. Identicamente, para os pesos de inércia inicial (w_0) e final (w_f) são utilizados os valores padrões de 1,0 e 0,1, respectivamente.

Duas versões seriais do algoritmo de enxame de partículas foram implementadas neste trabalho: o IU–SPSO (*Immediate Update Serial PSO*) e o SU–SPSO (*Swarm Update Serial PSO*). As duas formas são essencialmente idênticas em termos de implementação, diferindo entre si apenas no modo de atualização da velocidade e da posição das partículas. Enquanto na primeira essa atualização é realizada imediatamente após o cálculo da função objetivo por uma determinada partícula, no segundo é realizada apenas no final de cada *revoada*.

A implementação de versões operacionais desses algoritmos é bastante simples. Após inicialização das posições e velocidades das partículas no espaço de busca (a qual é feita, normalmente, de maneira aleatória), as partículas entram em um *loop* no qual suas posições e velocidades são continuamente atualizadas de acordo com a equação básica do método (equações 3.1 e 3.2).

Dentro de cada *revoada* é calculado o valor da função objetivo para cada partícula. No caso do IU–SPSO, imediatamente antes a esse cálculo, a velocidade e a posição de cada partícula são atualizadas de acordo com os melhores valores das partículas e do enxame, previamente calculados. No caso do SU–SPSO, o valor da função objetivo é calculado para todas as partículas nas suas atuais posições. As

atualizações dos melhores valores das partículas e do enxame são feitas ao fim de cada *revoada*, ou seja, quando todas as partículas concluirem os seus cálculos.

Ao final de cada *revoada* é realizado um teste sobre o valor atual do ótimo global encontrado pelo enxame (o *Critério 1*). Sempre que esse valor se repetir dentro de uma determinada tolerância o valor do contador (k) é incrementado. Se o valor do ótimo se repetir por N_{min} vezes (denominado de número de permanência), então o critério de convergência do algoritmo é testado. O critério de convergência utiliza uma determinada métrica sobre a disposição ou deslocamento das partículas no espaço de busca, sendo considerado satisfeito de acordo com a tolerância especificada. Essas métricas, os critérios de convergência do método, serão dicutidas na Seção 4.2.

As figura 3.1 e 3.2 mostram os fluxogramas das implementações dos dois algoritmos seriais. Nos fluxogramas, N_{aval} e Max_Aval são o número de avaliações da função objetivo já realizadas pelo enxame e o número máximo de avaliações permitidos no algoritmo. Ainda, as variáveis $\mathbf{x}_{m,i}$, $\mathbf{x}_g$ são, respectivamente, a melhor posição já encontrada pela partícula i no espaço, a melhor posição já encontrada pelo o enxame e $f_{m,i}$ e f_g são os valores da função objetivo relacionado a esses pontos.

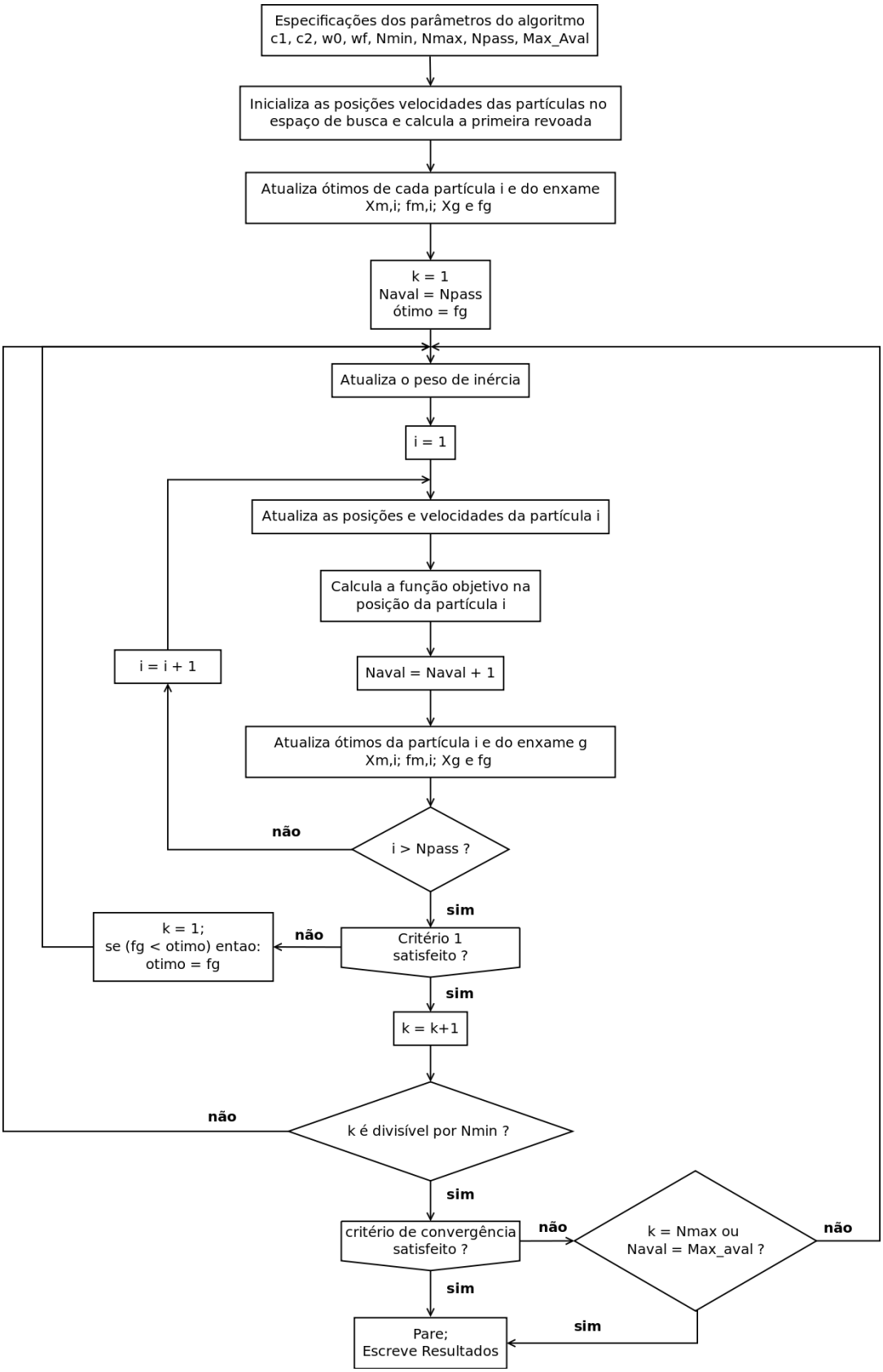

Figura 3.1: *Fluxograma da implementação do algoritmo serial IU–SPSO.*

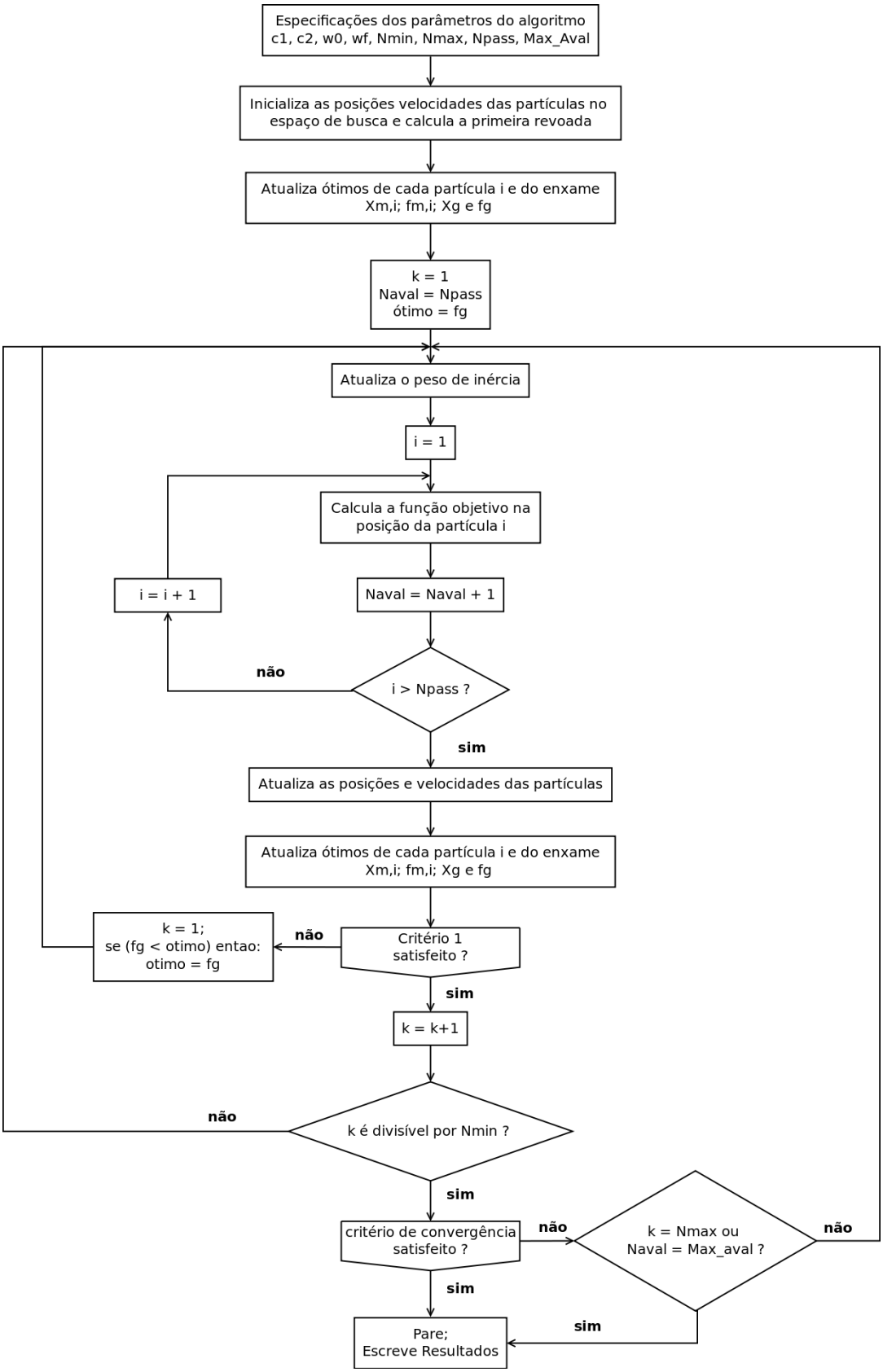

Figura 3.2: *Fluxograma da implementação do algoritmo serial SU–SPSO.*

3.2 Algoritmos paralelos síncronos IU–PPSO e SU–PPSO

Os algoritmos paralelos desenvolvidos neste trabalho utilizam o paradigma *mestre–escravo* para a comunicação de dados entre os processadores. Nesta estratégia, um processador mestre coordena todas as etapas do algoritmo e gerencia o envio da informação necessária (a posição das partículas no espaço de busca) para os processadores escravos, os quais calculam o valor da função objetivo. Quando calculado este valor, os escravos o retornam ao mestre, para que esse dê prosseguimento ao código.

Os códigos paralelos síncronos (IU–PPSO e SU–PPSO) podem ser encarados como simples extensão dos códigos seriais para computação paralela. São essencialmente idênticos em sua natureza, com a diferença de que a função objetivo é calculada paralelamente aos demais cálculos dos algoritmos.

3.2.1 Immediate Update Parallel PSO (IU–PPSO).

A figura 3.3 mostra o fluxograma da implementação do algorimto IU–PPSO. Para a comunicação entre os processadores mestre e escravos são utilizadas funções de comunicação ponto a ponto do MPI, particularmente as funções $MPI\_Send()$ e $MPI\_Recv()$, detalhadas na Seção 2.5. No fluxograma, as setas com *linha tracejada* ou *linha pontilhada* que *chegam* ou *deixam* os blocos têm precedência sobre as setas com *linha contínua* que *deixam* os mesmos. Essas *setas* indicam a execução das funções que realizam a transferência de mensagens entre os processadores.

Ao bloco que representa o conjunto de escravos *chegam* duas setas. A com *linha pontilhada* representa o envio da primeira ordem de cálculo aos escravos dentro de cada *revoada*. Esta operação é realizada *apenas uma vez* para cada revoada do algoritmo. Já a seta com *linha tracejada* representa a operação de troca de mensagen entre o mestre e *apenas um* escravo, representado no bloco pelo escravo genérico *"p"*.

A etapa de inicialização é semelhante à dos algoritmos seriais, com a diferença de que é realizada em modo paralelo, utilizando os processadores escravos disponíveis. O pseudo–código 1 apresenta a etapa de inicialização aleatória das partículas. Alternativamente, o código aceita a inicialização por arquivo de dados,

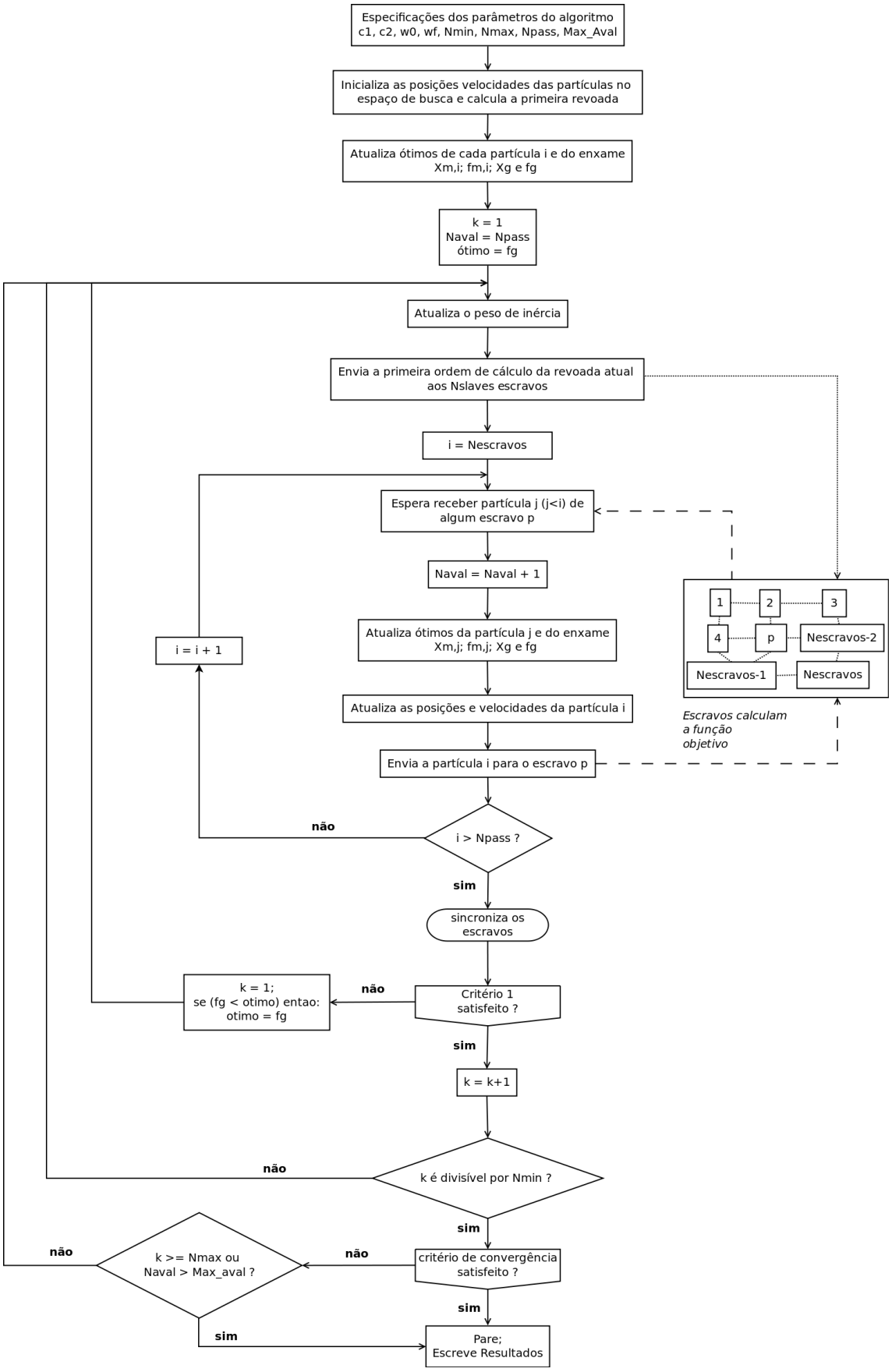

Figura 3.3: *Fluxograma da implementação do algoritmo IU–PPSO.*

Pseudo–código 1 *Pseudo código da etapa de inicialização aleatória das partículas nos algoritmos paralelos*

Require: $w_0, w_f, c_1, c_2, N_{pass}, \mathbf{X}_{min}, \mathbf{X}_{max}, N_{min}, N_{max}$ e Max_Aval
  for $(i = 1 \rightarrow N_{pass})$ **do**
    $\mathbf{x}_i \leftarrow \mathbf{X}_{min} + rand()[\mathbf{X}_{max} - \mathbf{X}_{min}]$
    $\mathbf{x}_{m,i} \leftarrow \mathbf{x}_i$
    if $(i < N_{escravos})$ **then**
      Envia $\mathbf{x}_i$ para o escravo i
    else
      Recebe F_{obj} referente à partícula j calculada pelo escravo *"p"*
      $F_{m,j} \leftarrow F_{obj}$
      Envia $\mathbf{x}_i$ para o escravo *"p"*
    end if
  end for
  Sincroniza os escravos

onde deve ser fornecido (no mínimo) o estado das partículas, ou seja, a posição, a velocidade e o valor da função objetivo associado. Ainda, o código pode ser reinicializado a partir de uma solução prévia, que é escrita em arquivo sempre que o usuário interrompe o código em execução. No fim da etapa de inicialização os escravos são sincronizados, retornando e atualizando as últimas $N_{escravos}$ partículas calculadas.

Antes da etapa de otimização em si, os valores ótimos da primeira revoada (população inicial) são atulizados. O pseudo–código 2 apresenta o procedimento para determinação do *melhor valor* da primeira revoada.

Pseudo–código 2 *Pseudo código da etapa de determinação do melhor valor da primeira revoada*

  $i \leftarrow 0$
  for $(j = 1 \rightarrow N_{pass})$ **do**
    if $(f_{m,j} - f_{m,i} < 0)$ **then**
      $i \leftarrow j$
    end if
  end for
  $f_g \leftarrow f_{m,i}$
  $\mathbf{x}_g \leftarrow \mathbf{x}_i$

A etapa de otimização inicia com o envio da primeira ordem de cálculo aos escravos dentro de cada *revoada*, que corresponde ao envio das $N_{escravos}$ primeiras partículas a serem calculadas, para os $N_{escravos}$ disponíveis, identicamente ao realizado na etapa de inicialização. Após o envio dessa primeira ordem de cálculo o algoritmo entra em um *loop* no qual executa as seguintes funções:

1. Recebe o valor da função objetivo referente à partícula j, calculado por um

escravo genérico $''p''$;

2. Atualiza os melhores valores da partícula j ($\mathbf{x}_{m,j}$ e $f_{m,j}$) e do enxame ($\mathbf{x}_g$ e f_g);

3. Atualiza a posição ($\mathbf{x}_i$) e a velocidade ($\mathbf{v}_i$) da próxima partícula a ser calculada;

4. Envia a nova posição ($\mathbf{x}_i$) para o escravo $''p''$ que acabou de ficar disponível.

Nos códigos desenvolvidos neste trabalho, o recebimento no processador mestre é feito utilizando a fonte MPI_ANY_SOURCE, pois, não se sabe, *a priori*, o tempo de cálculo da função objetivo nos escravos. Assim sendo, a fonte de recebimento da mensagem é qualquer escravo que tenha terminado o cálculo da função objetivo no momento da requisição pelo mestre. A identificação do escravo no mestre (estritamente necessária no código) é feita, portanto, pelo acesso do respectivo campo na estrutura *status*, ou seja, status.MPI_SOURCE. O pseudo código 3 detalha o procedimento de otimização descrito acima.

Pseudo–código 3 *Pseudo código da etapa otimização do algoritmo paralelo síncrono IU–PPSO*

```
N_aval ← N_pass
otimo ← f_g
k ← 0
Revoada ← 1
while (k < N_max e N_aval < Max_aval) do
   Atualiza w com a equação 3.3
   for (i = 1 → N_pass) do
      if (i < N_escravos) then
         Atualiza x_i e v_i da partícula i
         Envia x_i para o escravo i
      else
         Recebe F_obj referente à partícula j calculada pelo escravo p
         N_aval ← N_aval + 1
         if (F_obj < f_m,j) then
            f_m,j ← F_obj
            x_m,i ← x_j
         end if
         if (F_obj < f_g) then
            f_g ← F_obj
            x_g ← x_j
         end if
         Atualiza (v_i) e (x_i) da partícula i com equações 3.1 e 3.2
         Envia x_i para o escravo p
      end if
   end for
   Sincroniza os escravos
   Revoada ← Revoada + 1
   Testa a convergência (ver Pseudo código 4)
end while
```

A sincronização é feita pelo recebimento das últimas $N_{escravos}$ partículas calculadas pelos escravos ao final de cada revoada para teste do critério de convergência, o qual é realizado com a função de $MPI\_Recv()$, não sendo necessário uma função de sincronização coletiva do MPI.

O teste de convergência é realizado em duas etapas. Na primeira é verificado se o ótimo global da revoada atual *mudou* em relação ao da revoada anterior. Em caso afirmativo, o contador k (contador da revoada) é incrementado, do contrário ele é reiniciado. A segunda etapa é realizada somente quando o contador k se torna múltiplo inteiro de N_{min} (o número de permanência). Neste caso, uma métrica (**d**) sobre a disposição ou deslocamento das partículas no espaço de busca é calculada. Se esta métrica atende a uma tolerância especificada, o algoritmo é considerado convergido. Três métricas diferentes foram desenvolvidas e testadas neste trabalho, sendo apresentadas na Seção 4.2 do próximo capítulo. O pseudo código 4 mostra a etapa do teste de convegência supracitado.

Pseudo–código 4 *Pseudo código do teste de convergência do algoritmo paralelo síncrono IU–PPSO*

```
while (k < N_max e N_aval < Max_aval) do
  ..... (ver Pseudo Código 3) .....
  if [(otimo − f_g) < ε_a] then
    k ← k + 1
  else
    k ← 1
  end if
  if (k/N_min ∈ ℵ) then
    cálcula d
    if (‖ d‖ < ε_a) then
      break
    end if
  end if
end while
Envia a ordem de parada para os escravos
```

O pseudo–código 5 apresenta a tarefa executada nos escravos. A função F neste pseudo–código é a expressão matemática, o resultado da simulação de um modelo ou qualquer outro calculador da função objetivo.

Pseudo–código 5 *Pseudo–código da etada de cálculo da função objetivo pelos escravos*

```
while (não receber a ordem de parada) do
  Recebe x_i do mestre
  F_obj = F(x_i)
  Envia F_obj para o mestre
end while
```

3.2.2 Swarm Update Parallel PSO (SU–PPSO).

A única diferença entre os algoritmos SU–PPSO e o IU–PPSO está na forma de atualização dos ótimos globais e da velocidade e posição das partículas. O pseudo–código 6 (uma alteração simples do pseudo–código 3) apresenta a única etapa do SU–PPSO diferente do IU–PPSO. A figura 3.4 mostra o fluxograma da implementação do algorimto SU–PPSO.

Pseudo–código 6 *Pseudo código da etapa otimização do algoritmo paralelo síncrono SU–PPSO*

```
N_aval ← N_pass
otimo ← f_g
k ← 0
Revoada ← 1
while (k < N_max e N_aval < Max_aval) do
  Atualiza w com equação 3.3
  for (i = 1 → N_pass) do
    Atualiza x_i e v_i da partícula i
    if (i < N_escravos) then
      Envia x_i para o escravo i
    else
      Recebe F_obj referente à partícula j calculada pelo escravo p
      N_aval ← N_aval + 1
      if (F_obj < f_m,j) then
        f_m,j ← F_obj
        x_m,i ← x_j
      end if
      Envia x_i para o escravo p
    end if
  end for
  Sincroniza os escravos
  Revoada ← Revoada + 1
  Atualiza x_g e f_g (Ver pseudo código 2)
  Atualiza v e x de todas as partículas com as equações 3.1 e 3.2
  Testa a convergência (ver Pseudo código 4)
end while
```

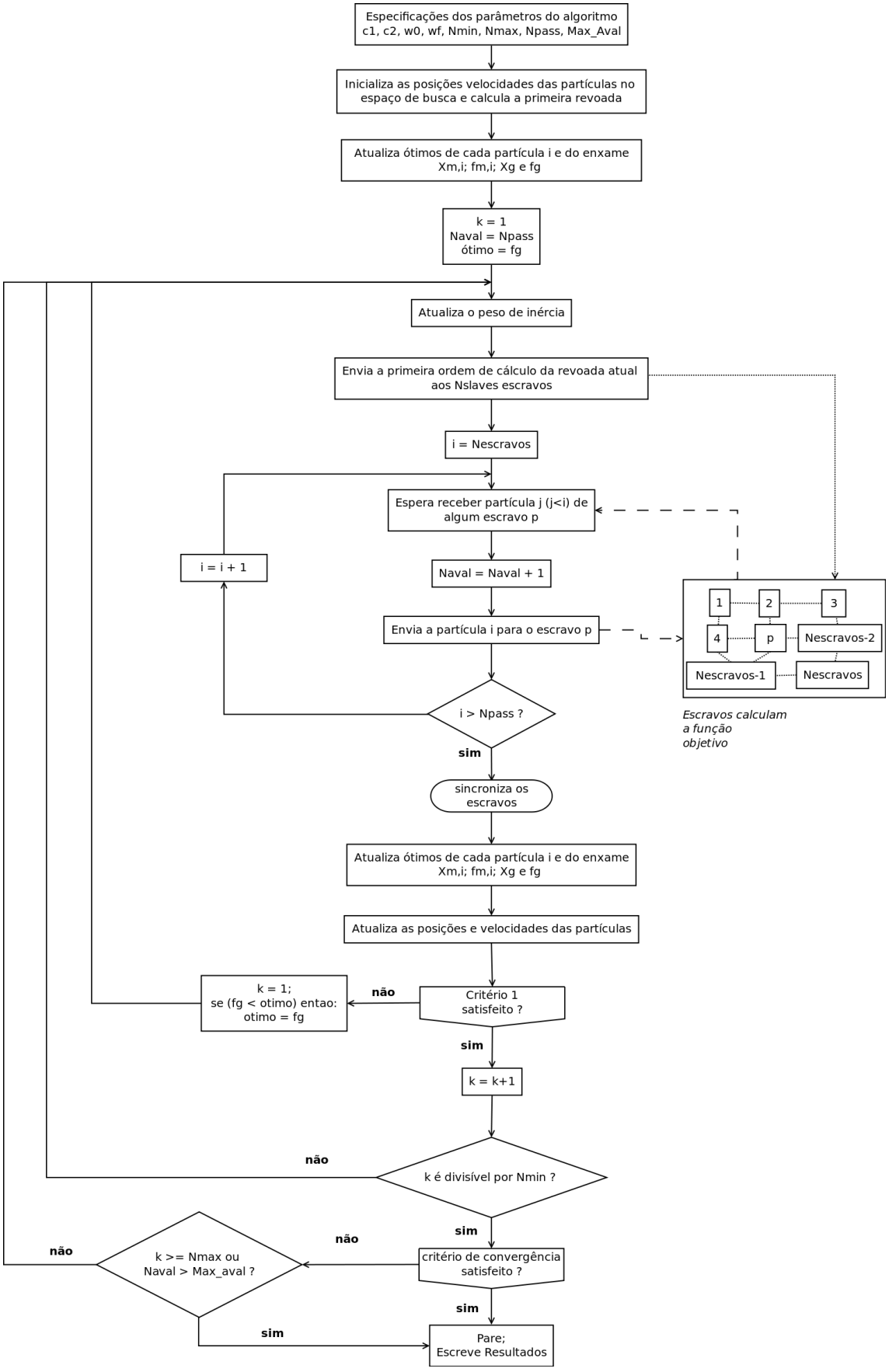

Figura 3.4: *Fluxograma da implementação do algoritmo SU–PPSO.*

3.3 Algoritmo paralelo assíncrono AIU–PPSO

A análise dos algoritmos síncronos deixa evidente um ponto fraco nesta estratégia de paralelização: *A necessidade de sincronização torna o código limitado pelo escravo mais lento.* Ou seja, no ponto de sincronização (final da revoada), o processador mestre deve esperar que todos os escravos terminem de calcular o valor da função objetivo para que ele possa prosseguir com os cálculos.

Conforme descrito em SCHUTTE *et al.* [3] (ver Seção 2.6) esses algoritmos devem exibir o melhor desempenho sob as seguintes condições:

1. Execução em cluster homogêneo;
2. Custo de cômputo da função é homogêneo;
3. Tarefas paralelas podem ser igualmente distribuídas entre os escravos.

O não cumprimento dessas condições tende a gerar *desbalanceamento de carga*, degradando o desempenho desses algoritmos. Principalmente as condições 2 e 3 não são geralmente satisfeitas em problemas práticos de engenharia, onde o custo da função objetivo pode ser fortemente dependente do conjunto de argumentos ou constantes do modelo sendo resolvido. Ainda, mesmo em clusters homogêneos o tempo de cálculo é distinto entre os processadores, embora, neste caso, este efeito seja minimizado.

Para contornar esses problemas, uma estratégia assíncrona de comunicação de dados entre os processadores foi implementada. Esta estratégia contém uma alteração fundamental em relação aos algoritmos síncronos: *não há ponto de sincronização dos escravos.* Assim, o conceito de *revoada* é destruído, aparecendo em seu lugar o conceito de *pseudo–revoada.*

A *pseudo–revoada* consiste em saídas controladas (a cada N_{pass} avaliações da função objetivo) pelo processador mestre do trecho de código onde ocorre a execução da otimização para testar a convergência do método. Esta saída é realizada, entretanto, sem que seja necessário interromper o cálculo da função objetivo nos escravos, equivalente ao ponto final da *revoada* nos algoritmos paralelos síncronos e seriais. Assim, a única *perda de tempo* significativa devido ao processo de paralelização no algoritmo assíncrono ocorre quando o escravo deseja fazer comunicação

com o mestre no momento em este está executando outros trechos do código. Nestas situações o escravo fica momentaneamente ocioso. Quando o cálculo da função objetivo é a etapa que demanda maior tempo computacional, consumindo um tempo muito maior do que o processamento do código no mestre, essa perda de tempo é desprezível.

Contudo, esta *ociosidade* pode ser minimizada pelo uso de funções do MPI (particularmente a função $MPI\_Probe()$) que indicam o momento exato em que um determinado processo deseja fazer uma comunicação. Com esta função, juntamente com os outros critérios que se julgarem importantes, pode–se determinar o instante propício para o término da *pseudo–revoada*.

Outra característica importante desta estratégia é a ordem de cálculo das partículas. Para conseguir a assincronia desejada, as partículas são dispostas em uma fila do tipo *FIFO (First In First Out)*. Assim, como o envio de mensagens dos escravos para o mestre depende do tempo de cálculo da função objetivo, as partículas poderão pertencer à *pseudo–revoadas* distintas, a depender da sua ordem de entrada na fila durante a execução do código. Possivelmente, em algumas execuções, os escravos poderão calcular números diferentes de avaliações da função objetivo. Isso faz com que o código assíncrono não mantenha a *coerência* descrita em SCHUTTE *et al.* [3].

A Seção 3.3.1 faz uma breve revisão sobre listas encadeadas circulares, as quais são utilizadas para criar uma fila *FIFO* conforme supracitado. O algoritmo assíncrono implementado, o AIU–PPSO , é detalhado na Seção 3.3.2.

3.3.1 Listas encadeadas

Listas encadeadas são estruturas de dados dinâmicas que crescem ou diminuem na medida em que novos elementos (os nós da lista) são adicionados ou removidos. Para cada elemento que é inserido ou removido na lista, o espaço na memória necessário para armazená–lo é dinamicamente alocado ou desalocado.

Diferentemente de vetores, os elementos (ou nós) da lista não ocupam um espaço contíguo na memória do sistema. Assim, para percorrer os nós de uma lista é necessário guardar explicitamente o seu encadeamento. Isso é feito armazenando junto com as informações dos nós um ponteiro para o próximo elemento da lista.

Os nós de uma lista encadeada são as estruturas de dados constituintes da lista e, como mencionado acima, devem conter, além da informação desejada, o endereço do seu vizinho na lista. Isso é realizado, definindo–se, dentro do nó, um ponteiro para uma estrutura de mesmo tipo, o que faz desses elementos estruturas de dados auto referenciadas. Em C, os nós de uma lista encadeada apresentam a declaração típica, dada pelo pseudo-código 7, onde declarou–se uma estrutura de nome *nome_da_estrutura*, que contém as variáveis de ponto flutuante de dupla precisão x, y e z, a variável inteira id e um ponteiro para a estrututura de mesmo tipo.

Pseudo–código 7 *Declaração dos elementos de uma lista encadeada*

```
typedef struct nome_da_estrutura
{
  double x, y, z;
  int id;
  struct nome_da_estrutura *próximo;
} Lista;
```

A diretiva **typedef** serve para definir um novo tipo de dado em C, no caso do exemplo o tipo *Lista*, que pode passar a ser utilizado para declarar nos programas variáveis do tipo *nome_da_estrutura*.

As listas encadeadas podem ser abertas ou circulares. No primeiro caso, a lista é representada por um ponteiro para o seu primeiro elemento, também chamado de cabeça da lista. O primeiro elemento guarda o endereço do segundo elemento, que guarda o do terceiro e assim por adiante até o último nó, que por sua vez aponta para um endereço *vazio* na memória, em C representado por **NULL**. No segundo caso, a lista continua sendo representada por um ponteiro para o seu *primeiro* elemento enquanto que o *último* passa a apontar para o endereço deste *primeiro*, fechando a lista. Neste caso, entretanto, não faz sentido estrito falar em primeiro ou último, pois quaisquer elementos da lista poderiam ser tratados como tal. Contudo, pode–se definir como primeiro elemento o nó que representa a lista e o último passa a ser, naturalmente, o elemento que aponta para o primeiro. A figura 3.5 mostra as estruturas típicas de listas encadeadas abertas e circulares.

A partir da definição da lista e de seus nós, podem–se definir as funções necessárias para a sua implementação. Nesta seção serão apresentadas as principais funções da lista encadeada circular, que foram utilizadas nos códigos desenvolvidos neste trabalho: as funções *de inicialização*, *de inserção*, *de remoção* e *de liberação*.

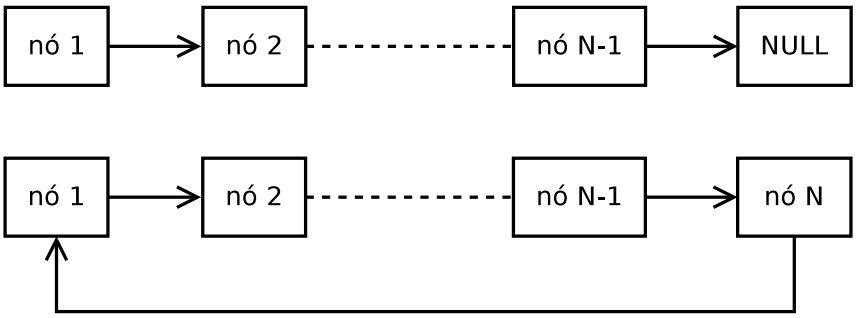

Figura 3.5: *figuras representativas de listas encadeadas. Acima listas abertas e abaixo listas circulares*

Função de inicialização

É a função necessária para inicializar a lista encadeada, seja ela aberta ou circular. Sendo a lista representada pelo ponteiro para o seu primeiro elemento, no início, como a lista não contém elemento, esse ponteiro deve ser inicializado. Isto pode ser feito apontando o ponteiro para um endereço *vazio* na memória. O código 8 mostra a função de inicialização, onde o tipo *Lista* foi pré–definido no trecho de código 7.

Pseudo–código 8 *Implementação da função de inicialização da lista encadeada.*

```
Lista *Initialize(void)
{
  return NULL
}
```

Função de inserção

Para cada elemento que será inserido na lista, é preciso alocar dinamicamente a memória necessária para armazená–lo e, após completá–lo com sua informação, encadeá–lo na lista. No caso da construção da lista *FIFO* utilizada neste trabalho, o encadeamento foi feito pela inserção do elemento na *cabeça* da lista, o qual passa a representá-la. Para fechar a lista, o atual *cabeça* da lista (novo elemento inserido) passa a apontar para o cauda da lista e a ser apontado pelo antigo cabeça.

A figura 3.6 mostra a sequência do processo de encadeamento realizado na função de inserção, em que o elemento é sempre inserido na cabeça da lista.

O pseudo-código 9 mostra a implementação da função de inserção. O código apresenta a inserção de um nó na cabeça na fila, cuja informação é o número inteiro *id*.

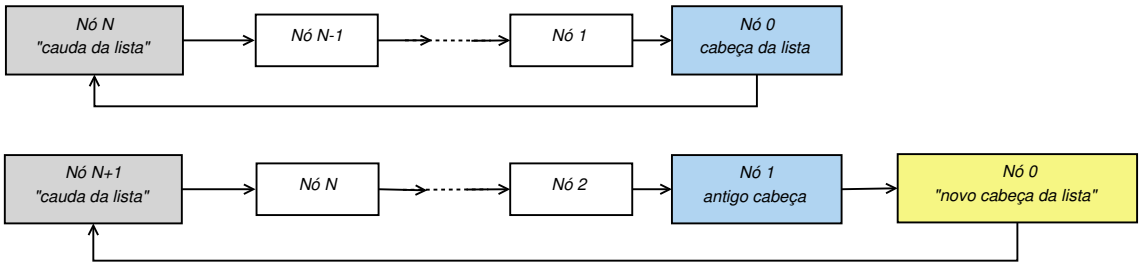

Figura 3.6: *Processo de encadeamento na inserção de um novo elemento na lista circular. Acima a lista original e abaixo após a inserção de um novo nó.*

Pseudo–código 9 *Pseudo código da implementação da função de inserção de nós na lista encadeada circular*

```
Lista *Insert(Lista *list, int id)
{
Lista *novo;
novo = (Lista*)calloc(1, sizeof(Lista));
novo->id = id;
if (list == NULL) then
   novo->prox = novo;
   return novo;
else
   novo->prox = list->prox;
   list->prox = novo;
   return novo;
end if
}
```

Função de remoção

De maneira contrária à inserção, o elemento é removido da cauda lista *FIFO*, o qual foi o primeiro elemento inserido na lista dentre todos os atuais elementos contidos na mesma. Ou seja, a lista utilizada sempre insere um elemento na cabeça da lista e remove o elemento da cauda. A figura 3.7 ilustra o procedimento de remoção implementado. O pseudo-código 10 mostra a implementação da função de remoção.

Função para liberar a lista

A função para liberar a lista tem como objetivo desalocar toda a memória alocada para os elementos ainda presentes na lista. O procedimento básico consiste em percorrer os elementos da lista um a um, liberando a memória deles sucessivamente. O psedo–código 11 mostra a implementação da função de liberação da memória.

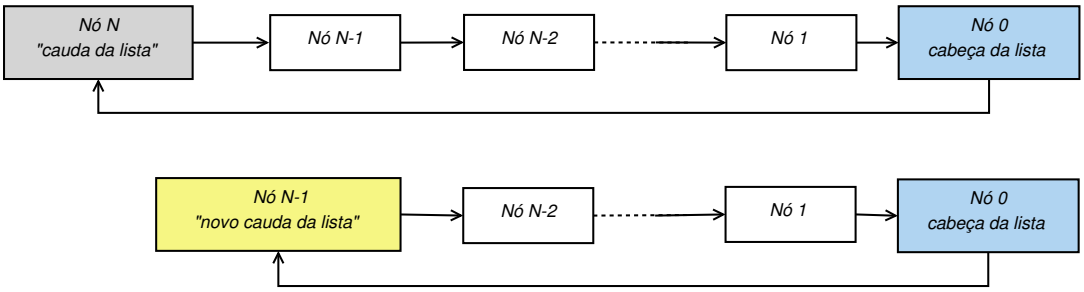

Figura 3.7: *Procedimento de remoção na lista FIFO".*

Pseudo–código 10 *Implementação da função de remoção de nós na lista encadeada circular*

```
Lista *Remove(Lista *list)
{
Lista *pt;
/*1. Se a lista está vazia*/
if (list == NULL) then
  return list;
end if
pt = list->prox;
list->prox = pt->prox;
free(pt);
return list;
}
```

Pseudo–código 11 *Implementação da função para liberar a memória alocada na lista encadeada circular*

```
void Libera(Lista *list)
{
Lista *pt = list->prox;
Lista *buf ;
if (list == NULL) then
  return;
end if
while (pt != list) do
  buf = pt;
  pt = pt->prox;
  free(buf);
end while
free(list);
}
```

3.3.2 AIU–PPSO. Asynchronous and Immediate Update Parallel PSO

A grande vantagem do AIU–PPSO em relação aos algoritmos síncronos reside na inexistência do ponto de sincronização dos escravos ao final de cada *pseudo–revoada*, conforme descrito na Seção 3.3. Com isso, este algoritmo deixa de ser limitado na *pseudo–revoada* ao escravo mais lento e todo o trecho de código executado pelo processador mestre (fora da *pseudo–revoada*) pode seguir sem que seja necessário a interrupção do cálculo da função objetivo nos escravos. Além disso, a assincronia parece fornecer informação adicional ao enxame, melhorando o seu desempenho em termos de número médio de avaliações da função objetivo. As figura 3.8 e 3.9 mostram o fluxograma de implementação do AIU–PPSO, que foi separado em dois para facilitar a sua leitura. As etapas de inicialização aleatória, de determinação do melhor valor da primeira *revoada*, da avaliação do critério de convergência e do cálculo nos escravos são idênticas às apresentadas nos pseudo–códigos 1, 2, 4 e 5.

Em termos de implementação, o AIU–PPSO difere do IU–PPSO apenas na etapa de otimização. Por não haver ponto de sincronização, apenas uma ordem de cálculo global é necessária antes do início do procedimento de otimização, enquanto que nos algoritmos síncronos, a mesma é realizada no início de cada *revoada*. A assincronia permite ainda, como já descrito anteriormente, que todo o trecho de cálculo realizado fora da *pseudo–revoada* seja realizado pelo *mestre* sem que seja necessário interromper o cálculo da função objetivo nos escravos. Por fim, a *liberação* dos escravos é, por consequência (já que apenas uma ordem de cálculo global foi realizada no início), realizada apenas uma vez, imediatamente antes do envio da ordem de parada. O pseudo–código 12 apresenta a etapa de otimização no AIU–PPSO.

A assincronia é obtida pela utilização da fila *FIFO*, a qual é construída com a lista encadeada circular. O final da *pseudo–revoada*, no código implementado, acontece a cada N_{pass} avaliações da função objetivo, embora seja possível, alternativamente e/ou conjuntamente, utilizar-se funções do MPI (em particular as função MPI_Probe() ou MPI_IProbe()) que indicam quando um processador está disponível para fazer comunicação, ou seja, quando há informação disponível para acesso na memória. Esta possilidade não foi implementada, mas pode ajudar a diminuir o tempo ocioso no escravo, aumentando a eficiência do código. Contudo, é de se esperar que este efeito seja significativo apenas quando o tempo de cálculo da função objetivo não for muito maior do que o tempo de processamento no mestre.

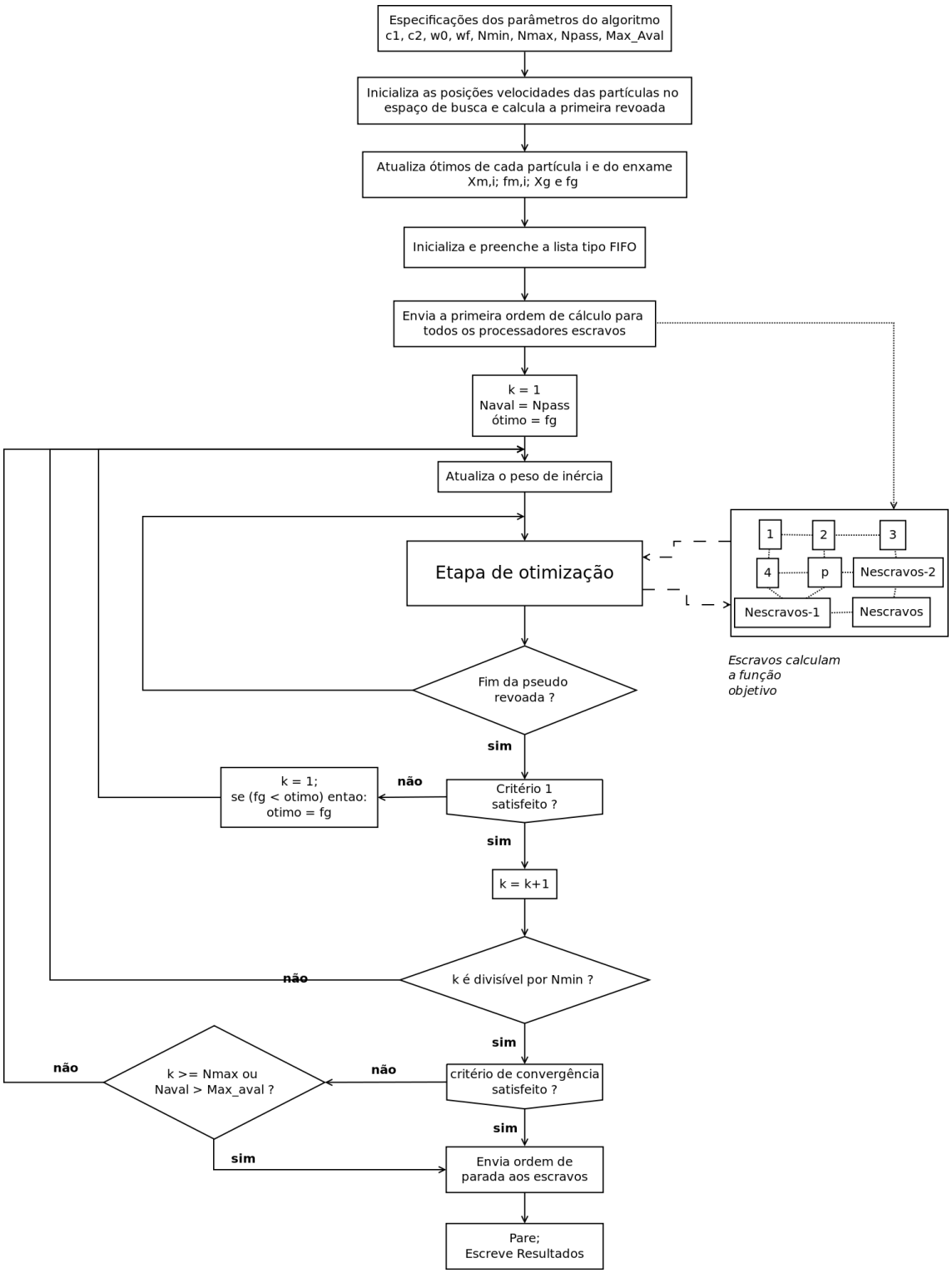

Figura 3.8: *Fluxograma geral da implementação do algoritmo assíncrono AIU–PPSO.*

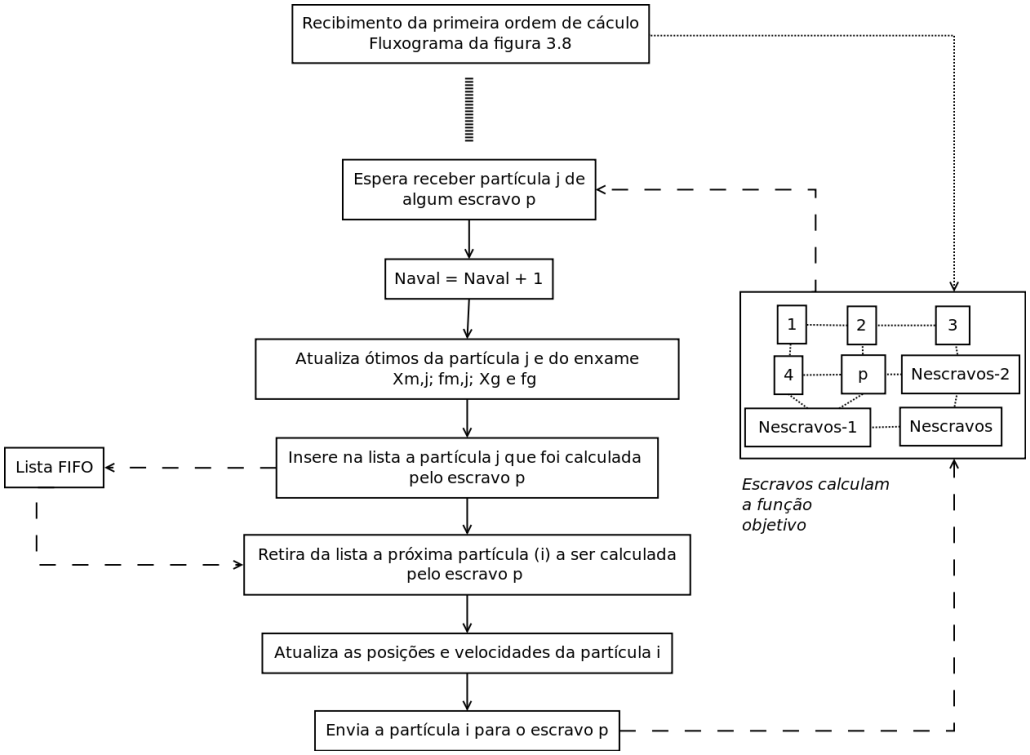

Figura 3.9: *Detalhamento da etapa de otimização do AIU–PPSO, referente ao bloco correspondente na figura 3.8.*

Pseudo–código 12 *Pseudo código da etapa otimização do algoritmo paralelo síncrono AIU–PPSO*

Inicia e preenche a fila *FIFO*
$N_{aval} \leftarrow N_{pass}$
$otimo \leftarrow f_g$
$k \leftarrow 1$
$Revoada \leftarrow 1$
Envia primeira ordem de cálculo aos escravos;
while ($k < N_{max}$ e $N_{aval} < Max\_aval$) **do**
  Atualiza w com equação 3.3
  Recebe F_{obj} referente à partícula j calculada pelo escravo p
  $N_{aval} \leftarrow N_{aval} + 1$
  Insere na fila a partícula j
  if $(F_{obj} < f_{m,j})$ **then**
    $f_{m,j} \leftarrow F_{obj}$
    $\mathbf{x}_{m,j} \leftarrow \mathbf{x}_j$
  end if
  if $(F_{obj} < f_g)$ **then**
    $f_g \leftarrow F_{obj}$
    $\mathbf{x}_g \leftarrow \mathbf{x}_j$
  end if
  Retira da fila a próxima partícula i a ser calculada
  Atualiza ($\mathbf{v}_i$) e ($\mathbf{x}_i$) da partícula i com equações 3.1 e 3.2
  Envia $\mathbf{x}_i$ para o escravo p
  if (Fim da pseudo–revoada) **then**
    Testa a convergência (ver Pseudo Código 4)
  end if
end while

Capítulo 4

Metodologia de avaliação dos algoritmos

Neste capítulo são apresentadas as definições necessárias para a avaliação dos algoritmos e as funções de teste padrão utilizadas para este fim. Na Seção 4.1 são definidas as métricas utilizadas para determinar a disposição ou deslocamento das partículas no espaço, as quais são utilizadas na definição dos critérios de convergência do enxame. Na Seção 4.2 é definido a caracterização de sucesso do algoritmo em um determinado experimento, que é utilizada para definir a *robustez* e a *perfomance* dos algoritmos. Na Seção 4.3 é definido o *speedup* e a *eficiência de paralelização*, conceitos utilizados para a avaliação de desempenho dos algoritmos paralelos. Por fim, nas Seções 4.4 e 4.5 são apresentados os recursos computacionais e as funções objetivo utilizadas nos diversos testes dos algoritmos desenvolvidos.

4.1 Definição das métricas. Critérios de convergência do enxame

A norma euclidiana de um vetor no espaço $\Re^n$ é definida de acordo com a equação 4.1.

$$\|\mathbf{x}\|_2 = \Big[\sum_{i=1}^{n} x_i^2\Big]^{\frac{1}{2}} \tag{4.1}$$

Nos algoritmos implementados, a movimentação das partículas foi feita no espaço previamente normalizado ao intervalo [0, 1] em todas as variáveis independentes da função objetivo. Em outras palavras, as partículas se movimentam em um *hipercubo* de lado igual a 1. Com essa normalização, a distância máxima entre dois pontos neste espaço corresponde a *diagonal do hipercubo* que é igual à $\sqrt{n}$.

Dado essa normalização do espaço, optou-se pela redefinição da norma utilizada para o cálculo de distância entre os vetores, calculando-a relativamente a essa maior distância no espaço, de acordo com a equação 4.2.

$$\|\mathbf{x}\| = \Big[\frac{1}{n}\sum_{i=1}^{n} x_i^2\Big]^{\frac{1}{2}} = \frac{\|\mathbf{x}\|_2}{\sqrt{n}} \tag{4.2}$$

A norma pré-definida pela equação 4.2 pode ser utilizada para calcular a distância entre dois vetores relativamente à maior distância do espaço normalizado.

Critérios de convergência

A norma dada pela equação 4.2 foi utilizada para definir critérios de convergência baseados na disposição ou no deslocamento das partículas no espaço de busca.

Quatro critérios de convergência foram propostos e denominados, respectivamente, de *Critérios 1, 2, 3 e 4*. Em todos estes critérios, o *número de permanência* (N_{min}), o qual especifica o número mínimo de vezes que o melhor valor da função objetivo calculado pelo enxame deve se manter entre as (*pseudo-*)*revoadas*, deve ser satisfeito para que os critérios de convergência sejam avaliados. No caso do *Critério 1*, este é o próprio critério de convergência do método. A permanência do valor da função objetivo é medida pela condição dada na equação 4.3 sobre o melhor valor da função objetivo entre duas (*pseudo-*)*revoadas* do método.

$$|f_g^k - f_g^{k-1}| < \epsilon_a + \epsilon_r |f_g^k| \tag{4.3}$$

onde k representa a *revoada* ou *pseudo-revoada* (no caso do AIU–PPSO) e ϵ_a e ϵ_r são as tolerâncias absoluta e relativa pré-estabelecidas pelo usuário. A utilização da tolerância relativa pode ser útil quando os valores de função objetivo são elevados,

tornando o critério absoluto demasiadamente rigoroso, ao passo que a utilização da tolerância absoluta se torna mais adequada quando o valor ótimo da função objetivo é próximo de zero, tornando o critério relativo muito rigoroso.

O *Critério 1* é, portanto, satisfeito quando o critério estabelecido pela equação 4.3 for satisfeito por N_{min} vezes consecutivas. No caso dos *Critérios 2, 3 e 4*, uma vez satisfeito o *Critério 1*, são avaliados, respectivamente, as seguintes distâncias:

Critério 2 Distância da posição média ponderada do enxame para a melhor posição do enxame;

Critério 3 Deslocamento da posição média do enxame;

Critério 4 Raio da menor *hiperesfera*, centrada na coordenada de melhor posição do enxame, que contém todas as partículas.

A posição média ponderada do enxame é definida pela equação 4.4.

$$\overline{\mathbf{y}} = \sum_{i=1;i\neq g}^{N_{pass}} \overline{\omega}_i \mathbf{y}_i \tag{4.4}$$

em que ω_i é o peso definido pela equação 4.5 e $\mathbf{y}_i = [\mathbf{x}_i|\tilde{f}_{m,i}]$ é o vetor de posição das partícula i no espaço $\Re^{n+1}$, que inclui como coordenada não apenas as variáveis de otimização mas também a função objetivo normalizada, concatenada ao final do vetor $\mathbf{x}_i$.

$$\overline{\omega}_i = \frac{\frac{1}{\|\mathbf{y}_i - \mathbf{y}_g\|}}{\sum_{i=1;i\neq g}^{N_{pass}} \frac{1}{\|\mathbf{y}_i - \mathbf{y}_g\|}} \tag{4.5}$$

onde $\mathbf{y}_g = [\mathbf{x}_g|\tilde{f}_g]$ são as coordenadas de melhor posição do enxame.

A normalização do valor da função objetivo é realizado pela medida do maior e menor valores da função objetivo, calculados, uma única vez, para a população inicial do enxame, dado pela equação 4.6.

$$\tilde{f}_{m,i} = \frac{f_{m,i} - \min_i f^0_{m,i}}{\max_i f^0_{m,i} - \min_i f^0_{m,i}} \tag{4.6}$$

O *Critério 2*, dado pela equação 4.7, avalia, portanto, a distância entre a posição média ponderada do enxame e a melhor posição registrada pelo enxame até o instante em questão.

$$\|\overline{\mathbf{y}} - \mathbf{y}_g\| < \epsilon_a \tag{4.7}$$

No *Critério 3*, dado pela equação 4.8, avalia-se o deslocamento da posição média entre duas avaliações sucessivas do critério, o que só ocorre no mínimo a cada N_{min} (*pseudo-*)*revoadas* do algoritmo.

$$\|\overline{\mathbf{y}}^k - \overline{\mathbf{y}}^{k-N_{min}}\| < \epsilon_a \tag{4.8}$$

No *Critério 4*, dado pela equação 4.9, é avaliada a maior distância entre as partículas para o ponto de melhor posição do enxame, o que define o raio da menor *hiperesfera* no espaço $\Re^{n+1}$ centrada na melhor posição do enxame e que contém todas as partículas do mesmo.

$$\max_i \|\mathbf{y}_i - \mathbf{y}_g\| < \epsilon_a \tag{4.9}$$

4.2 Caracterização de Sucesso. Robustez e Perfomance dos algoritmos

Considera-se que uma dada busca do enxame nos algoritmos PSO resultou em *sucesso* quando a distância entre a solução obtida pelo enxame, após a convergência ou término do procedimento de busca, e a solução conhecida do problema satisfaz a tolerância especificada, ou seja, quando a equação 4.10 for satisfeita.

$$\|\mathbf{y}_{ot} - \mathbf{y}^*_{ot}\| < \epsilon_a^{ot} \tag{4.10}$$

em que $\mathbf{y}_{ot} = [\mathbf{x}_{ot}|\tilde{f}_{ot}]$ é a solução do problema obtida pelo PSO e $\mathbf{y}^*_{ot} = [\mathbf{x}^*_{ot}|\tilde{f}^*_{ot}]$ é o ótimo global conhecido do problema.

Vale ressaltar que a tolerância especificada para a caracterização de sucesso na busca não é, necessariamente, a mesma tolerância especificada para a convergência do enxame durante o procedimento de busca, embora, para a maior parte dos testes realizados, tenha-se adotado os mesmos valores. A caracterização de sucesso definida acima apenas faz sentido em problemas para os quais o ponto de ótimo global é de fato conhecido no intervalo de busca utilizado, caso das funções testes utilizadas para avaliar os algoritmos.

A partir da definição de *sucesso* pode-se definir a *robustez* dos algoritmos na solução de um dado problema teste. A robustez é definida como a razão entre o número de problemas resolvidos com *sucesso* e o número total de experimentos realizados. Cada experimento corresponde a uma simulação independente do algoritmo, utilizando um população inicial diferente. Mais especificamente, para cada experimento, uma semente diferente de números aleatórios é utilizada para iniciar o enxame. A equação 4.11 define a robustez (χ).

$$\chi = \frac{N_{sucesso}}{N_{experimentos}} \tag{4.11}$$

Com a definição de *sucesso* pode-se qualificar ainda a *perfomance* dos algoritmos para resolver os problemas. No presente trabalho, esta *perfomance* é dada diretamente pelo número médio de avaliações da função objetivo, considerando-se também o seu desvio padrão, necessários para resolver o problema. Assim, o método tem melhor *perfomance* quanto menor for o número de avaliações da função objetivo necessários para resolver o problema, número este medido apenas nas buscas que resultaram em sucesso.

4.3 Speedup e Eficiência de Paralelização

Para medir o desempenho dos algoritmos paralelos, utilizaram-se os conceitos de *speedup*[1] (S) e a *eficiência de paralelização* (η). O *speedup* (*speedup justo*) é definido como a razão entre o tempo de computação gasto pelo algoritmo serial e o tempo de computação gasto pela algoritmo paralelo, dado pela equação 4.12.

$$S = \frac{T_{serial}}{T_{paralelo}} \tag{4.12}$$

sendo T_{serial} o tempo computacional gasto pelo algoritmo serial mais eficiente para resolver o problema (que como será mostrado na Seção 5.1.2, corresponde ao IU–SPSO) e $T_{paralelo}$ o tempo computacional gasto pelo algoritmo paralelo sobre N_{proc} processadores. Para medida do T_{serial} os experimentos foram executados isoladamente em único processo dos nós computacionais, de modo a se utilizar da maneira mais eficiente os recursos computacionais disponíveis.

O *speedup* (*speedup justo*) mostra o ganho efetivo em termos de tempo de processamento que o algoritmo paralelo fornece sobre o algoritmo serial. Caso o algoritmo paralelo sobre N_{proc} processadores resultasse em N_{proc} vezes mais rápido do que o algorimo serial, teríamos um speedup ideal igual a N_{proc}. A *eficiência de paralelização* (η) é obtida, portanto, como a razão entre o *speedup efetivo*, calculado pela 4.12, e o *speedup ideal*, conforme mostra a equação 4.13.

$$\eta = \frac{S}{N_{proc}} \tag{4.13}$$

Como os tempos medidos para para cada algoritmo são resultados de experimentos independentes, a relação fornecida pela equação 4.13 é calculada com os valores médios dos tempos de processamento seriais e paralelos. Assim, uma relação para o cálculo de desvio padrão de medidas indiretas (no caso o *speedup*) é necessária. A equação 4.14, cuja demonstração encontra-se no Apêndice A, fornece o desvio padrão para o *speedup*.

[1]Mais precisamente, o *speedup* é definido como a razão entre o tempo gasto pelo código paralelo em 1 e em N_{proc} processadores. A definição utilizada neste trabalho, dada pela equação 4.12, corresponde à de *speedup justo* (SECCHI [39]).

$$\sigma_S^2 = \frac{[\sigma_{T_s}^2 + E(T_s)^2][\sigma_{T_p}^2 + E(T_p)^2]}{E(T_p)^4} - \Big[\frac{E(T_s)}{E(T_p)}\Big]^2 + \frac{\sigma_{T_p}^2}{E(T_p)^4}\Big[2\sigma_{T_s}^2 - \sigma_{T_p}^2 \frac{E(T_s)^2}{E(T_p)^2}\Big] \quad (4.14)$$

onde E e σ são respectivamente a média e o desvio padrão do tempo serial (T_s), do tempo paralelo (T_p) e do speedup.

4.4 Recursos Computacionais

Para a análise dos algoritmos paralelos foram utilizados os computadores paralelos do cluster ARES[2] do Laboratório de Termofluidodinâmica do Programa de Engenharia Química da COPPE (LTFD/PEQ/COPPE/UFRJ). Este cluster é formado por 16 computadores (nós de computação) com dois processadores Quad-Core AMD Opteron Processor 2356 (totalizando 128 processadores) de $2.3GHz$ e $16GB$ de memória ligados em rede Infiniband DDR de $20Gbit/s$.

4.5 Funções de teste

Diversas funções de teste padrão, descritas a seguir, foram utilizadas para validação e análises de desempenho dos algoritmos.

Função Ackley

A função Ackley é uma função teste multidimensional dada pela equação 4.15. Em duas variáveis independentes esta função apresenta uma extensa região praticamente plana contendo diversos mínimos e máximos locais de pequena amplitude. Próximo à origem, entretanto, há um grande *abismo*, sendo o mínimo global localizado na origem onde a função tem valor nulo. A figura 4.1 mostra a superfície e as curvas de níveis desta função em duas variáveis no intervalo considerado.

[2]Este era o nome do cluster na época de realização das análises. Atualmente o cluster tem o nome MARTE e conta com mais 4 nós com 2 processadores AMD Opteron de 2.1 GHz e $64GB$ de memória

$$F(\mathbf{x}) = -20\exp\Big(-0,2\sqrt{\sum_{i=1}^{n}\frac{x_i^2}{n}}\Big) - \exp\Big(\sum_{i=1}^{n}\frac{\cos 2\pi x_i}{n}\Big) + 20 + \exp(1)$$
$$\mathbf{x} \in [-32,68, 32,68]^n \tag{4.15}$$

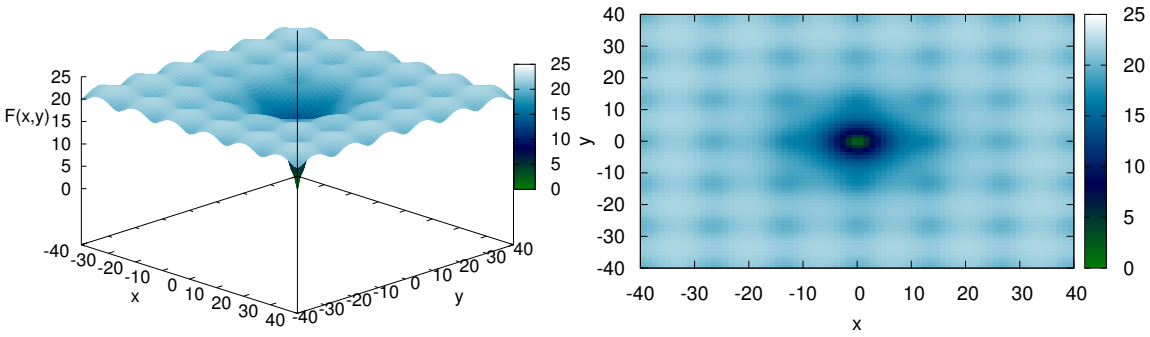

Figura 4.1: *Função Ackley.À esquerda a superfície de nível da função objetivo e à direita suas curvas de níveis*

Função Alpina

A função Alpina é uma função teste bidimensional dada pela equação 4.16. No intervalo considerado apresenta cerca de quatro pontos de mínimo de 5 pontos de máximo, tendo o seu máximo global em (7,917, 7,917) onde o valor da função é 7,8856. A figura 4.2 mostra a superfície e as curvas de níveis desta função no intervalo considerado.

$$F(\mathbf{x}) = \sqrt{x_1 x_2}\sin(x_1)\sin(x_2); \quad \mathbf{x} \in [0,10]^2 \tag{4.16}$$

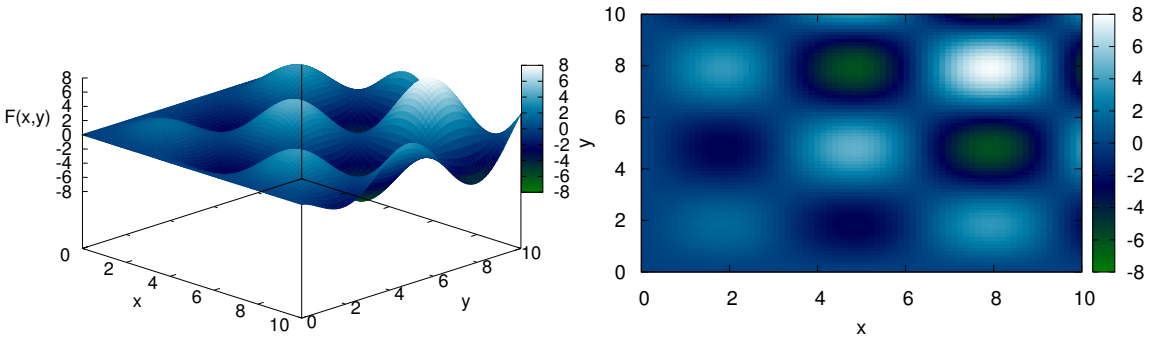

Figura 4.2: *Função Alpina.À esquerda a superfície de nível da função objetivo e à direita suas curvas de níveis*

Função de Griewank

A função multidimensional de Griewank, dada pela equação 4.17, apresenta demasiados mínimos e máximos locais distribuídos de maneira quase uniforme ao longo do intervalo de busca. Seus mínimos locais e o mínimo global apresentam valores da função objetivo muito próximos, o que torna essa função extremamente difícil de se otimizar. A figura 4.3 mostra a sua superfície e suas curvas de níveis no intervalo [-10, 10] em cada variável.

$$F(\mathbf{x}) = \sum_{i=1}^{n} \frac{1}{4000} x_i^2 - \prod_{i=1}^{n} cos\Big[\frac{x_i}{\sqrt{i+1}}\Big]; \quad \mathbf{x} \in [-600, 600]^n \tag{4.17}$$

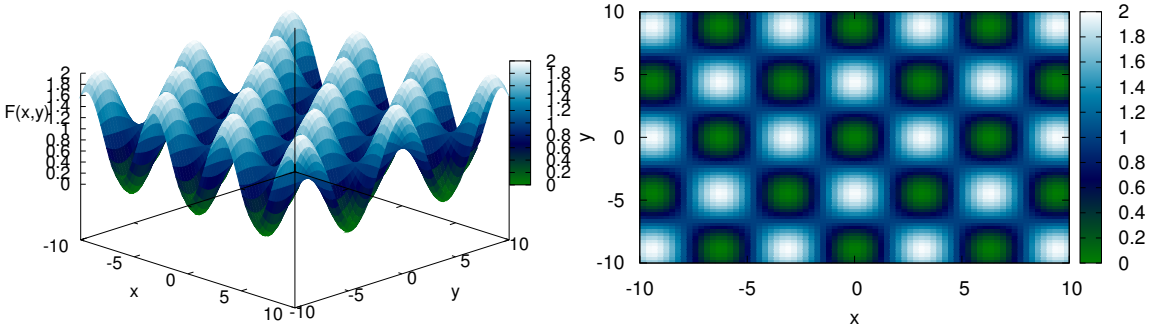

Figura 4.3: *Função Griewank.À esquerda a superfície de nível da função objetivo e à direita suas curvas de níveis*

Função Levy

A função Levy é uma função teste bidimensional que apresenta muitos pontos de mínimo e de máximo locais, tendo o seu mínimo global em (-1,3068, -1,4248) onde o valor da função é -176,1376. A equação 4.18 descreve a função. A figura 4.4 mostra sua superfície e suas curvas de níveis no intervalo .

$$\begin{aligned} F(\mathbf{x}) = &\Big[\textstyle\sum_{i=1}^{5} i \cdot cos[(i-1)x_1 + 1]\Big]\Big[\textstyle\sum_{i=1}^{5} i \cdot cos[(i+1)x_2 + 1]\Big] + \\ &(x_1 + 1,42513)^2 + (x_2 + 0,80032)^2; \quad \mathbf{x} \in [-10, 10]^2 \end{aligned} \tag{4.18}$$

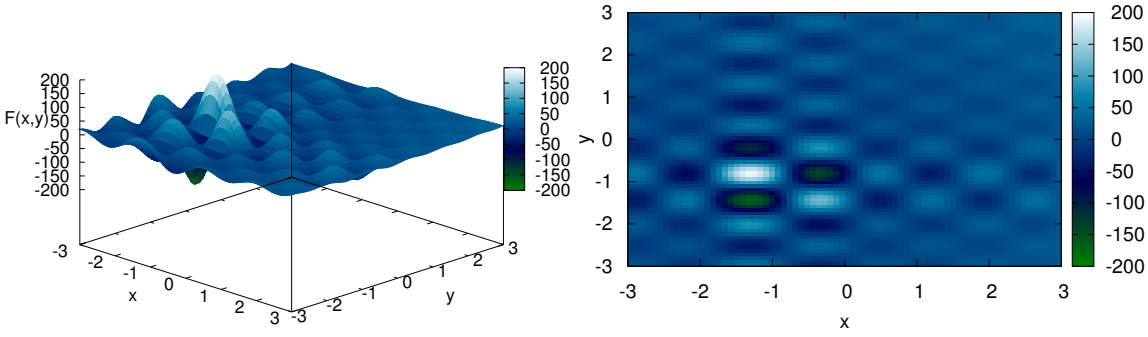

Figura 4.4: *Função Levy.À esquerda a superfície de nível da função objetivo e à direita suas curvas de níveis*

Função Rastrigin

A função Rastrigin é uma função teste multidimensional dada pela equação 4.19. Em duas dimensões apresenta-se com a forma de um parabolóide com inúmeros pontos de máximo e mínimos locais de menor amplitude ao longo desta superfície parabólica. O mínimo global é localizado na origem, onde a função também apresenta valor nulo. A figura 4.5 mostra as superície e as curvas de níveis da função objetivo.

$$F(\mathbf{x}) = 10n + \sum_{i=1}^{n} x_i^2 - 10cos[2\pi x_i]; \quad \mathbf{x} \in [-5, 12, 5, 12]^n \tag{4.19}$$

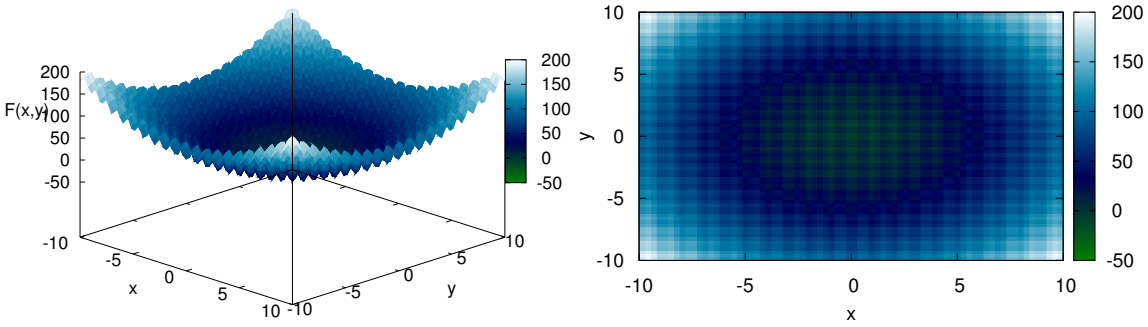

Figura 4.5: *Função Rastrigin.À esquerda a superfície de nível da função objetivo e à direita suas curvas de níveis*

Função Rosenbrook

A função Rosenbrook multidimensional, dada pela equação 4.20, foi outra função teste utilizada. Esta função apresenta um único ponto de mínimo localizado na posição $[1, 1]^n$, onde a função também tem valor nulo. No espaço bidimensional

apresenta-se na forma de um grande *vale*, região onde a variação no valor da função é pequena. A figura 4.6 mostra a superfície e as curvas de níveis desta função.

$$F(\mathbf{x}) = \sum_{i=1}^{n} 100[x_{i+1} - x_i^2]^2 + (x_i - 1)^2; \quad \mathbf{x} \in [-100, 100]^n \tag{4.20}$$

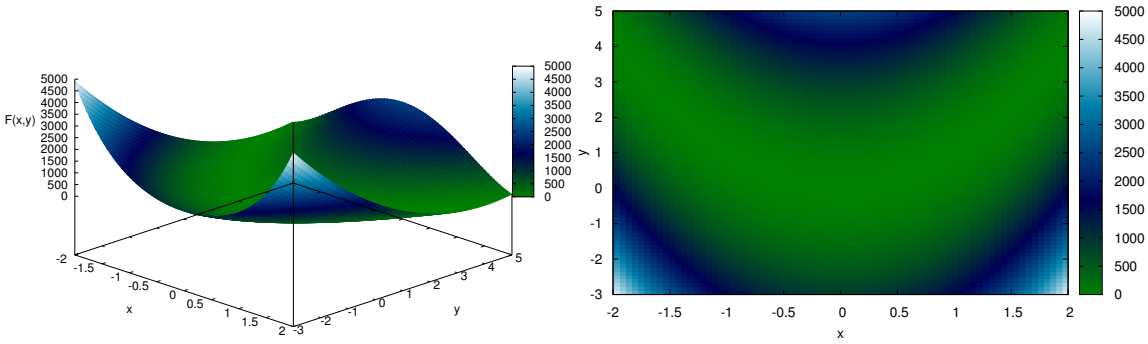

Figura 4.6: *Função Rosenbrook.À esquerda a superfície de nível da função objetivo e à direita suas curvas de níveis*

Função Schwefel

A função multidimensional de Schwefel dada pela equação 4.21, apresenta diversos mínimos locais distribuídos no espaço de busca, distantes um do outro e com valor de função objetivo muito próximo ao do ótimo global, o qual se encontra na posição $[420,9687, 420,9687]^n$. A figura 4.7 apresenta sua superfície e suas curvas de níveis.

$$F(\mathbf{x}) = 418n - \sum_{i=1}^{n} x_i sen[\sqrt{(|x_i|)}]; \quad \mathbf{x} \in [-500, 500]^n \tag{4.21}$$

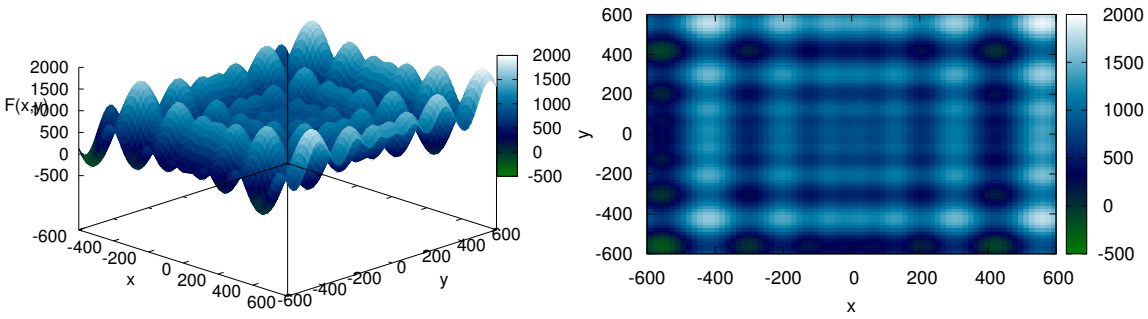

Figura 4.7: *Função Schwefel.À esquerda a superfície de nível da função objetivo e à direita suas curvas de níveis*

Função H1

A função bidimensional H1 é dada pela equação 4.22 (KOH *et al.* [4]). Ela apresenta diversos mínimos e máximos locais de pequena amplitude distribuídos ao longo de uma extensa região. Próximo à origem há uma elevação abrupta do valor da função objetivo, sendo que o máximo global está localizado na posição [8,6998, 6,7665].

$$F(\mathbf{x}) = \frac{sen^2\left[x_1 - \frac{x_2}{8}\right] + sen^2\left[x_2 + \frac{x_1}{8}\right]}{1 + \sqrt{(x_1 - 8,6998)^2 + (x_2 - 6,7665)^2}}; \quad \mathbf{x} \in [-100, 100]^n \tag{4.22}$$

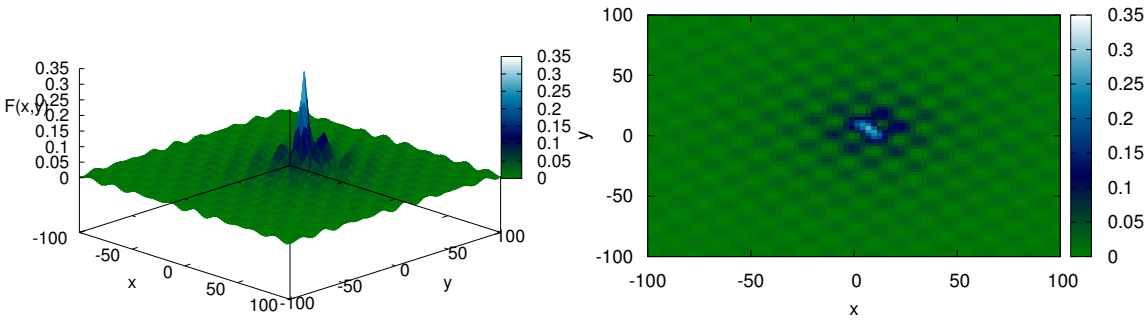

Figura 4.8: *Função Schwefel.À esquerda a superfície de nível da função objetivo e à direita suas curvas de níveis*

Capítulo 5

Resultados e Discussões

Neste capítulo são apresentados os resultados obtidos neste trabalho. Na Seção 5.1 são apresentados os resultados da análise de algumas das características dos métodos PSO seriais implementados. Na Seção 5.2 são apresentados os resultados da análise de desempenho das versões paralelas implementadas, bem como os resultados da aplicação do AIU–PPSO na otimização de um problema de balanço populacional desenvolvido em ARAÚJO [1], sendo este último comparado aos obtidos pelo otimizador ODRPACK95 (ZWOKAK *et al.* [5]) usado nesta mesma referência.

5.1 Características dos métodos PSO

Nesta seção são apresentados os resultados obtidos com o objetivo de avaliar as características dos algoritmos de enxame de partículas implementados em suas versões seriais. Na Seção 5.1.1 é feita a comparação entre os critérios de convergência desenvolvidos, conforme definidos na Secção 4.1. Esta comparação é realizada em termos da *robustez* e *perfomance* dos algoritmos na solução de um determinado número de experimentos com as funções de testes. Na Seção 5.1.2 apresenta-se a comparação, também em termos de *robustez* e *perfomance*, entre as duas formas de atualização das posições e velocidade das partículas: *atualizações imediatas (Immediate Update)* e *atualizações por revoada (Swarm Update)*. Por fim, na Seção 5.1.3 são apresentados os resultados referentes à análise de escalabilidade do número de partículas do enxame com a dimensão do espaço de busca.

5.1.1 Avaliação dos algoritmos

Para todos os testes que serão apresentados foram utilizados os valores de parâmetros dados na Tabela 5.1. Com esse conjunto de parâmetros, as partículas no PSO apresentam comportamento convergente para peso de inércia (w) maior do que 0,5, conforme análise descrita no Apêndice C.

Tabela 5.1: *Parâmetros utilizados nos experimentos numéricos com o IU–SPSO*

	c_1	c_2	w_0	w_f
Todas as Funções	1,5	1,5	1,0	0,1

	N_{pass}
Ackley	20
Alpina	20
Rosenbrook	120
Levy	50
Rastrigin	50
Griewank	400
Schwefel	100

Da Tabela 5.2 à 5.5 são apresentados os resultados obtidos para a *robustez* e *perfomance* do IU–SPSO obtidos em 1000 experimentos numéricos independentes para cada um dos critérios de convergência definidos na Seção 4.1. As funções objetivos testadas foram aquelas definidas na Seção 4.5. Ressalta-se que a *perfomance* do algoritmo é medida em termos do número de avaliações da função objetivo, sendo avaliada apenas para os experimentos que resultaram em *sucesso*, conforme definido na Seção 4.2.

As figuras 5.1 à 5.7 mostram os histogramas de *robustez* (χ) e os gráficos de número de avaliações da função objetivo para os respectivos com a variação do número de permanência no ótimo (N_{min}).

Tabela 5.2: *Resultados da otimização obtidos com o critério 1*

	Robustez [%]				*Nº médio de Avaliações/Desvio Padrão*			
	N_{min}				N_{min}			
	10	20	40	80	10	20	40	80
Ackley	28,3	64,9	92,4	99,1	2343/196	3330/180	4911/227	7634/331
Alpina	18,8	65,4	91,4	97,9	1288/115	1817/148	2790/241	4544/397
Rosenbrook	6,5	48,7	85,1	97,1	7872/881	11160/1080	16523/1373	25528/1606
Levy	10,7	55,4	88	98,6	3338/373	4892/418	7331/519	11542/665
Rastrigin	15,6	62,1	89,1	98,3	3118/293	4422/401	6680/487	10669/635
Griewank	9,5	52,4	85,2	96,9	25242/5199	39498/5068	59702/6236	94787/8756
Schwefel	32,5	84,2	97,2	100	5969/709	8597/726	12813/869	20257/1133

Tabela 5.3: *Resultados da otimização obtidos com o critério 2*

	Robustez [%]				*Nº médio de Avaliações/Desvio Padrão*			
	N_{min}				N_{min}			
	10	20	40	80	10	20	40	80
Ackley	94,7	99,4	100	100	2322/180	3328/186	4910/229	7642/324
Alpina	95,2	99,2	99,8	99,9	1389/122	1945/179	2865/251	4571/352
Rosenbrook	89,5	98	100	100	7948/1155	11180/1175	16486/1430	25533/1604
Levy	98,7	99,9	99,9	100	3458/355	4946/412	7337/486	11524/644
Rastrigin	99,4	99,9	100	100	3124/269	4440/346	6693/496	10634/623
Griewank	74,9	84,1	91,8	98	27005/3458	39847/4027	59720/5910	94620/9068
Schwefel	99,2	99,9	100	100	6026/564	8557/665	12818/846	20249/1172

Tabela 5.4: *Resultados da otimização obtidos com o critério 3*

	Robustez [%]				*Nº médio de Avaliações/Desvio Padrão*			
	N_{min}				N_{min}			
	10	20	40	80	10	20	40	80
Ackley	100	100	100	100	3297/132	5160/173	8538/223	14868/315
Alpina	100	100	100	100	1997/128	3327/179	5805/259	10576/372
Rosenbrook	100	100	100	100	11277/1429	17350/1144	28555/1359	49677/1722
Levy	99,8	100	100	100	5051/381	8004/419	13423/489	23613/647
Rastrigin	100	100	100	100	4648/301	7494/365	12712/499	22700/641
Griewank	92,4	98,3	99,5	100	41454/4591	66503/7435	110518/9239	191850/10577
Schwefel	99,5	99,9	100	100	9148/630	14705/752	24976/839	44342/1168

Tabela 5.5: *Resultados da otimização obtidos com o critério 4*

	Robustez [%]				*Nº médio de Avaliações/Desvio Padrão*			
	N_{min}				N_{min}			
	10	20	40	80	10	20	40	80
Ackley	100	100	100	100	2616/176	3642/277	5234/466	7757/520
Alpina	100	100	100	100	1992/122	2973/231	4305/444	6570/411
Rosenbrook	100	100	100	100	11216/1447	17250/1124	28574/1320	49411/2276
Levy	99,8	99,9	100	100	5029/366	7923/468	12708/1222	19646/3288
Rastrigin	100	100	100	100	4651/301	7439/381	11619/1253	16794/2521
Griewank	92,7	97,8	99,4	100	41661/4760	66359/7047	110113/8916	192506/11214
Schwefel	99,7	99,8	100	100	9158/645	14707/694	23146/2266	3143/4974

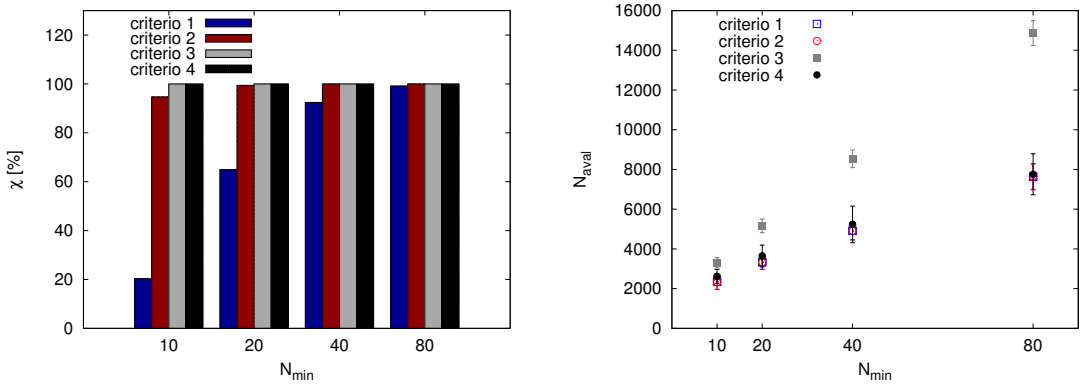

Figura 5.1: *Efeito do número de permanência no ótimo sobre a robustez e eficiência do algoritmo na otimização da função Ackley*

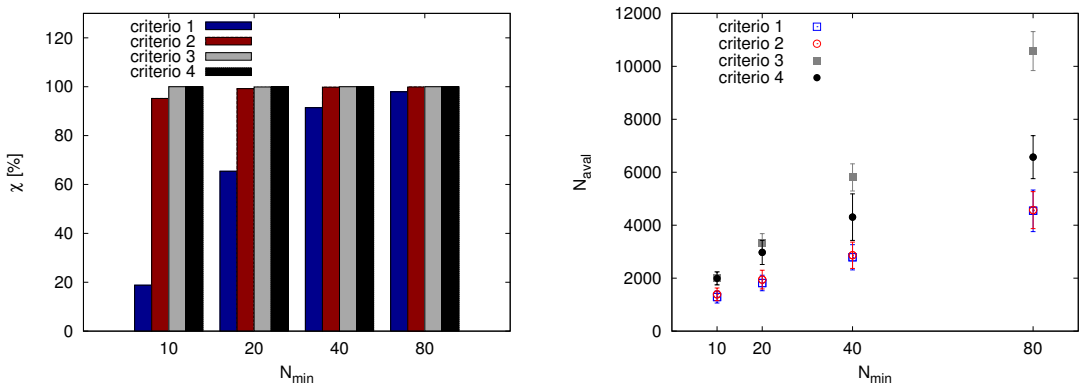

Figura 5.2: *Efeito do número de permanência no ótimosobre a robustez e eficiência do algoritmo na otimização da função Alpina*

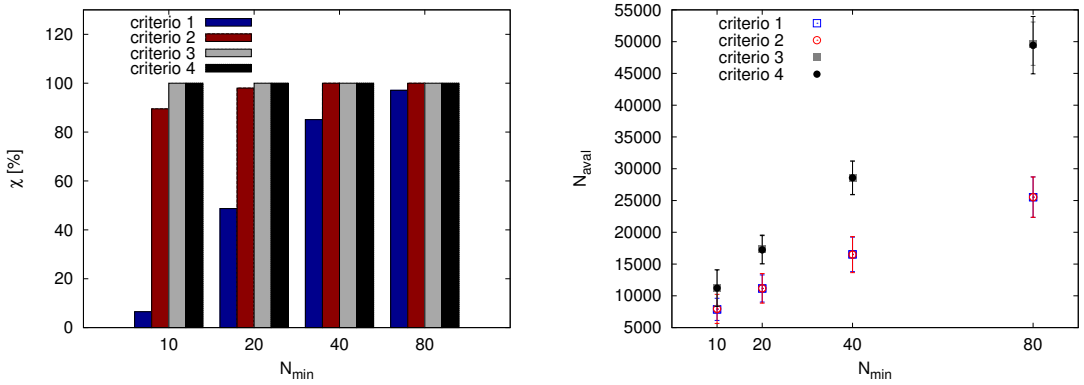

Figura 5.3: *Efeito do número de permanência no ótimo sobre a robustez e eficiência do algoritmo na otimização da função Rosenbrook*

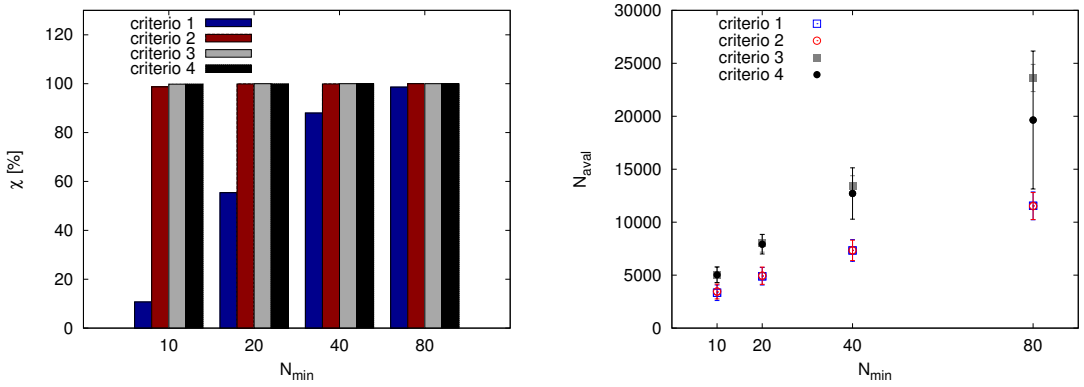

Figura 5.4: *Efeito do número de permanência no ótimo sobre a robustez e eficiência do algoritmo na otimização da função Levy*

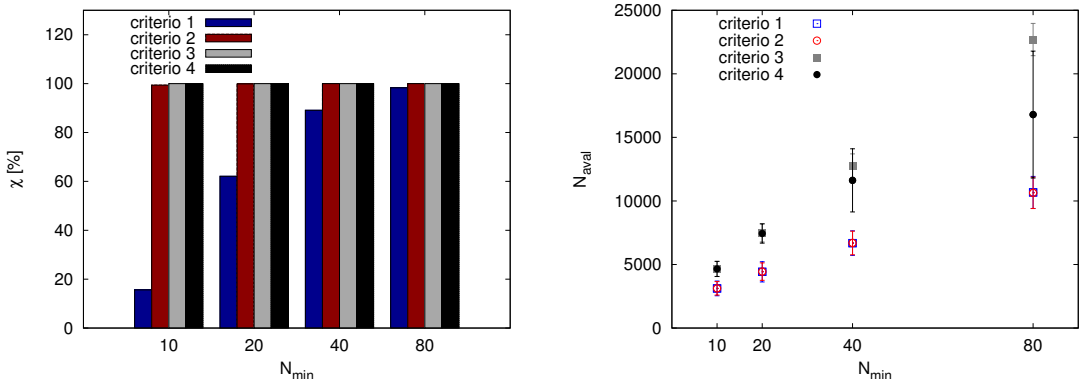

Figura 5.5: *Efeito do número de permanência no ótimo sobre a robustez e eficiência do algoritmo na otimização da função Rastrigin*

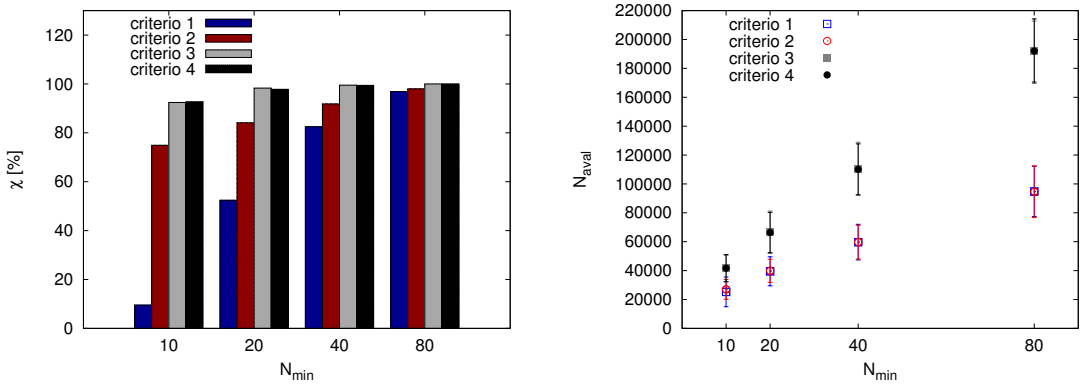

Figura 5.6: *Efeito do número de permanência no ótimo sobre a robustez e eficiência do algoritmo na otimização da função Griewank*

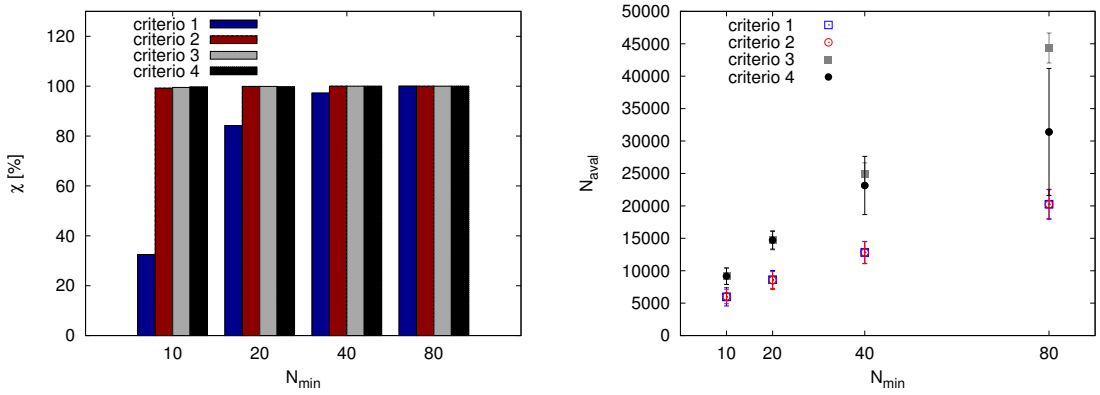

Figura 5.7: *Efeito do número de permanência no ótimo sobre a robustez e eficiência do algoritmo na otimização da função Schwefel*

Para todos os problemas, pelo menos quatro resultados são observados:

1. A robustez do algoritmo aumenta com o número de permanência no ótimo;
2. A utilização de um critério adicional ao número de permanência no ótimo aumenta a robustez do método;
3. O *Critério 2* aumentou significativamente a robustez em relação ao *Critério 1* com baixa perda de *perfomance* em relação ao mesmo;
4. O critério de velocidade do ponto médio do enxame (*Critério 3*) foi o de pior *perfomance* em média.

É evidente da própria definição que o *Critério 1* é o menos robusto e o de melhor *perfomance*, visto que todos os outros o consideram satisfeito para que sejam avaliados. Poderia ser esperado também que o aumento desse número, assim como a utilização de um critério adicional, promoveria uma melhora de robustez às custas de perda de *perfomance*. Contudo, a utilização do *Critério 2*, aumentou significativamente a robustez do algoritmo sem afetar a sua *perfomance* em relação ao *Critério 1*.

O *Critério 3* foi mais robusto e o de pior *perfomance* em média. Observando a definição deste critério, esse resultado parece bastante razoável e intuitivo, pois, se a posição média do enxame não se deslocou entre duas avaliações do critério (o que só acontece no mínimo entre duas *revoadas* do processo de busca), pode-se imaginar que as partículas, com a exceção de situações peculiares, estão com velocidades extremamente baixas. Isso só é possível se todas as partículas estiverem muito próximas umas das outras, pois, do contrário, elas teriam velocidades diferentes, aumentando as chances de deslocamento do ponto médio. Por outro lado, pode-se imaginar a situação em que o enxame ainda não está muito concentrado (o que implica que as partículas ainda se movem) mas, a distância da posição média ponderada do enxame para o ótimo global (*Critério 2*) já é pequena (visto que o peso na média das partículas mais próximas ao ótimo do enxame é maior), eventualmente satisfazendo a tolerância especificada.

O *Critério 4* era esperado ser o mais restritivo dentre todos, por exigir que todas as partículas estejam contidas na bola do hiperespaço de raio menor que a tolerância especificada. Dos resultados pode ser observado que ele foi no mínimo tão robusto quanto o *Critério 3* e em média (e ressaltando-se que apenas em média)

Tabela 5.6: *Qualificação dos critérios de convergência*

Robustez	$critério4 = critério3 > critério2 > critério1$
Perfomance	$critério1 = critério2 > critério4 > critério3$

obteve melhor *perfomance* do que este. Por outro lado, observa-se que este critério foi o que apresentou maior dispersão no número de avaliações da função objetivo, sendo que esta dispersão foi particularmente acentuada nos problemas com as funções de Levy, Rastrigin e Schwefel.

A Tabela 5.6 qualifica os critérios segundo os resultados apresentados em termos de *robustez* e *perfomance*. Desta qualificação, tendo em vista os resultados apresentados, pode se considerar que os *Critérios 2 e 4* são os que apresentam o melhor compromisso entre *robustez* e *perfomance* dentre os critérios propostos para o método.

5.1.2 Comparação entre IU–SPSO e SU–SPSO

Conforme discutido no Capítulo 3, a velocidade e a posição das partículas podem ser atualizadas imediatamente antes do cálculo da função objetivo utilizando a informação atual sobre o estado do enxame (*Immediate Update*) ou ao final de cada revoada (*Swarm Update*). Segundo SCHUTTE *et al.* [3], a primeira opção faz com que o enxame responda mais rapidamente às informações, aumentando assim a *robustez* e a *perfomance* do método. Dessa maneira, a fim de confirmar a característica supracitada, as duas formas de atualização foram comparadas, novamente em termos da *robustez* e a *perfomance* do método.

As Tabelas 5.7 e 5.8 mostram os resultados obtidos para a robustez e para a média e desvio padrão do número de avalições da função objetivo obtidos em 1000 experimentos numéricos independentes com os algoritmos IU–SPSO e SU–SPSO. Para todos os experimentos foi considerado o *Critério 1* que, por ser o menos restritivo, permite que a diferença entre as características dos algoritmos sejam evidenciadas. Os parâmetros do algoritmo considerado são os mesmos dados na Tabela 5.1.

Tabela 5.7: *Resultados da otimização obtidos com o IU–SPSO*

	Robustez [%]				*Nº médio de Avaliações/Desvio Padrão*			
	N_{min}				N_{min}			
	10	20	40	80	10	20	40	80
Ackley	11,1	47,1	73,0	93,1	1556/139,3	2238/148,1	3307/176,0	5156/241,2
Alpina	14,1	54,1	83,4	95,9	1029/127,3	1481/168,9	2258/236,0	3675/340,3
Rosenbrook	7,0	44,3	77,4	91,6	7311/1561,0	10674/2602,0	15995/1510,8	25138/1758,6
Levy	13,2	54,6	84,4	96,4	3770/609,11	5613/640,7	8520/661,1	13524/823,2
Rastrigin	11,5	53,0	83,2	95,3	2869/403,2	4223/457,11	6505/599,4	10456/776,12
Griewank	5,9	37,2	73,8	90,0	22840/6888,6	37461/6884,9	57810/8561,0	92334/9788,0
Schwefel	23,0	69,8	95,3	99,8	5824/872,0	8494/912,8	12714/1091,7	20090/1410,0

Da figura 5.8 à 5.14 mostram-se os gráficos correspondentes aos valores apresentados nas Tabelas 5.7 e 5.8 para as diferentes funções objetivos de teste.

Tabela 5.8: *Resultados da otimização obtidos com o SU–SPSO*

	Robustez [%]				*Nº médio de Avaliações/Desvio Padrão*			
	N_{min}				N_{min}			
	10	20	40	80	10	20	40	80
Ackley	1,4	15,3	50,3	73,8	1785/134,82	2556/192,0	3853/278,8	5959/398,0
Alpina	1,1	16,0	51,3	81,1	1229/148,7	1805/232,8	2721/279,6	4336/445,1
Rosenbrook	0	10,2	40,7	68,9	–	12773/1129,9	18875/1588,2	29346/2285,9
Levy	0,4	13,3	45,8	72,6	4455/274,1	6913/730,5	10476/931,8	16278/1308,8
Rastrigin	1,1	11,3	40,6	72,1	3277/470,8	5180/612,4	8006/800,3	12613/1176,9
Griewank	0	5,6	30,2	57,1	–	48328/5774,5	72410/6978,6	114300/9935,2
Schwefel	3,4	34,1	74,7	93,2	7411/1101,5	10417/986,4	15571/1399,7	24482/1955,8

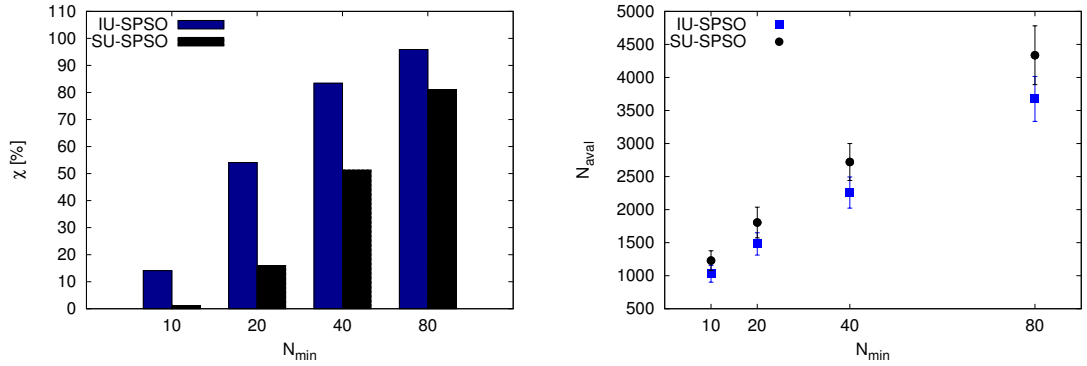

Figura 5.8: *Comparação entre os algoritmos IU-SPSO e SU-SPSO na otimização da função Alpina*

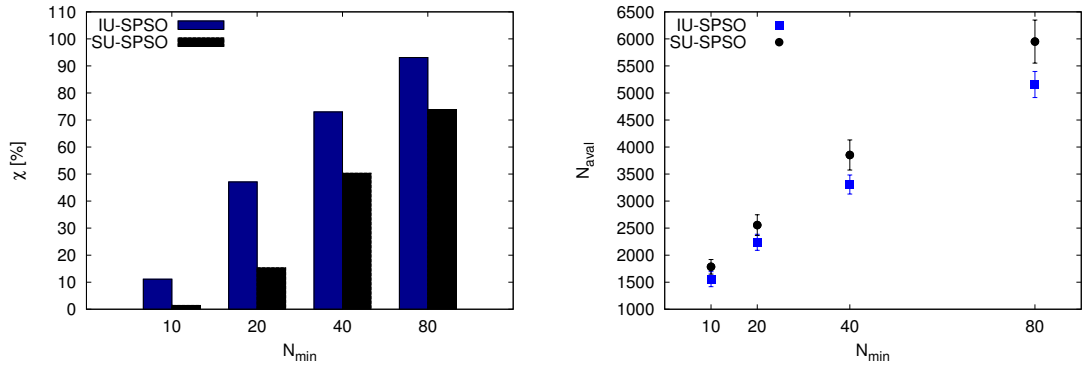

Figura 5.9: *Comparação entre os algoritmos IU-SPSO e SU-SPSO na otimização da função Ackley*

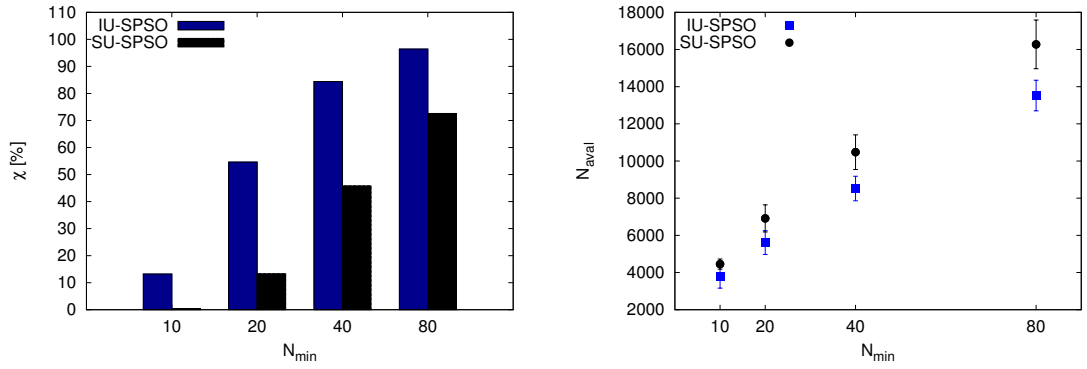

Figura 5.10: *Comparação entre os algoritmos IU-SPSO e SU-SPSO na otimização da função Levy*

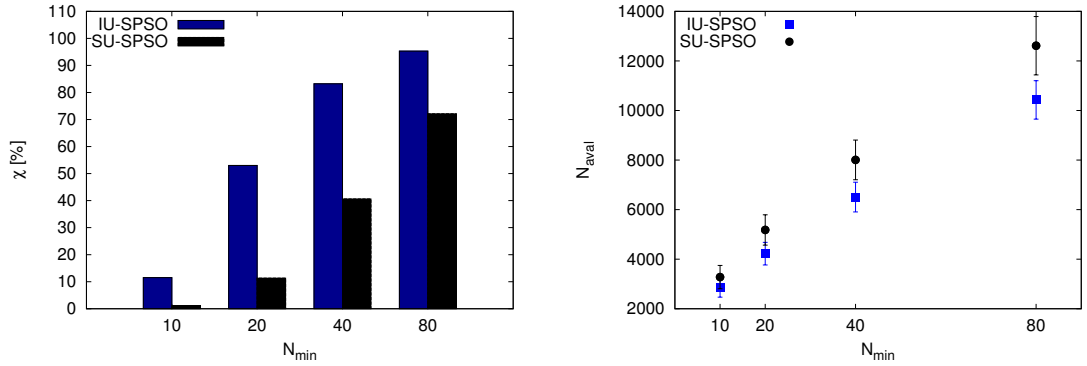

Figura 5.11: *Comparação entre os algoritmos IU-SPSO e SU-SPSO na otimização da função Rastrigin*

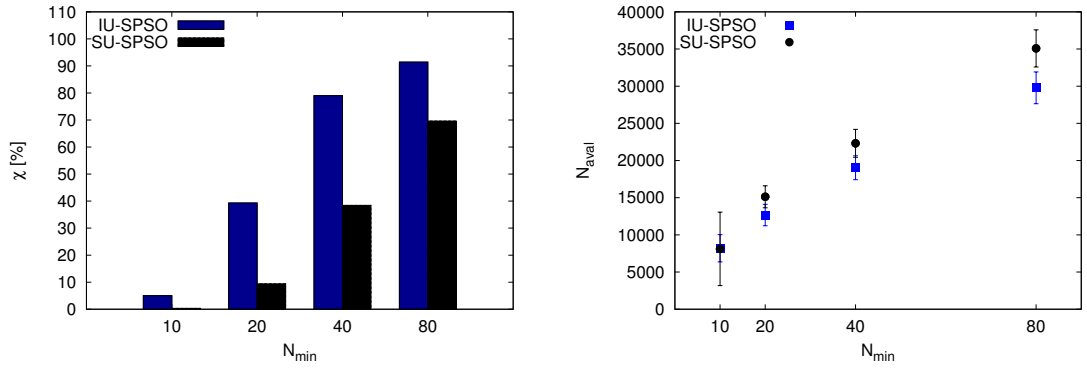

Figura 5.12: *Comparação entre os algoritmos IU-SPSO e SU-SPSO na otimização da função Rosenbrook*

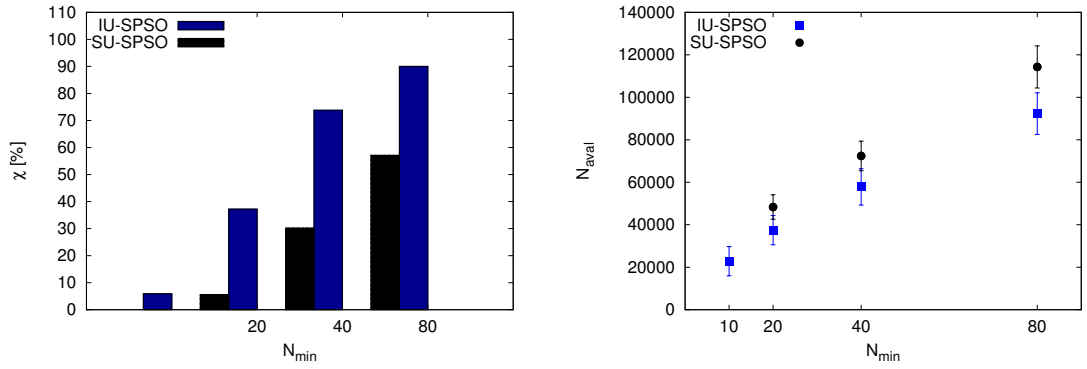

Figura 5.13: *Comparação entre os algoritmos IU-SPSO e SU-SPSO na otimização da função Griewank*

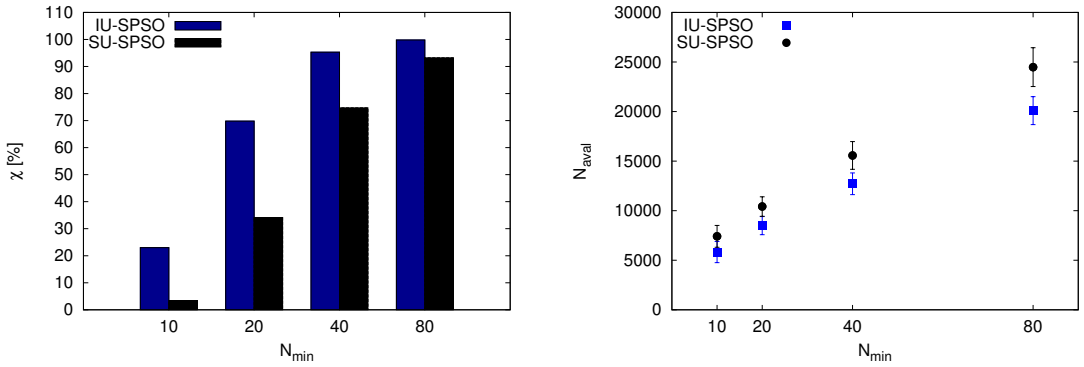

Figura 5.14: *Comparação entre os algoritmos IU-SPSO e SU-SPSO na otimização da função Schwefel*

Para todos os resultados observa-se que a atualização imediata das partículas foi de fato a mais robusta e, em média, a de melhor *perfomance*. Esses resultados confirmam a hipótese de que atualizar a posição e velocidade da partícula com os valores atuais da posição ótima do enxame faz com que o enxame responda mais rapidamente a mudanças nesta posição.

5.1.3 Análise de escalabilidade do número de partículas com a dimensão do espaço

Um ponto de particular interesse na Engenharia é a otimização de problemas com elevada dimensão. Poucos estudos na literatura avaliam o desempenho do PSO ou de outros PBM's em problemas deste tipo, citando-se aqui o trabalho de PICCAND *et al.* [51]. Apesar deste trabalho não apresentar nenhuma relação formal para a escalabilidade do número de partículas do enxame com a dimensão do espaço, seus resultados sugerem uma grande dificuldade do PSO para encontrar o ótimo global em problemas com dimensão elevada, mostrando que para um tamanho fixo do enxame, a taxa de sucesso tende a cair e o número de avaliações da função objetivo tende a crescer rapidamente com o aumento da dimensão. Este comportamento é ainda bastante dependente do tipo da função objetivo em questão.

Um resultado de particular interesse seria encontrar a escalabilidade do número de partículas com a dimensão e a sua relação com o tipo de função objetivo se essas relações existirem ou puderem ser obtidas. Esta seção apresenta, neste sentido, alguns resultados com o objetivo de correlacionar a dimensão do espaço de busca com o número de partículas necessário para que o problema seja resolvido com *sucesso*.

Na Tabela 5.9 são apresentados os resultados obtidos com o IU–SPSO na otimização para cada função objetivo analisada. Para todos os problemas 50 experimentos numéricos independentes foram realizados. O número de partículas considerado para a avaliação da escalabilidade foi aquele para o qual no mínimo 90 % dos experimentos realizados resultaram em sucesso para a dimensão considerada. A busca pelo número de partículas foi realizada pela redução do intervalo $[N^-_{pass}, N^+_{pass}]$, onde N^-_{pass} e N^+_{pass} são os números de partículas para os quais foram obtidos, pela primeira vez, robustez menor e maior do que 90 %. Devido a complexidade dos problemas, particularmente com as dimensões mais elevadas, apenas uma redução de intervalo foi realizada. Assim sendo, resultados mais refinados do que os que serão apresentados podem ser obtidos.

A média de avaliações com seu respectivo desvio padrão foram obtidos apenas para os problemas resolvidos com *sucesso*. Para as análises, o *Critério 3* de convergência foi adotado, pois este era o critério mais robusto disponível[1]. As figuras 5.15 à 5.19 mostram os gráficos em escala logarítmica dos dados apresentados na Tabela 5.9.

As Tabelas 5.10 e 5.11 mostram os coeficientes da correlação linear entre o logaritmo decimal do número de partículas e do número de avaliações da função objetivo com o logaritmo decimal da dimensão do espaço para os dados da Tabela 5.9. Com excessão para função de *Ackley*, o IU–SPSO não obteve a *robustez* desejada para até 10^7 avaliações da função objetivo, número máximo considerado nas análises.

As funções *Griewank* e de *Schwefel* foram as que demonstraram a maior dificuldade de otimização pelo enxame. A forma dessas funções apresentadas na Seção 4.5 sugerem uma explicação para esse resultado observado. Essas duas funções apresentam vários mínimos e máximos locais distribuídos quase uniformente pelo espaço de busca com valores de função objetivo muito próximos em valor ao do ótimo global. Esta característica parece fazer o enxame se prender a mínimos locais, elevando o número de partículas necessários para obtenção de sucesso.

A função de *Ackley* foi a que apresentou o menor coeficiente angular. Contudo, mesmo para essa função, cerca de 4000 partículas e 3 milhões de avaliações foram necessárias com 32 dimensões. Analisando a forma de sua superfície no caso com 2 variáveis de otimização (ver figura 4.1 da Seção 4.5), pode-se especular que o *"abismo"* na região central favoreça a busca pelo enxame quando uma distribuição uniforme de partículas é utilizada inicialmente.

[1] O *Critério 4* foi desenvolvido posteriormente à execução destas análises

Tabela 5.9: *Número de partículas e número médio de avaliações para diferentes dimensões do espaço de busca*

	Dimensão	*Nº de partículas*	*Nº médio de Avaliações*
Ackley	2	5	2585
	4	20	10737
	8	100	56400
	16	500	335675
	32	4000	2976200
Rastrigin	2	15	7067
	4	80	44000
	8	700	447125
	16	40000	25635789
Rosenbrook	2	20	13646
	4	160	169832
	8	800	1120320
	16	40000	44874000
Griewank	2	120	63318
	4	2000	1193100
	8	100000	61485000
	16	500000	76071635
Schwefel	2	25	11405
	4	300	155550
	8	10000	8193000
	16	500000	98826635

Tabela 5.10: *Parâmetros da regressão linear do número de partículas com a dimensão do espaço*

	coeficiente angular	*coef. linear*	*coef. de correlação*
Ackley	1,8470	0,2052	0,9906
Rastrigin	3,8948	-0,1732	0,9782
Rosenbrook	3,6667	0,1021	0,9801
Griewank	4,3653	0,8807	0,9889
Schwefel	4,8381	-0,2128	0,9952

As funções de *Rosenbrook* e *Rastrigin* apresentaram resultados semelhantes. Embora apresentem formas distintas, essas duas funções guardam uma semelhaça entre elas, que é o decréscimo parabólico para o ótimo global, o que pode ter contribuído para a proximidade dos resultados.

Tabela 5.11: *Parâmetros da regressão linear do número do de avaliações da função objetivo com a dimensão do espaço*

	coeficiente angular	*coef. linear*	*coef. de correlação*
Ackley	1,9517	2,8742	0,9917
Rastrigin	4,0222	2,4667	0,9824
Rosenbrook	3,8475	2,9241	0,9908
Griewank	3,9781	3,8998	0,9562
Schwefel	4,5335	2,6558	0,9958

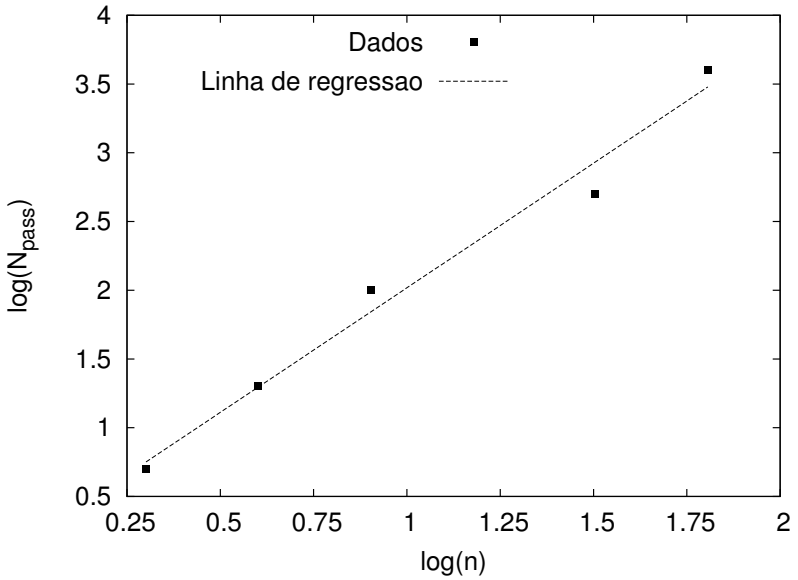

Figura 5.15: *Número de partículas em função da dimensão do espaço para a função Ackley*

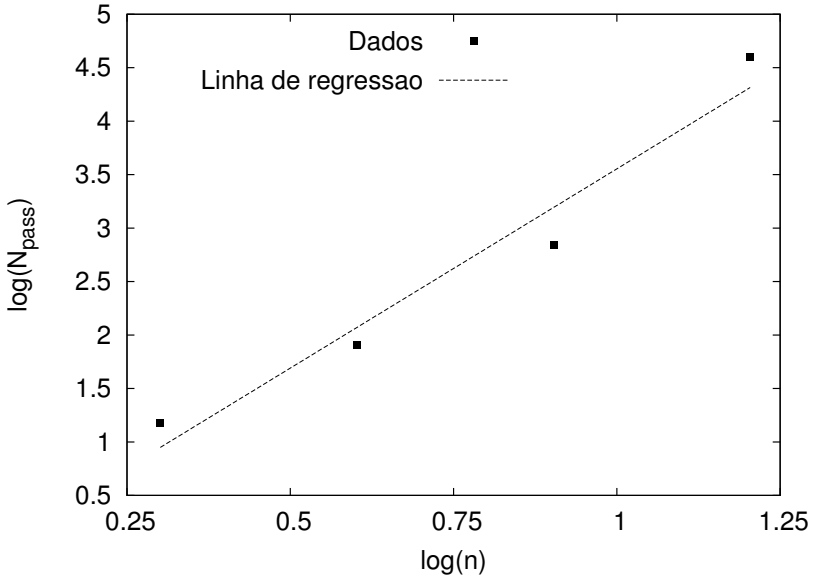

Figura 5.16: *Número de partículas em função da dimensão do espaço para a função Rastrigin*

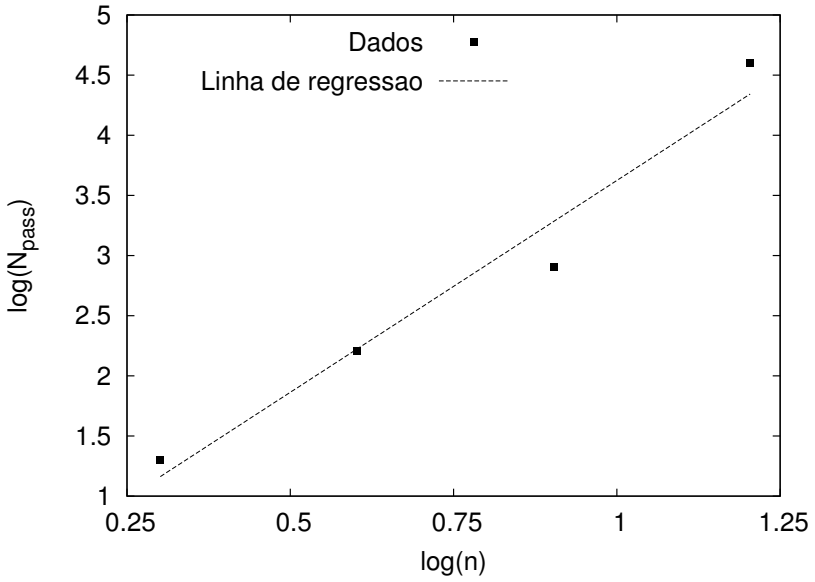

Figura 5.17: *Número de partículas em função da dimensão do espaço para a função Rosenbrook*

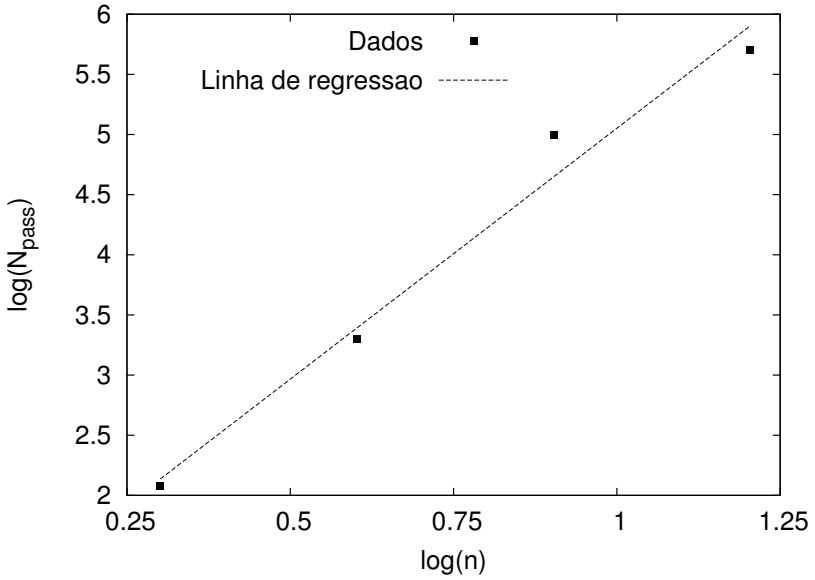

Figura 5.18: *Número de partículas em função da dimensão do espaço para a função Griewank*

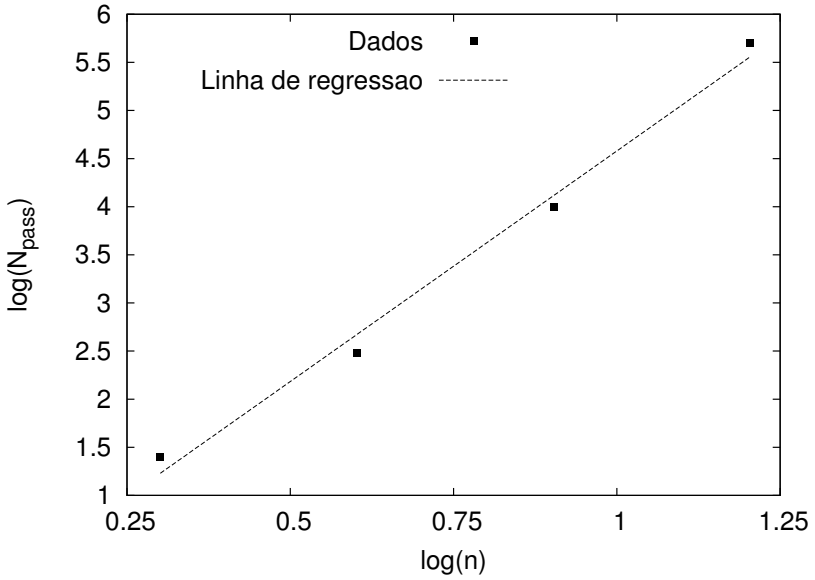

Figura 5.19: *Número de partículas em função da dimensão do espaço para a função Schwefel*

Confirmando o observado por PICCAND *et al.* [51], o PSO (na implementação do IU–SPSO) demonstrou forte queda de desempenho com o aumento da dimensão do espaço. O número de partículas necessário cresce com uma potência entre 1,8 (melhor caso - função de *Ackley*) e 4,8 (pior caso - função de *Schwefel*) da dimensão elevando, de maneira similar, o número de avaliações da função objetivo necessário para atingir uma determinada *robustez*, cujos coeficientes angulares ficaram entre 1,95 e 4,53 para essas funções (vide Tabelas 5.10 e 5.11). Portanto, dos resultados obtidos, conclui-se que o número de partículas no enxame, necessário para garantir sucesso na busca pelo ótimo global em um determinado problema, varia com a dimensão segundo a equação 5.1.

$$N_{pass} = A \cdot (n)^k \tag{5.1}$$

sendo N_{pass} e n, respectivamente, o número de partículas e a dimensão do espaço e A e k são os coeficientes da correlação, os quais são dependentes da função objetivo.

5.2 Análise dos algoritmos paralelos

Na Seção 5.2.1 são apresentadas as análises de escalabilidade dos algoritmos paralelos com o número de processadores do domínio de computação paralela. Em seguida, na Seção 5.2.2 apresentam-se os resultados da aplicação do AIU–PPSO ao problema de estimação de parâmetros de um modelo de quebra e coalescência de gotas definidos em ARAÚJO [1].

5.2.1 Análise de escalabilidade dos algoritmos com o número de processadores

Para analisar a eficiência dos algoritmos paralelos desenvolvidos e a sua escalabilidade com o número de processadores utilizados (N_{proc}), os mesmos foram aplicados na otimização de funções testes definidas na Seção 4.5. Para aumentar o custo de avaliação da função objetivo e com isso emular problemas de grande porte, um *atraso* temporal aleatório, foi acrescido ao cálculo das funções. Todos os resultados apresentados correspondem à resolução com sucesso de 50 experimentos numéricos independentes. Além disso, para todos os problemas foi considerado como critério de convergência o *Critério 3* (o mais robusto à época de obtenção dos resultados) com tolerância absoluta igual a 10^{-4}.

Três tipos de problemas foram resolvidos:

1. *Problemas com baixo custo da* F_{obj};
2. *Problemas com custo homogêneo da* F_{obj};
3. *Problemas com custo heterogêneo da* F_{obj}.

A Tabela 5.12 apresenta os custos de avaliação da função objetivo em segundos para os 3 problemas descritos e para as 3 funções teste utilizadas. Todos os tempos medidos correspondem ao tempo de CPU da máquina (CPU TIME) obtidos com a função *getrusage()*[2]. Para as análises, 31 partículas foram utilizadas. A tolerância de 10^{-4} foi utilizada para caracterização de *sucesso*.

[2]Informações podem ser encontradas em `http://pubs.opengroup.org/onlinepubs/009695399/functions/getrusage.html`.

Tabela 5.12: *Custo de avaliação da função objetivo em segundos para os problemas propostos*

	Função Ackley	*Função Alpina*	*Função Levy*
Problema 1	$0,03172 \pm 0,0057$	$0,0131 \pm 0,0055$	$0,0310 \pm 0,0057$
Problema 2	$1,2629 \pm 0,0171$	$1,2080 \pm 0,0056$	$1,1727 \pm 0,0347$
Problema 3	$1,2493 \pm 0,3599$	$1,1581 \pm 0,3549$	$1,2079 \pm 0,3616$

As figuras 5.20, 5.21 e 5.22 apresentam os resultados para *speedup* (S) e *eficiência* (η) de paralelização obtidos para o *Problema 1* com cada função teste. Nas figuras, a linha tracejada na diagonal representa o speedup ideal, ou seja $S = N_{proc}$.

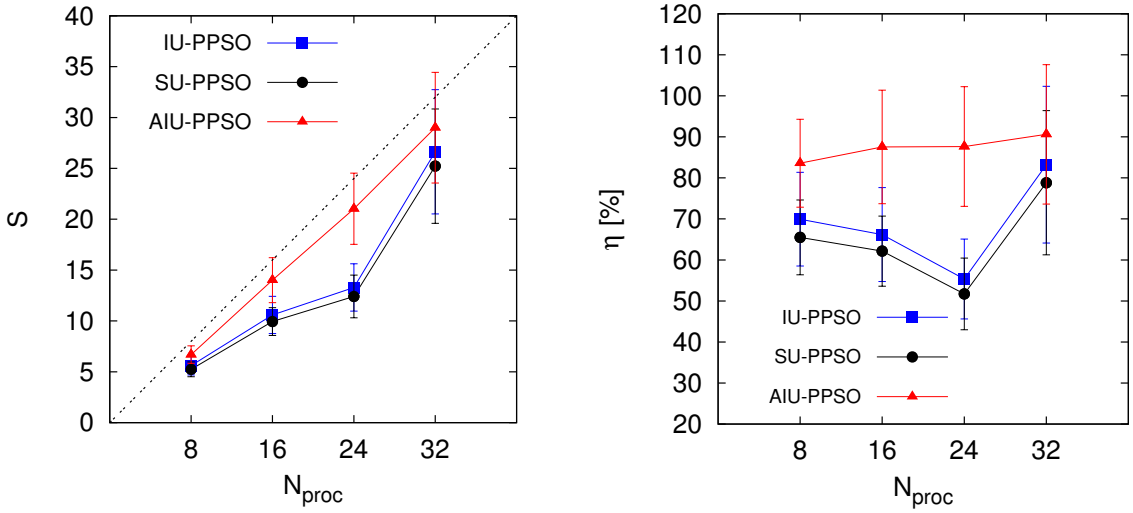

Figura 5.20: *Speedup e eficiência obtidos para a função Alpina no Problema 1*

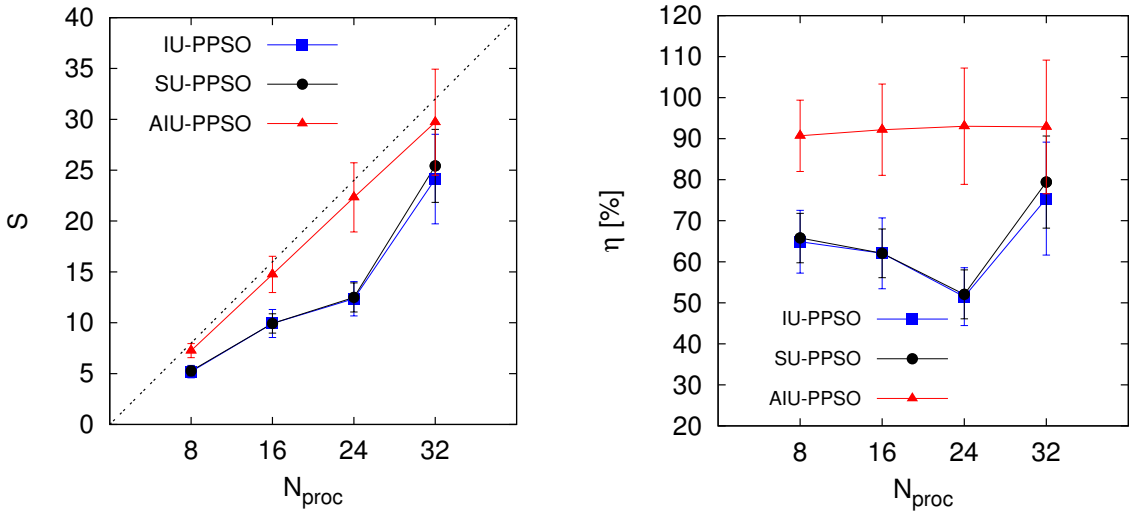

Figura 5.21: *Speedup e eficiência obtidos para a função Ackley no Problema 1*

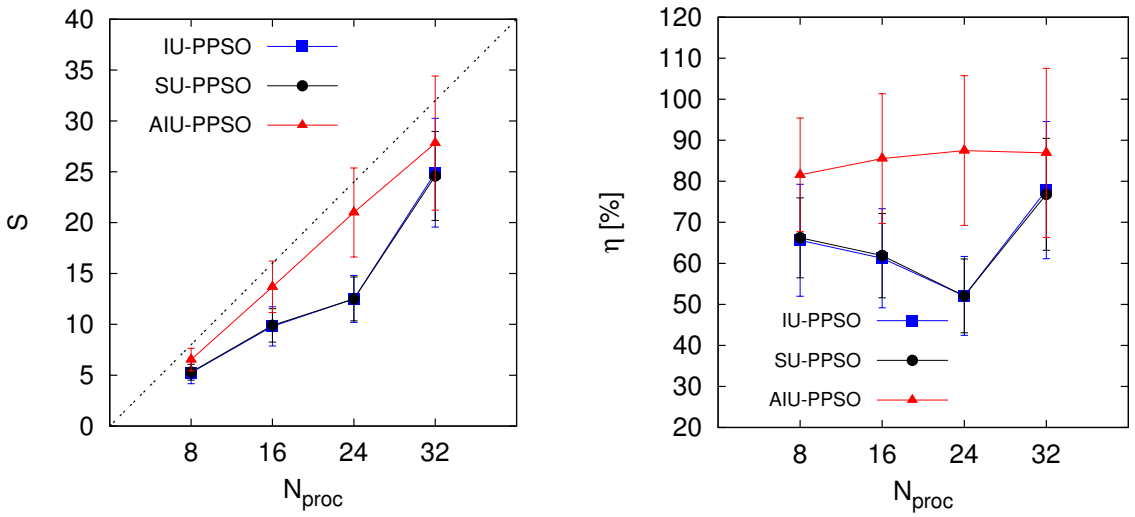

Figura 5.22: *Speedup e eficiência obtidos para a função Levy no Problema 1*

A maior eficiência do AIU–PPSO (observada em todos os casos testados) se deve à natureza da sua paralelização. A assincronia do código permite ao *mestre* realizar os cálculos de teste de convergência ao final de cada *pseudo-revoada* sem que seja necessário interromper o cálculo da função objetivo nos escravos. Como descrito na Seção 3.3, a única perda de tempo devido à paralelização ocorre quando o escravo deseja fazer comunicação com o mestre no momento em este está executando este trecho de código, deixando o escravo momentaneamente ocioso. Por outro lado, nos algoritmos seriais e paralelos síncronos a sincronização ao final da *revoada* torna esta perda de tempo intrínseca.

As figuras 5.23, 5.24 e 5.25 apresentam os gráficos de *speedup* (S) e *eficiência* (η) para as funções Alpina, Ackley e Levy obtidos para o *Problema 2* e as figuras 5.26, 5.27 e 5.28 mostram os resultados para o *Problema 3*.

Os resultados são essencialmente os mesmos que os obtidos no *Problema 1*, ressaltando-se, entretanto, que a eficiência média do AIU–PPSO neste caso ficou em torno de 90 %. Não se observou, portanto, o efeito do grau de variabilidade do tempo da função objetivo sobre o desempenho dos códigos nos três tipos de problemas testados.

Para ratificar o exposto no parágrafo anterior, as figuras 5.29, 5.30 e 5.31 mostram a variação do *speedup* com o número de processadores para os três problemas com a função Alpina para os algoritmos IU–PPSO, SU–PPSO e AIU–PPSO respectivamente. Os resultados são essencialmente os mesmos para as funções de Ackley e Levy.

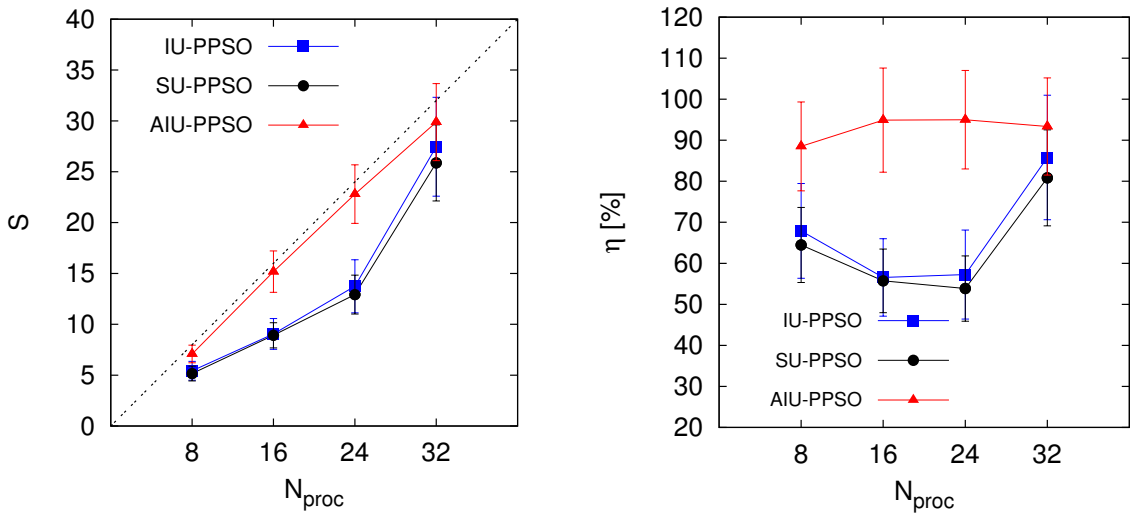

Figura 5.23: *Speedup e eficiência obtidos para a função Alpina no Problema 2*

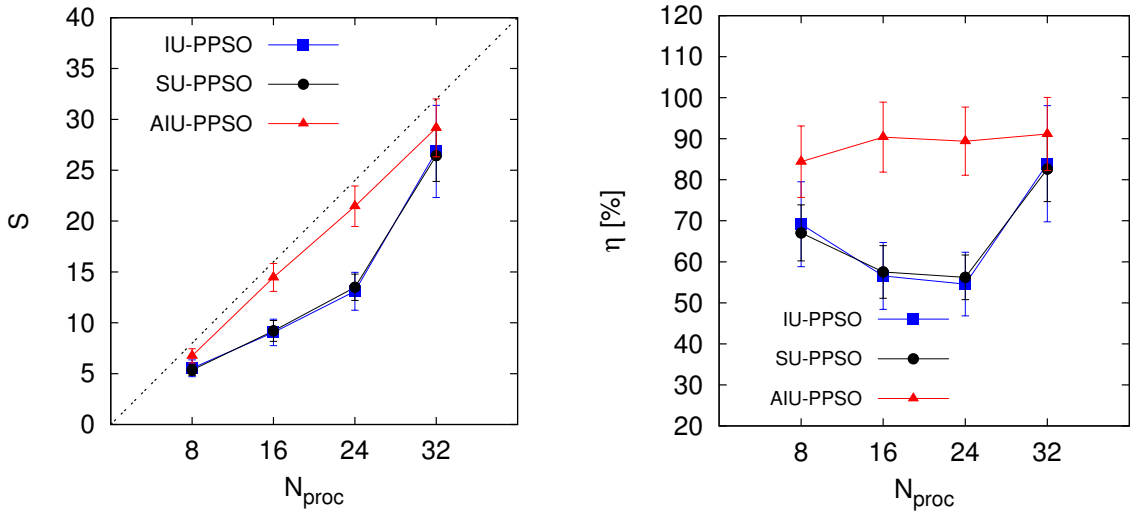

Figura 5.24: *Speedup e eficiência obtidos para a função Ackley no Problema 2*

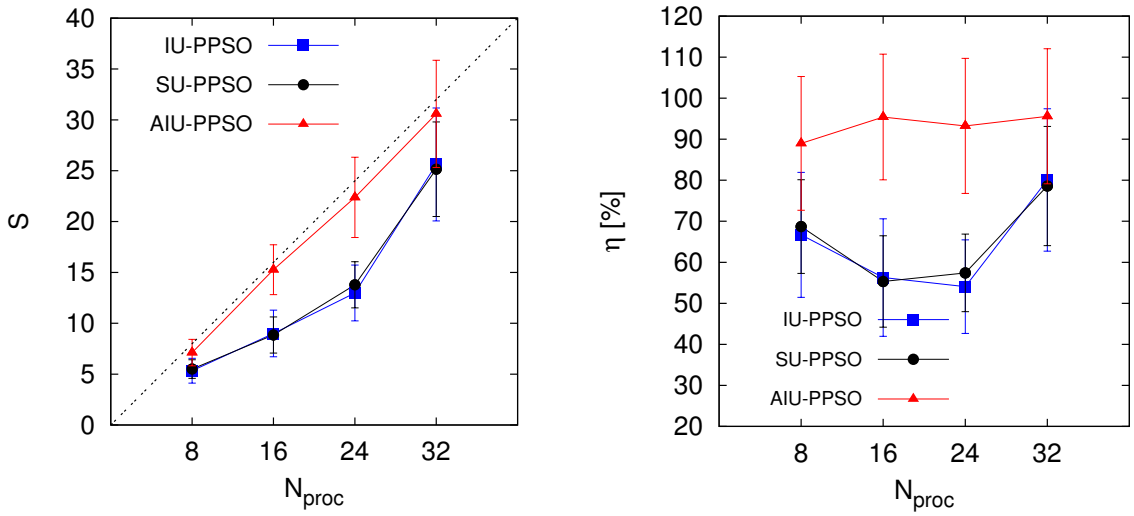

Figura 5.25: *Speedup e eficiência obtidos para a função Levy no Problema 2*

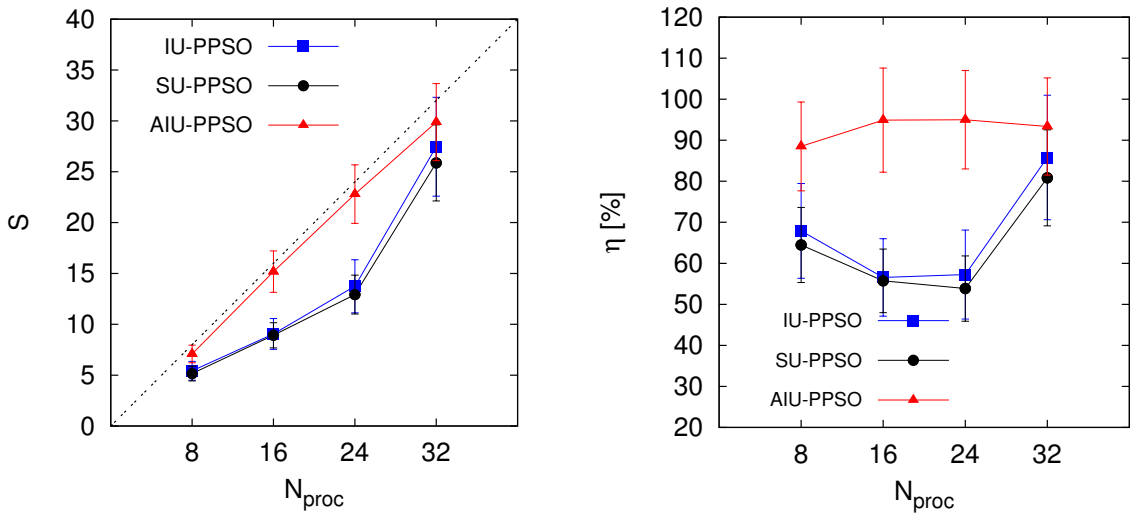

Figura 5.26: *Speedup e eficiência obtidos para a função Alpina no Problema 3*

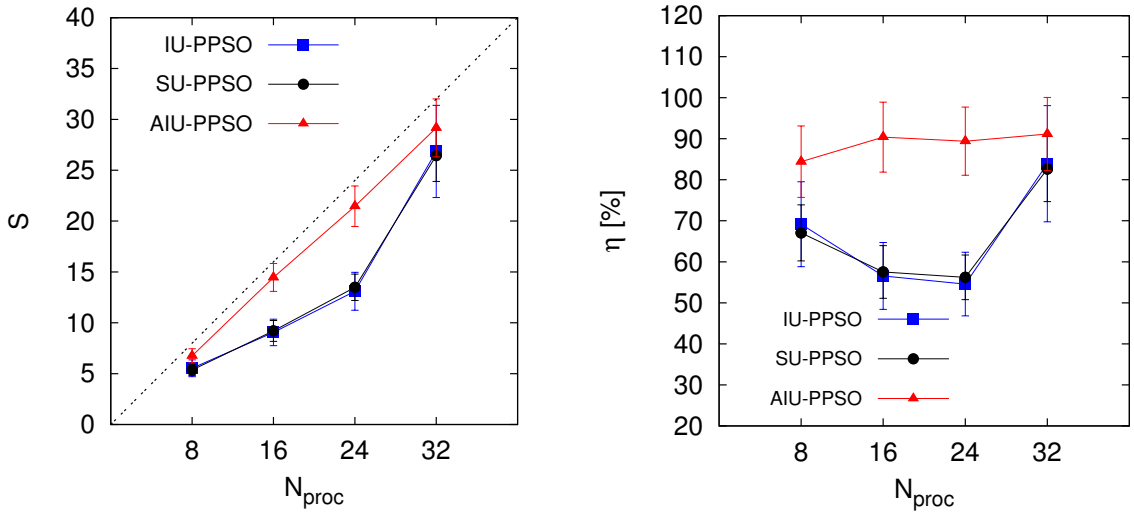

Figura 5.27: *Speedup e eficiência obtidos para a função Ackley no Problema 3*

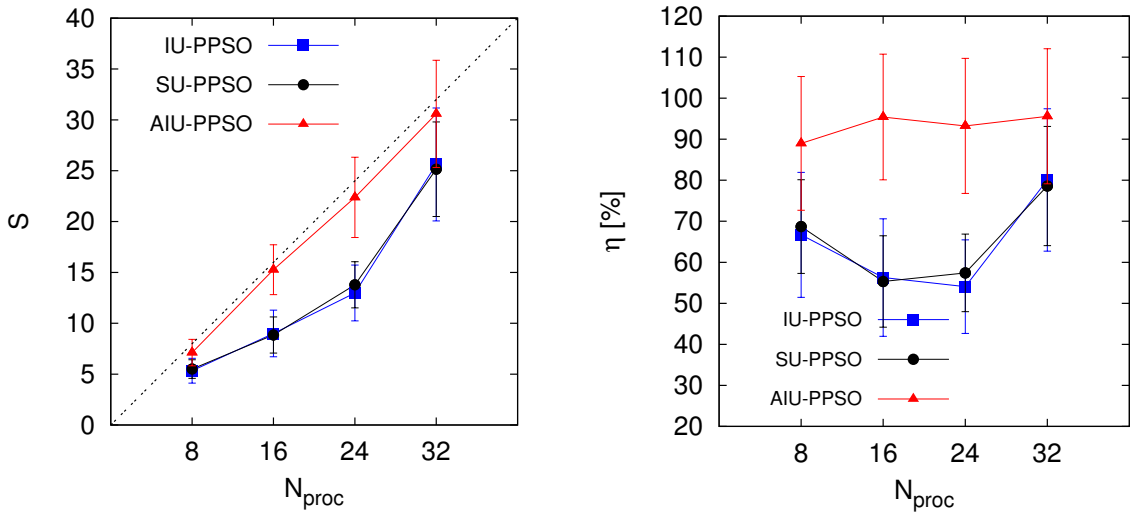

Figura 5.28: *Speedup e eficiência obtidos para a função Levy no Problema 3*

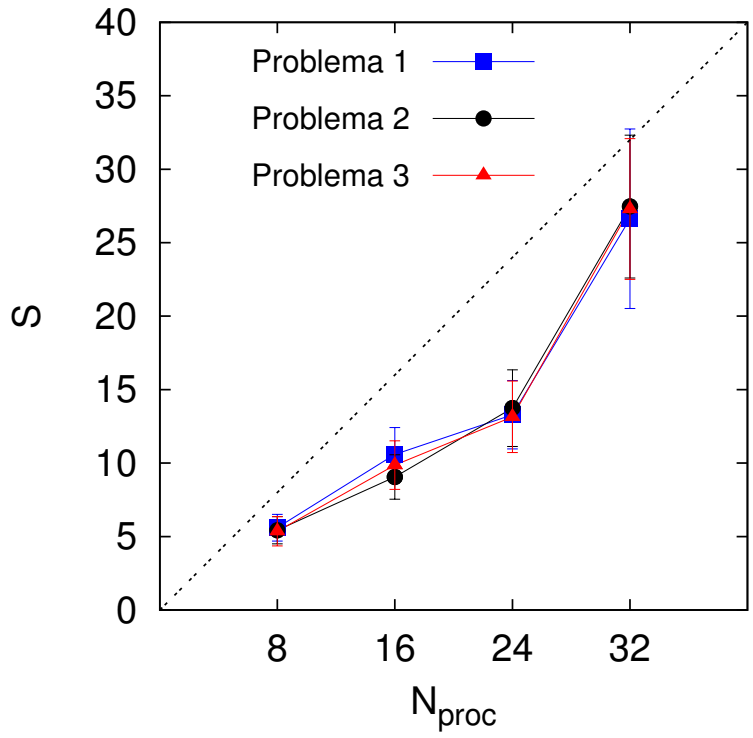

Figura 5.29: *Variação do speedup com o número de processadores para o IU–PPSO com a função Alpina*

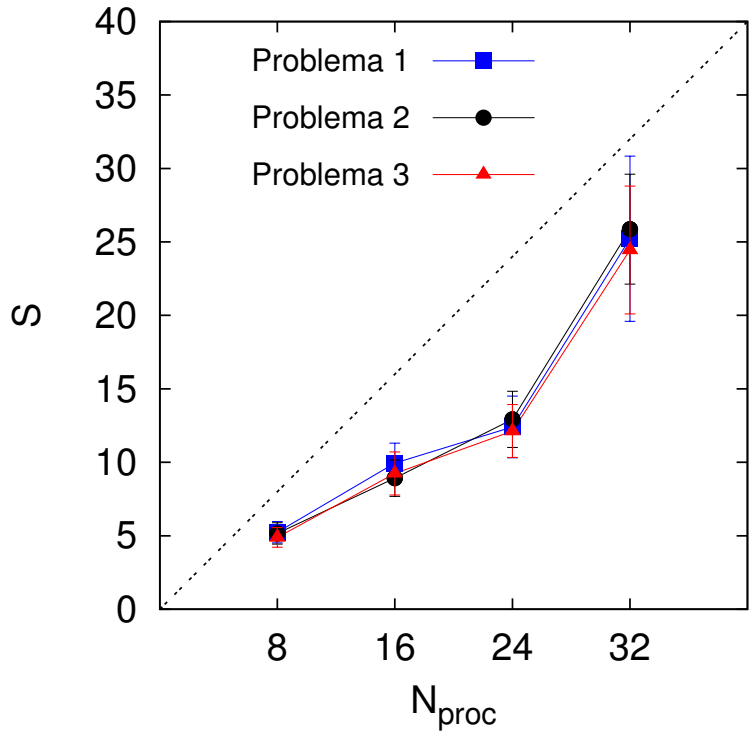

Figura 5.30: *Variação do speedup com o número de processadores para o SU–PPSO com a função Alpina*

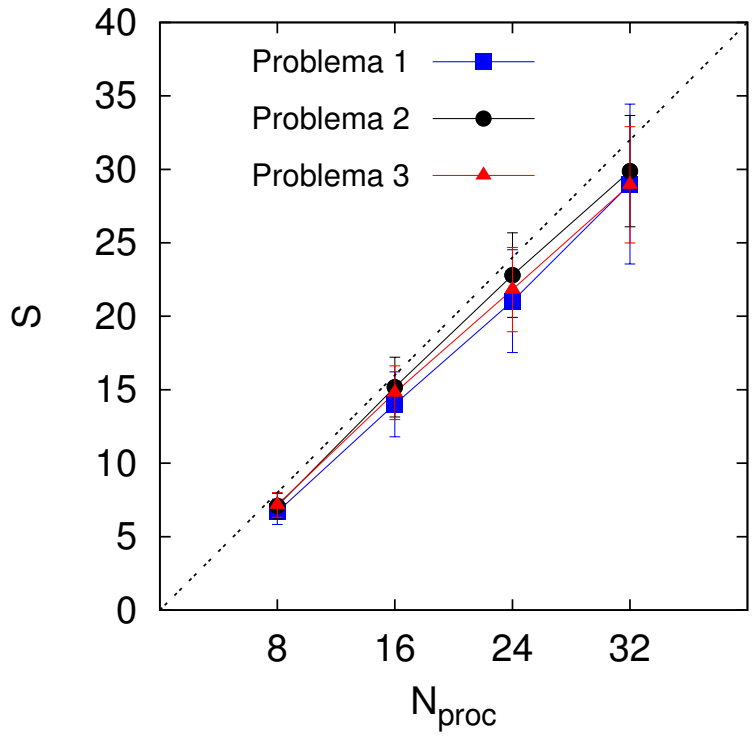

Figura 5.31: *Variação do speedup com o número de processadores para o AIU–PPSO com a função Alpina*

Em todos os casos observou-se um comportamento decrescente da eficência dos algoritmos IU–PPSO e SU–PPSO para até 24 processadores e uma inversão desse comportamento com 32 processadores, onde se aproximam do AIU–PPSO. Esse comportamento pode estar associado ao caso limite que ocorre neste ponto: *o número de processadores escravos é igual ao número de partículas*. Nesta situação, desprezando o custo de comunicação de dados entre os processadores, como todas as partículas são calculadas de uma única vez a cada *revoada*, a ocisiosidade dos escravos nos algoritmos síncronos diminui, tornando-se no máximo igual à diferença entre os tempos do escravo mais lento e mais rápido da *revoada* somado ao tempo de execução do critério de convergência pelo mestre. Ainda, para todos os casos analisados, o comportamento da *eficiência* (apenas dos algoritmos síncronos) com o número de processadores pode ser aproximado pela equação 5.2, sendo esta equação válida para a situação em que o grau de dispersão não afeta o desempenho dos agoritmos paralelos, o que foi o caso dos problemas resolvidos neste trabalho.

$$\eta_{aprox} = \frac{\eta_{max} N_{pass}}{ceil\left(\frac{N_{pass}}{N_{proc}}\right) N_{proc}} \tag{5.2}$$

onde η_{max} é a eficiência máxima imposta pela latência da rede (igual a 80 %) e $ceil(x)$ é o menor inteiro não menor do que x.

As figuras 5.32, 5.33 e 5.34 mostram as comparações entre a eficiência obtida com equação 5.2 e a eficiência dos algoritmos síncronos para as funções Alpina, Ackley e Levy no caso do *Problema 1*.

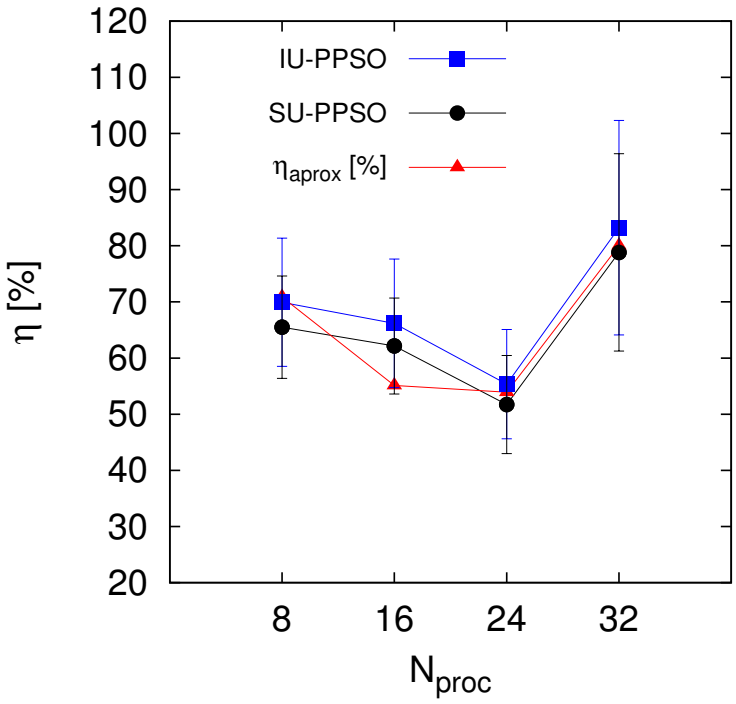

Figura 5.32: *Comparação entre a eficiência dos algoritmos síncronos com a eficiência aproximada pela equação 5.2 para o teste com a função Alpina*

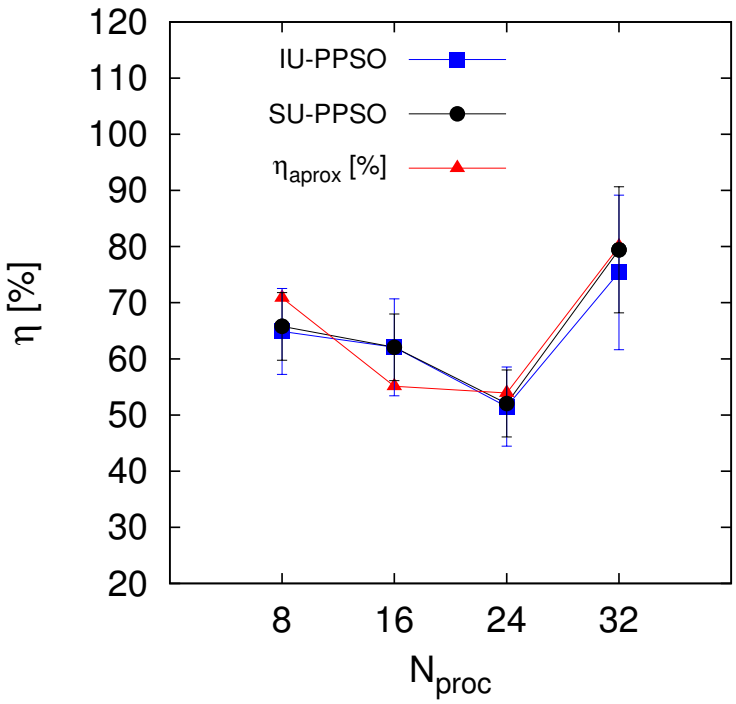

Figura 5.33: *Comparação entre a eficiência dos algoritmos síncronos com a eficiência aproximada pela equação 5.2 para o teste com a função Ackley*

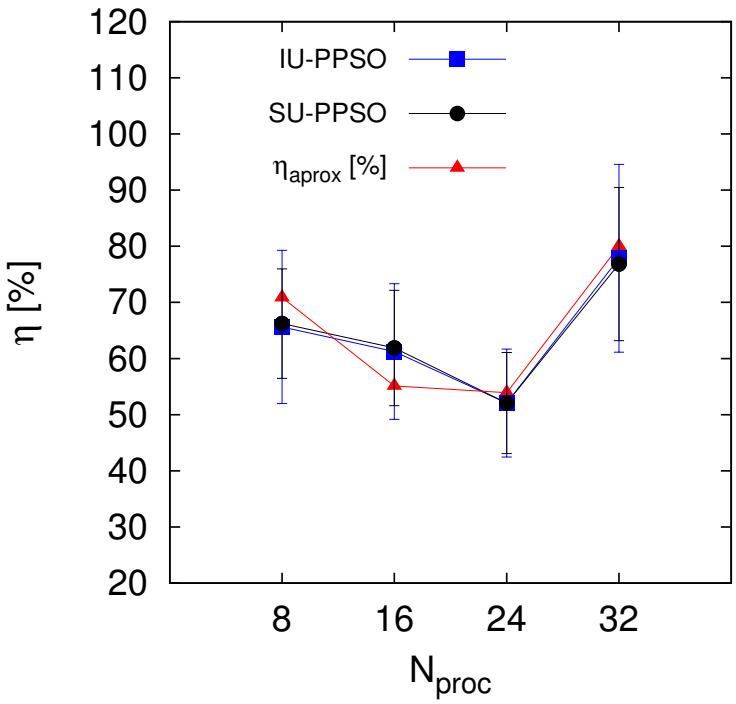

Figura 5.34: *Comparação entre a eficiência dos algoritmos síncronos com a eficiência aproximada pela equação 5.2 para o teste com a função Levy*

Mais dois problemas foram conduzidos utilizando 112 e todos os 128 processadores disponíveis no cluster utilizado. As funções objetivo utilizadas nestes testes foram a função *Schwefel* com dimensão 2 e a função Ackley com 32 dimensões. Para os problemas citados foram utilizados, respectivamente, 150 e 1000 partículas e 50 experimentos independentes foram realizados. A Tabela 5.13 apresenta o custo de avaliação da F_{obj} em segundos.

Tabela 5.13: *Custo da função objetivo para as funções Schwefel 2D e Ackley 32D*

	Tempo de CPU [s]
Função Schwefel 2D	$0,02537 \pm 0,006893$
Função Ackley 32D	$0,03179 \pm 0,005794$

As figuras 5.35 e 5.36 apresentam o *speedup* (S) e a *eficiência* (η) obtida nos problemas supracitados e a figura 5.37 mostra a comparação entre a eficiência dos algoritmos síncronos com aquela aproximada pela equação 5.2 para o caso da função Schwefel, reforçando a validade da aproximação considerada.

O speedup ideal foi obtido com o AIU–PPSO para até os 128 processadores utilizados. Por outro lado, os algoritmos síncronos demonstraram perda constante de eficiência na medida em que o número de processadores é aumentado. Deste modo, pode-se esperar um limite no número de processadores a serem utilizados com os algoritmos síncronos, o mesmo não podendo ser dito para o AIU–PPSO.

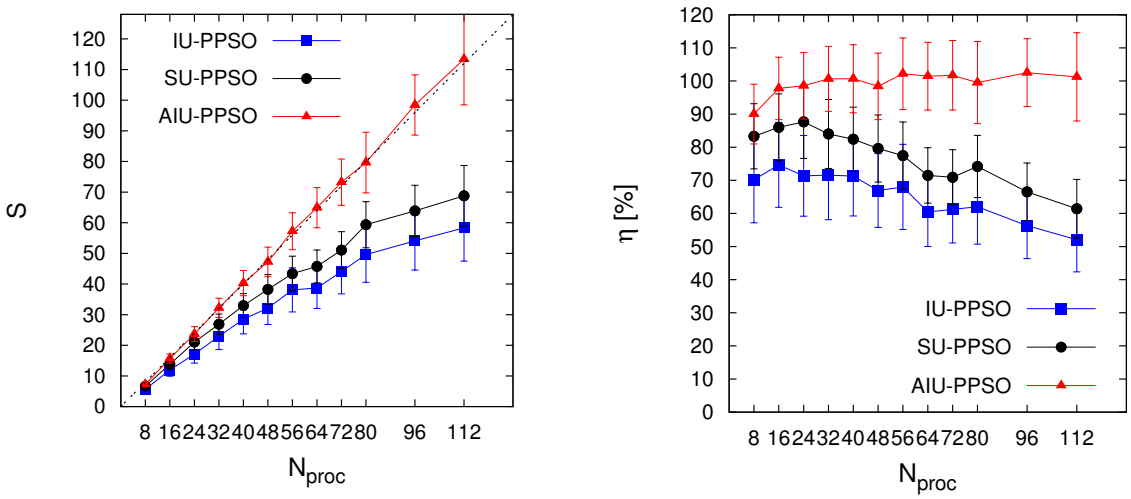

Figura 5.35: *Speedup e eficiência obtidos para a função Schwefel com 2 dimensões*

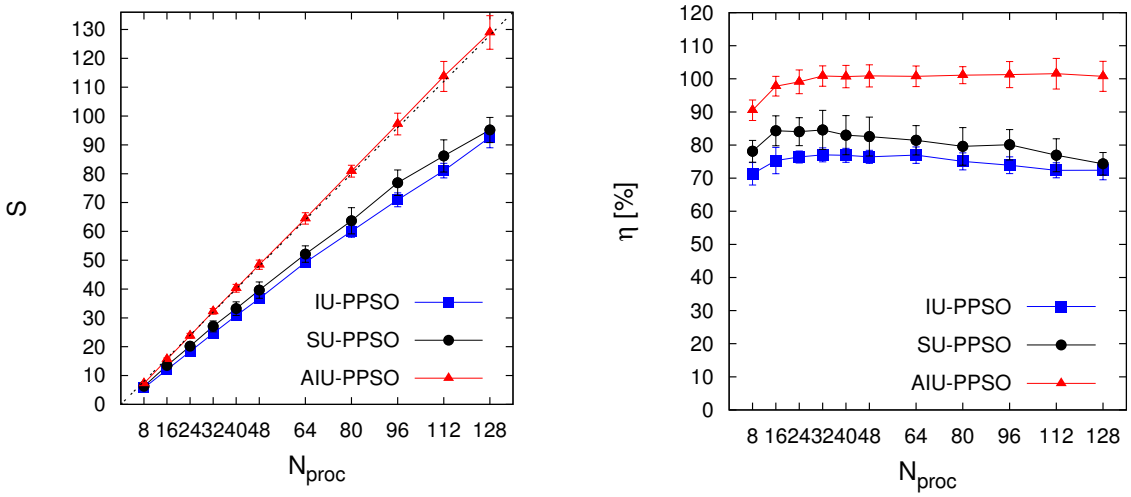

Figura 5.36: *Speedup e eficiência obtidos para a função Ackley com 32 dimensões*

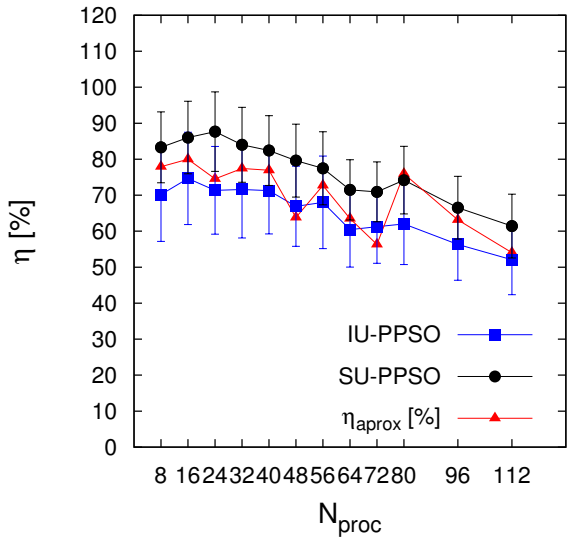

Figura 5.37: *Comparação entre a eficiência dos algoritmos síncronos com a eficiência aproximada pela equação 5.2 para o teste com a função Schwefel*

Análise de escalabilidade conforme KOH *et al.* [4]

Para finalizar o estudo de escalabilidade dos algoritmos paralelos, foram conduzidos experimentos numéricos conforme o problema tratado em KOH *et al.* [4]. Nestes casos, foi feito a otimização da função H_1 conforme descrito nesta referência.

Três problemas para a análise de escalabilidade foram resolvidos em cima desta função, diferindo entre si pela variabilidade inserida no cálculo da função objetivo. A variabilidade foi inserida de maneira aleatória, conforme mostrado na Tabela 5.14. Para a análise de escalabidade, 50 experimentos numéricos independentes foram conduzidos, utilizando o *Critério 2* como critério de convergência. As tolerâncias utilizadas para a convergência do enxame e para a caracterização de sucesso foram 10^{-4} e 10^{-3}, respectivamente. A Tabela 5.15 mostra os parâmetros utilizados nos problemas.

Tabela 5.14: *Custo da função objetivo para a função* H_1

	Tempo de CPU médio [s]	*Intervalo de tempo [s]*
Variabilidade nula	$0,5$	$[0,5, 0,5]$
Variabilidade máxima de 20 %	$0,5$	$[0,5, 0,6]$
Variabilidade máxima de 50 %	$0,5$	$[0,5, 0,75]$

Tabela 5.15: *Parâmetros dos algoritmos utilizados na otimização da função* H_1

Parâmetro	*Valor*
c_1	1,5
c_2	1,5
w_0	0,75
w_f	0,01
N_{pass}	50
$Nmin$	10

As figuras 5.38 e 5.39 mostram o *speedup* e a *eficiência* obtidas para os três problemas resolvidos, com suas respectivas barras de desvio padrão. No gráfico de speedup a linha tracejada na diagonal representa o speedup ideal.

O grau de variabilidade no cálculo da função não afetou o desempenho do AIU–PPSO, da mesma forma que os resultados apresentados por KOH *et al.* [4] para o seu algoritmo assíncrono, o PAPSO. No caso dos algoritmos síncronos, por outro lado, diferente dos resultados apresentados nesta referência com o seu algoritmo síncrono (PSPSO), o grau de variabilidade não apresentou nenhum efeito significativo sobre o desempenho dos algoritmos IU–PPSO e SU–PPSO.

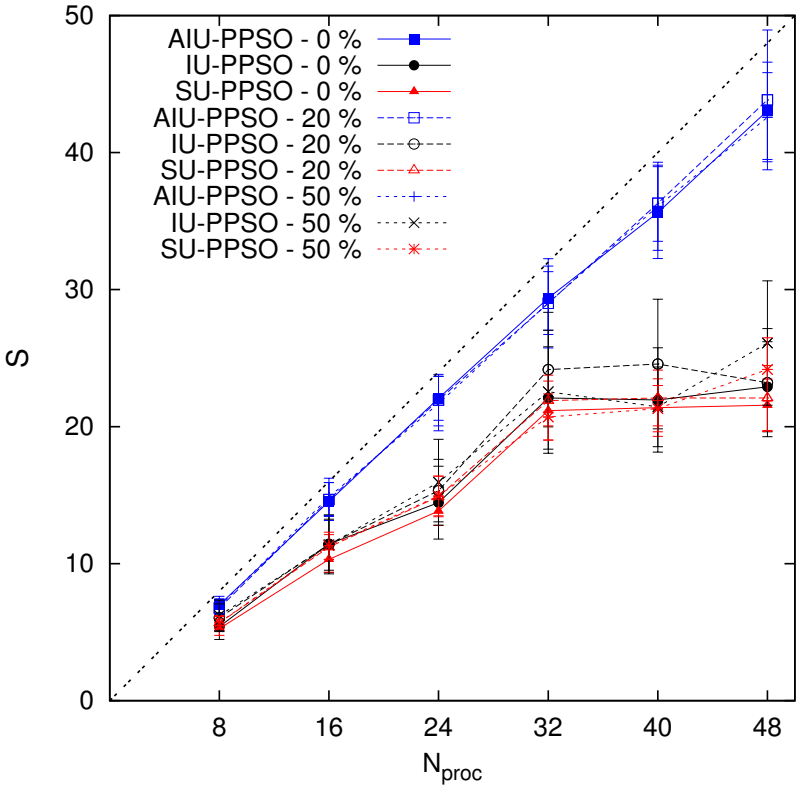

Figura 5.38: *Efeito da variabilidade no cálculo F_{obj} sobre Speedup para a função H_1 com 2 dimensões*

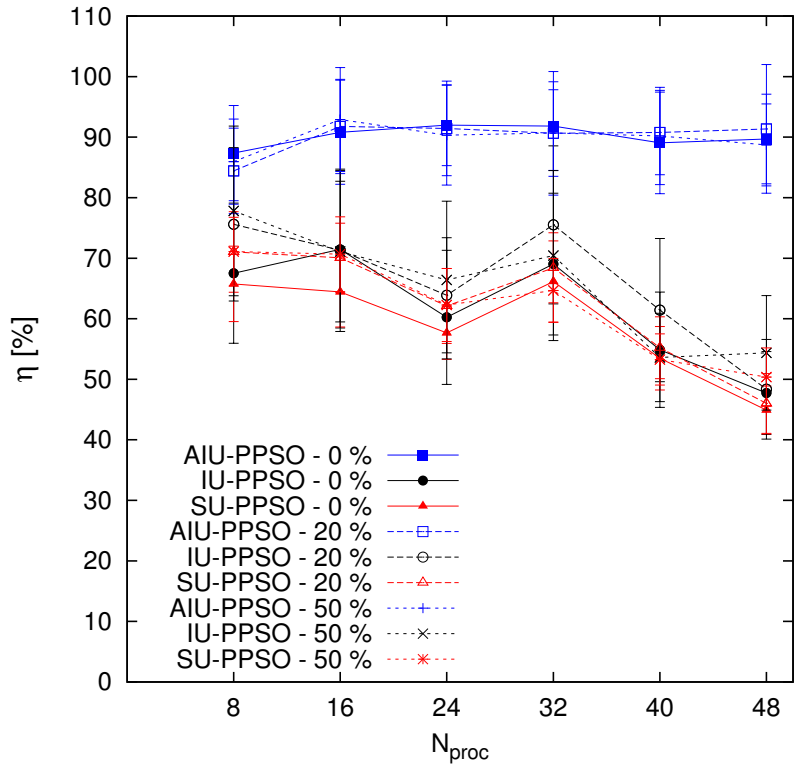

Figura 5.39: *Efeito da variabilidade no cálculo F_{obj} sobre a Eficiência para a função H_1 com 2 dimensões*

Confirmando os resultados obtidos anteriormente, o grau de variabilidade imposto de maneira aleatória sobre o tempo de cálculo da função objetivo não afetou o desempenho dos algoritmos desenvolvidos neste trabalho. Analisando a natureza dos algoritmos síncronos, pode-se inferir que os resultados observados são consistentes, pois a variabilidade no cálculo da função objetivo deve os afetar do mesmo modo que afeta o algoritmo serial, já que os mesmos apresentam o conceito de revoada. Outra questão é que, como a variabilidade é imposta aleatoriamente, a sua distribuição entre as partículas deve ser aproximadamente uniforme entre as revoadas, ou seja, o tempo total de cálculo realizado por cada partícula ao longo das *revoadas* ou *pseudo-revoadas* deve ser aproximadamente igual.

Partindo das hipóteses acima, um problema de avaliação mais consistente seria considerar a variabilidade no cálculo da função objetivo pela posição da partícula no espaço de busca, ou seja, a diferentes posições (ou mesmo regiões) do espaço estariam associados tempos diferentes de cálculo da função objetivo. Este comportamento está presente no problema avaliado na próxima seção.

5.2.2 Aplicação do AIU–PPSO ao problema de estimação de parâmetros de modelo de quebra e coalescência de gotas

O problema desta seção trata da estimação de parâmetros de um modelo de quebra e coalescência de gotas para escoamento de emulsões, desenvolvido por ARAÚJO [1], o qual utiliza técnicas de balanço populacional na sua modelagem. Um resumo desta modelagem é apresentada no Apêndice D. A solução numérica, utilizando o método das classes (RAMKRISHNA [52]), foi desenvolvida em FORTRAN na referência citada.

O objetivo principal foi a aplicação do algoritmo AIU–PPSO a este problema que, devido ao seu porte, seria inviável de ser resolvido utilizando PBM's seriais. Além disso, os resultados foram comparados aos obtidos com o otimizador ODR-PACK95 (ZWOKAK *et al.* [5] apud ARAÚJO [1]).

De modo a simplificar a apresentação (em relação àquela apresentada na formulação do modelo no Apêndice D), considere que existe um modelo que fornece uma relação explícita dada pela equação 5.3 (ARAÚJO [1]).

$$y_i \approx f(x_i; \beta) \tag{5.3}$$

onde y_i e x_i são, respectivamente, os valores das variáveis de resposta e explanatória obtidas experimentalmente, f_i são os valores das variáveis de resposta segundo o modelo utilizado, β são os parâmetros da modelagem e i representa o índice do experimento.

Sendo uma relação aproximada, a seguinte expressão para o erro (ϵ_i) de predição da modelagem em relação ao experimento i é dada pela equação 5.4.

$$\epsilon_i = f_i\left(x_i + \delta_i; \beta\right) - y_i \tag{5.4}$$

onde δ_i são os erros desconhecidos das variáveis explanatórias, dados por:

$$\delta_i = \bar{x}_i - x_i \tag{5.5}$$

onde $\bar{x}_i$ é o valor verdadeiro da variável explanatória, ou seja, o valor tal que quando a função objetivo é mínima, o erro das variáveis de resposta são mínimos, segundo o modelo utilizado.

A função objetivo, F_{obj}, utilizada pelo otimizador ODRPACK95 é dada pela equação 5.6 (ZWOKAK *et al.* [5] apud ARAÚJO [1])

$$F_{obj} = S_\epsilon + S_\delta \tag{5.6}$$

onde S_ϵ e S_δ são as funções de resíduo ponderado para a variável de resposta e explanatória, respectivamente.

A função de resíduo ponderado para a variável de resposta é dada por:

$$S_\epsilon = \sum_i^n \epsilon_i^T W_{\epsilon_i} \epsilon_i \tag{5.7}$$

onde n é o número de experimentos, W_{ϵ_i} é o peso de cada variável de resposta e ϵ_i é o erro desconhecido da variável de resposta (equação 5.4).

A função de resíduo ponderado para a variável explanatória é dada por:

$$S_\delta = \sum_i^n \delta_i^T W_{\delta_i} \delta_i \tag{5.8}$$

onde W_{δ_i} é o peso de cada variável explanatória.

A função objetivo, conforme descrita acima, foi utilizada na otimização com o AIU–PPSO . Para estimar o custo de avaliação desta função em ARES, foram simulados 64 problemas uniformente espaçados no espaço dos três parâmetros principais do modelo (β_1, β_2 e β_3). O custo computacional estimado de avaliação da F_{obj} foi de **1138,93 s** com desvio padrão de **187,65 s**.

Dois problemas foram resolvidos. No primeiro, foram considerados apenas os três parâmetros principais do modelo (β_1, β_2 e β_3) como variáveis de otimização. No segundo, além dos três parâmetros do modelo, o desvio em uma das variáveis explanatórias (experimental), *o volume do acidente* também foi considerado. Tendo sido considerados 78 experimentos, o segundo problema contemplou 81 variáveis de otimização. A Tabela 5.16 apresenta os detalhes dos problemas de estimação que foram resolvidos. O *Critério 3* de convergência foi adotado para os dois problemas,

Tabela 5.16: *Problemas de estimação resolvidos com o AIU–PPSO.*

	Problema 1	*Problema 2*
n	3	81
N_{pass}	200	2000
w_0	1,0	1,0
w_f	0,1	0,1
$c_1 = c_2$	1,5	1,5
tolerância para convergência (ϵ_a)	10^{-4}	10^{-3}
$T_{Fobj}[s]$	$1138,93 \pm 187,65$	

pois, até o momento em que este problema foi avaliado, este era o critério mais robusto disponível. Devido ao seu elevado porte, a tolerância para a convergência do problema com 81 variáveis de otimização foi relaxada ao final do procedimento para 10^{-3}. Contudo, os resultados apresentados, sugerem que o enxame já se encontrava tendendo à convergência.

O *Problema 1*, com apenas três parâmetros do modelo, convergiu em 121 *pseudo-revoadas* e 24079 avaliações da função objetivo, levando 4,66 dias até a convergência. No problema com 81 variáveis, o enxame convergiu em 165 *pseudo-revoadas* e 330119 avaliações, levando 66,72 dias até a convergência.

As figuras 5.40 e 5.41 mostram a evolução dos três parâmetros do modelo no espaço normalizado e a evolução do valor da função objetivo ao longo das *pseudo-revoadas* para o problema com 81 variáveis de otimização. Neste problema, cada *pseudo-revoada* consistiu de 2000 avaliações da função objetivo.

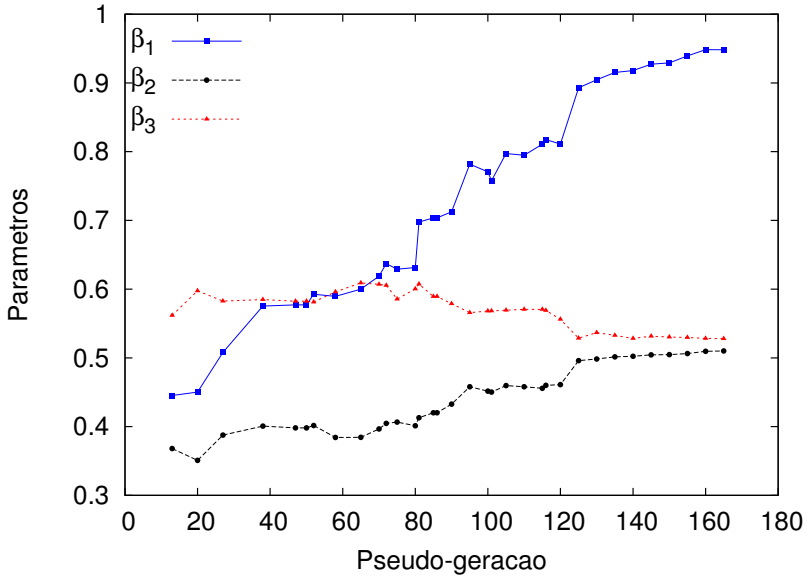

Figura 5.40: *Evolução dos parâmetros do modelo de quebra e coalescência ao longo das pseudo-revoadas no processo de otimização*

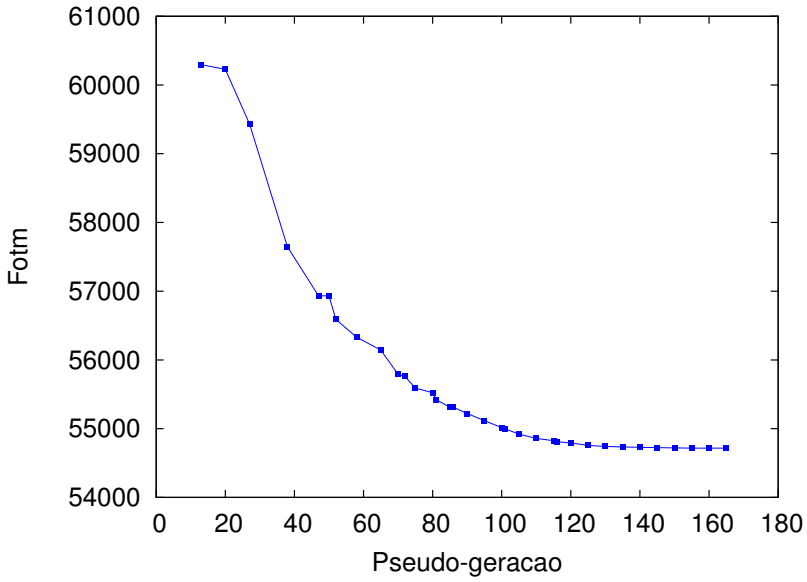

Figura 5.41: *Evolução do valor da função objetivo ao longo das pseudo-revoadas no processo de otimização*

Observa-se que os parâmetros do modelo, assim como o valor da função objetivo, não apresentam mais variações significativas ao após 150 *pseudo-revoadas*, mostrando a tendência à convergência. Ressalta-se que o critério de convergência leva em consideração todos os parâmetros do problema e o valor da função objetivo no espaço normalizado no intervalo [0,1]. Assim sendo, o critério adotado de 10^{-3} sobre o deslocamento do ponto médio entre duas avaliações do mesmo, indica que esse ponto não se deslocou mais do que a fração de 0,001 da maior distância neste espaço no intervalo de, pelo menos, N_{min} *pseudo-revoadas*. Pela figura 5.40, observa-se que o parâmetro β_1 deve ser o que mais está influenciando neste critério. Contudo, tendo em vista a variação muito pequena no valor da função objetivo, seu efeito já não deve ser muito grande.

Um ponto muito importante que deve ser destacado é que a aplicação das implementações seriais do *PSO* ou de qualquer outro *PBM* (Population Based Method) seria inviável neste problema com o atual custo da função objetivo nos nós de ARES. Considerando o número de avaliações que foram necessários da função objetivo (330119), o custo médio de $1138s$ da função objetivo e que todos os demais cálculos do algoritmo tem custo desprezível frente a este valor, uma estimativa para o tempo serial do problema seria aproximandamente de $N_{aval} \cdot T_{F_{obj}}$ que é igual a 11,91 anos.

Este tempo serial pode, de outra maneira, ser utilizado para estimar a eficiência do AIU–PPSO na otimização deste problema. Durante o processo de otimização

utilizou-se, em algumas vezes, devido a demanda do laboratório LTFD, 64 processadores e em outras 80 processadores. Tendo sido utilizado prioritariamente 64 processadores, é conservativo utilizar para essa estimativa de eficiência a média de 72 processadores na solução do problema, o que fornece, a eficiência de $\frac{T_s}{N_{proc}T_p}$ que é igual a 90.5 %. Este valor é totalmente condizente aos valores obtidos de eficiência nos testes de escalabilidade da Seção 5.2.1.

Comparação entre resultados obtidos pelo AIU–PPSO e a ODRPACK95

Os resultados obtidos com o AIU–PPSO foram comparados aos obtidos pelo otimizador ODRPACK95 de ZWOKAK *et al.* [5] na sua utilização na tese de doutorado de ARAÚJO [1].

Antes da comparação das distribuições de saída em si, cabe destacar o ganho em tempo fornecido pelo AIU–PPSO ao problema. Segundo o autor de ARAÚJO [1], a etapa explanatória sobre os três parâmetros do modelo, sem considerar os erros nas demais variáveis de entrada (erros nas variáveis experimentais consideradas no problema de estimação), consumiu cerca de 1 mês de trabalho entre diversas escolhas iniciais dos parâmetros, necessários ao ODRPACK95. Neste caso, com uma população de 200 partículas e apenas uma distribuição aleatória no espaço de busca, um resultado superior foi alcançado em 4.66 dias sem custos humanos adicionais de um processo de análises de resultados nas várias tentativas durante a etapa explanatória.

Tendo destacado esta questão, as distribuições de saída obtidas pela simulação do modelo nos valores ótimos de parâmetros encontrados pelos otimizadores foram comparadas. Para esta comparação não foi utilizado diretamente o valor da função objetivo pelo seguinte motivo: *a possibilidade do otimizador ODRPACK95 atualizar os pesos da função objetivo baseados nos erros das variáveis explanatórias de entrada e de saída do modelo.*

Como não se sabia ao certo se esta implementação da função objetivo foi rigorosamente idêntica em ambos os métodos, preferiu-se utilizar, *a posteriori*, uma função para a comparação entre as distribuições obtidas nos pontos ótimos encontrados pelo o ODRPACK95 e o AIU–PPSO. Para tal, ARAÚJO desenvolveu uma expressão para a comparação entre as distribuições obtidas, descrito a seguir.

Seja, a função de comparação entre distribuições:

$$S_{\omega,y} = \sum_{i}^{n} \frac{1}{\mu_i^2} \omega_{i,y}^T \omega_{i,y} \tag{5.9}$$

onde n é o número de experimentos e μ_i é o primeiro momento da distribuição volumétrica de tamanho de gotas, ou seja, a fração volumétrica (α) da fase dispersa.

A definição de $\omega_{i,y}$ é dada por

$$\omega_{i,y} = f_i - y_i \tag{5.10}$$

onde y_i são valores das variáveis experimentais da distribuição volumétrica de gotas obtida após o acidente e f_i são os respectivos valores determinados pelo modelo utilizado.

Assumindo n_y variáveis da distribuição volumétrica, a expansão da equação 5.9 fornece:

$$S_{\omega,y} = \sum_{i}^{n} \sum_{j}^{n_y} \frac{(f_{i,j} - y_{i,j})^2}{\alpha^2} \tag{5.11}$$

A mesma ideia pode ser utilizada para expressar a função S, em relação a distribuição volumétrica de gotas obtida antes do acidente, definindo, assim, $S_{\omega,x}$ por:

$$S_{\omega,x} = \sum_{i}^{n} \sum_{j}^{n_x} \frac{(\overline{x}_{i,j} - x_{i,j})^2}{\alpha^2} \tag{5.12}$$

onde n_x é o número de variáveis da distribuição volumétrica de gotas antes do acidente e x_i e $\overline{x}_i$ são as distribuições volumétricas de gotas obtidas antes do acidente experimentalmente e determinadas pelo ODRPACK95, respectivamente. Como otimizador global (AIU–PPSO) desenvolvido não considera o erro nas variáveis de entrada (antes do acidente), esta função foi avaliada apenas para os resultados obtidos com o ODRPACK95 quando os erros nestas variáveis foram considerados (Casos 1, 5 e 6 decritos a seguir).

Tendo definido a função de comparação entre as distribuições, os casos enumerados a seguir foram definidos para análise. Em todos eles considera-se que há erros nas variáveis de resposta. Os resultados obtidos pelo otimizador global são referentes ao *Problema 2* da Tabela 5.16. Os resultados para cada caso são apresen-

tados nas Tabelas 5.17 e 5.18.

Caso 0 Dados obtidos com o ODRPACK e publicados ARAÚJO [1], assumindo que as variáveis exploratórias não possuem erro.

Caso 1 Dados obtidos com o ODRPACK e publicados em ARAÚJO [1], assumindo que todas as variáveis possuem erro.

Caso 2 Dados obtidos com o otimizador global AIU–PPSO.

Caso 3 Dados obtidos com o ODR utilizando os volumes ótimos obtidos no Caso 2 e considerando que não existe erro nas variáveis exploratórias.

Caso 4 Dados obtidos com o ODR utilizando os volumes ótimos obtidos no Caso 2 e considerando que apenas os volumes do acidente possui erro.

Caso 5 Dados obtidos com o ODR utilizando os volumes ótimos obtidos no Caso 2 e considerando que existe erro na distribuição volumétrica de tamanho de partículas obtida antes do acidente (ou seja, assume-se que o volume do acidente, a vazão, a temperatura e a pressão não possuem erro).

Caso 6 Dados obtidos com o ODR utilizando os volumes ótimos obtidos no Caso 2 e considerando que existe erro em todas as variáveis exploratórias.

Tabela 5.17: *Tabela com os resultados de $S_{\omega,y}$ para cada caso avaliado*

Caso	$S_{\omega_i,y}$	$S_{\omega_0,y}/S_{\omega_i,y}$
Caso 0	0,305	1,000
Caso 1	0,316	0,965
Caso 2	0,254	1,201
Caso 3	0,254	1,201
Caso 4	0,254	1,201
Caso 5	0,252	1,210
Caso 6	0,267	1,142

Tabela 5.18: *Tabela com os resultados de $S_{\omega,x}$ para cada caso avaliado*

Caso	$S_{\omega_i,x}$
Caso 0	–
Caso 1	$4,91 \times 10^{-5}$
Caso 2	–
Caso 3	–
Caso 4	–
Caso 5	$1,31 \times 10^{-7}$
Caso 6	$6,76 \times 10^{-6}$

A tendência esperada de diminuição do valor da função $S_{\omega,y}$ é observada entre os casos 1 a 4, porém, nota-se um aumento no valor de $S_{\omega,y}$ no caso 6. Essa observação é possível, pois, nesse caso, considera-se que a vazão, a temperatura e a pressão consideradas, além do próprio volume do acidente, possuem erro, e todas essas variáveis fazem parte da função objetivo, mas não fazem parte da função $S_{\omega,y}$. A mesma explicação aplica-se a comparação entre o caso 0 e o caso 1.

A qualidade geral do resultado obtido com os volumes do acidente provenientes do otimizador global é cerca de **20 %** melhor do que sem a otimização global, segundo os critérios adotados.

Os resultados dos três parâmetros principais do modelo obtidos em cada caso analisado estão descritos na Tabela 5.19.

Tabela 5.19: *Tabela com os resultados dos parâmetros para cada caso avaliado, sendo C_c o parâmetro de coalescência, C_b o parâmetro de quebra e ς o número de partículas geradas na quebra*

Caso	C_c	C_b	ς
Caso 0	$(1,00 \pm 0,05) \times 10^{-2}$	$(0,98 \pm 0,06) \times 10^{-2}$	$26,0 \pm 0,9$
Caso 1	$(1,05 \pm 0,23) \times 10^{-2}$	$(1,06 \pm 0,80) \times 10^{-2}$	$26,9 \pm 9,4$
Caso 2	$1,90 \times 10^{-2}$	$1,07 \times 10^{-2}$	$31,1$
Caso 3	$(1,90 \pm 0,02) \times 10^{-2}$	$(1,07 \pm 0,01) \times 10^{-2}$	$31,0 \pm 0,3$
Caso 4	$(1,90 \pm 0,10) \times 10^{-2}$	$(1,07 \pm 0,02) \times 10^{-2}$	$31,0 \pm 1,0$
Caso 5	$(1,90 \pm 0,26) \times 10^{-2}$	$(1,07 \pm 0,5) \times 10^{-2}$	$31,2 \pm 14,0$
Caso 6	$(1,88 \pm 0,06) \times 10^{-2}$	$(1,08 \pm 0,7) \times 10^{-2}$	$32,7 \pm 16,8$

Para completar, as figuras 5.42 à 5.45 apresentam algumas das distribuições de saída obtidas do modelo de quebra e coalescência no ponto ótimo encontrado pela ODRPACK95 (Caso 1) e o AIU–PPSO (Caso 2), comparando-as à distribuição de saída obtida experimentalmente em cada um dos 78 pontos experimentais de volume do acidente (eixo horizontal) para os experimentos *pt2g12stdest*, *pt2g13stdest*, *pt3g12stdest* e *pt3g13stdest*, definidos em ARAÚJO [1]. Para esses experimentos, pode-se observar a melhor aproximação da distribuição obtida com os parâmetros encontrados pelo AIU–PPSO.

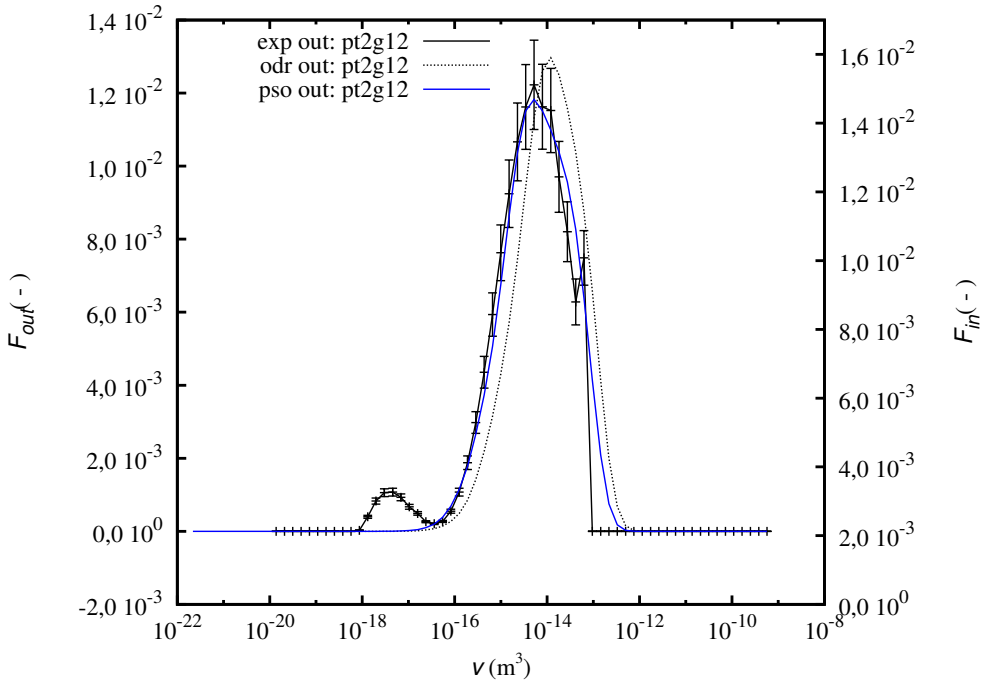

Figura 5.42: *Distribuição de saída do modelo no ponto ótimo encontrado pelos otimizadores. Dados experimentais do experimento pt2g12stdest conforme ARAÚJO [1]*

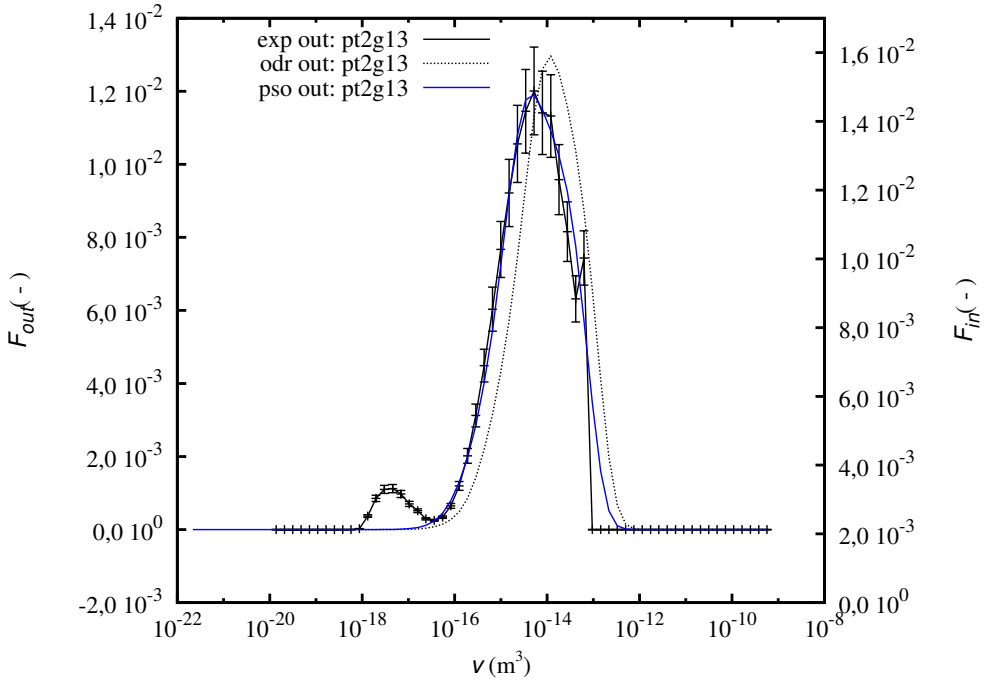

Figura 5.43: *Distribuição de saída do modelo no ponto ótimo encontrado pelos otimizadores. Dados experimentais do experimento pt2g13stdest conforme ARAÚJO [1]*

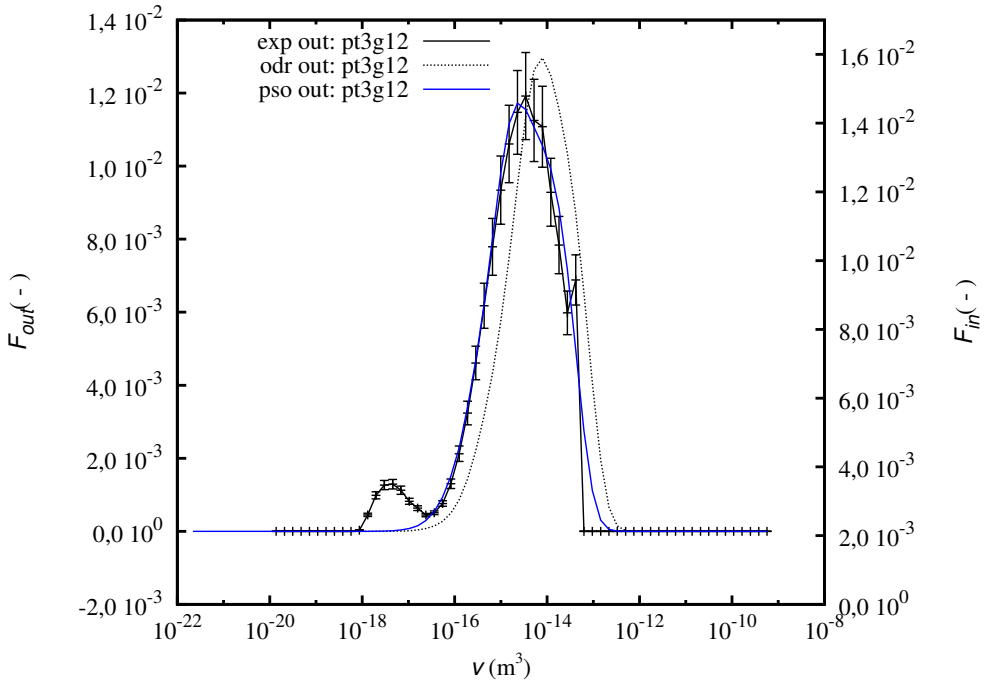

Figura 5.44: *Distribuição de saída do modelo no ponto ótimo encontrado pelos otimizadores. Dados experimentais do experimento pt3g12stdest conforme ARAÚJO [1]*

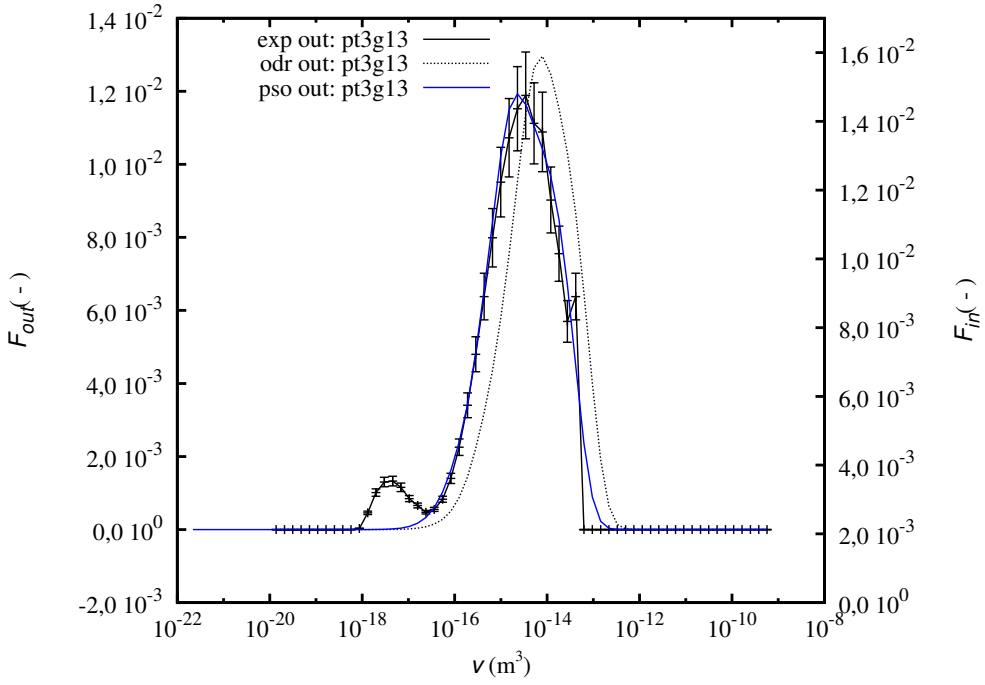

Figura 5.45: *Distribuição de saída do modelo no ponto ótimo encontrado pelos otimizadores. Dados experimentais do experimento pt3g13stdest conforme ARAÚJO [1]*

Capítulo 6

Conclusões

Nesta dissertação foi explorado, principalmente, o desenvolvimento de estratégias paralelas do algoritmo de enxame de partículas (PSO). Em conjunto a esse desenvolvimento algumas características básicas dos algoritmos implementados foram analisadas.

Sobre as características do método

No que diz respeito às características analisadas dos métodos, os seguintes estudos foram realizados:

1. Desenvolvimento de novos critérios de convergência para o enxame;
2. Comparação entre as formas de atualização imediata (*Immediate Update*) e por revoada (*Swarm Update*);
3. Análise da escalabilidade do número de partículas com a dimensão do espaço.

Sobre os critérios de convergência, a utilização de um critério adicional ao número de permanência (N_{min}) do ótimo global (*Critério 1*) aumenta, sobretudo, a *robustez* do método. Em particular, o *Critério 2* (Seção 4.1) aumentou significativamente a *robustez* do método sem comprometer a sua *perfomance*, sendo, por isso, classificado como o de melhor compromisso entre *robustez* e *perfomance* entre os critérios avaliados. O *Critério 4*, juntamente com o *Critério 3*, foram os mais

robustos, às custas, entretanto, de perda de *perfomance*, principalmente no caso do *Critério 3*.

Quanto à forma de atualização das velocidades e posições das partículas, a forma imediata (*Immediate Update*) de atualização foi superior em desempenho à forma de atualização por revoada (*Swarm Update*), apresentando maior *robustez* e *perfomance* em todos os problemas avaliados.

Por fim, o estudo de escalabilidade do número de partículas do enxame com a dimensão do espaço de busca mostrou que o PSO é muito sensível à dimensão do espaço. Os resultados sobre esse estudo (Seção 5.1.3), mostraram que para manter uma determinada taxa de *sucesso* (no caso, acima de 90 %), o número de partículas necessárias no enxame cresce na potência de 2 (função Ackley) a 5 (função Schwefel) da dimensão do espaço e o número de avaliações da função objetivo cresce, como consequência, da mesma maneira. Não se conseguiu obter uma relação de escalabilidade do número de partículas com a dimensão do problema que fosse independente da função objetivo.

Sobre os algoritmos paralelos

Em todos os resultados da análise de escalabilidade dos algoritmos paralelos com o número de processadores (Seções 5.2.1 e 5.2.2), o AIU–PPSO apresentou melhor desempenho, mantendo *speedup's* lineares e eficiência constante (acima de 90 % em média) até todos os 128 processadores do cluster ARES do laboratório de Termofluidodinâmica (LTFD) do PEQ/COPPE/UFRJ. Os algoritmos síncronos, IU–PPSO e SU–PPSO, apresentaram pontos de eficiência máxima, a partir do qual o aumento do número de processadores degrada seu desempenho. Essa perda de eficiência está associada ao ponto de sincronização inerentes a esses algoritmos que os torna limitado, a cada *revoada*, ao escravo mais lento. Esses métodos só apresentaram resultados próximos ao AIU–PPSO no caso limite em que o número de processadores escravos é igual ao número de partículas do enxame (ver Seção 5.2.1), ponto no qual, a forma de implementação torna esses algoritmos muito próximos.

Na análise dos algoritmos conforme realizada em KOH *et al.* [4] (Seção 5.2.2) não se observou o efeito da variabilidade do tempo de cálculo da função objetivo sobre o desempenho dos algoritmos. Esse resultado é compatível ao obtido para o PAPPSO, algoritmo assíncrono desenvolvido nesta referência. Contudo, para os algoritmos síncronos IU–PPSO e SU–PPSO os resultados não apresentaram a perda

de eficiência com o aumento da variabilidade como apresentado por KOH *et al.* [4] para o seu algoritmo síncrono.

Sobre a aplicação ao AIU–PPSO no problema de estimação de parâmetros

A implementação do AIU–PPSO tornou possível a aplicação do PSO a um problema de porte significativamente elevado. Dado o custo de avaliação da função objetivo para este problema, que foi de cerca de *18 minutos* em um processador do cluster ARES, sua resolução através de uma implementação serial do PSO ou de outro MBP seria inviável. Mesmo em comparação com métodos determinísticos locais de otimização, como o próprio ODRPACK95, a utilização do AIU–PPSO pode reduzir o custo de trabalho necessário ao eliminar a necessidade de escolhas de condições iniciais em etapas explanatórias.

Com relação aos resultados obtidos, a utilização do AIU–PPSO melhorou em cerca de 20 % os resultados obtidos com o ODRPACK95 obtidos em ARAÚJO [1], como mostrado na Seção 5.2.2 na comparação das distribuições de saída do modelo em relação aos dados experimentais disponíveis.

Capítulo 7

Sugestões

Dos resultados e conclusões obtidos nesta dissertação e de toda a literatura disponível acerca do algoritmo de enxame de partículas, algumas sugestões são apresentadas a seguir, as quais são divididas em *melhoramento do método* e *melhoramento do código*.

Sobre o melhoramento do método

Como tentou-se destacar ao longo desta dissertação, o grande problema do *PSO* e também de outros *GA's* é o elevado número de avaliações da função objetivo, quando o cálculo dessa função é a etapa mais demandante de tempo computacional. As estratégias paralelas, sob certo ponto de vista, visam compensar este custo de avaliação distribuindo o cálculo de várias partículas sobre computadores paralelos, ou seja, pela realização simultânea de várias avaliações da função objetivo. Por outro lado, a paralelização não altera a natureza do algoritmo e, portanto, não altera significativamente sua *perfomance* (em termos de número de avaliações da função objetivo).

O ideal, portanto, seria utilizar as estratégias paralelas em algoritmos cada vez mais *"inteligentes"*, mais robustos e de alta *perfomance*. O desenvolvimento desses algoritmos mais *inteligentes* é o tema de maior estudo sobre o *PSO* na literatura atual. Como dito anteriormente, a busca na literatura por estratégias melhoradas do *PSO* e a sua implementação (o quanto mais unificada possível) seria um ganho significativo para a obtenção de um código eficiente para a aplicação prática.

Dado o exposto acima, os seguintes pontos de estudo são sugeridos, pontos esses baseados nos resultados desta dissertação e em estudos bibliográficos realizados para a mesma.

1. Desenvolvimento de um critério de convergência híbrido entre os *Critérios 2* e *4* (cf. Seção 4.1);
2. Avaliação de novas relações dinâmicas para descrever o movimento das partículas;
3. Desenvolvimento/implementação de novas topologias de enxames e enxames cooperativos;
4. Avaliação de métricas para o controle do tamanho da população;
5. Avaliação das variantes do PSO em problemas de elevada dimensão.

Com relação aos itens acima, o primeiro deles é conclusão direta dos resultados obtidos na Seção 5.1.1. O *Critério 2* elevou significativamente a *robustez* do algoritmo em relação ao *Critério 1* sem perda de *perfomance* em relação ao mesmo. Por outro lado, o *Critério 4* (junto com o *Critério 3*) foi o mais robusto, às custas de perda de *perfomance* e aumento de sua variabilidade. Assim sendo, um critério híbrido entre eles poderia resultar em um novo critério, ainda mais robusto que o *Critério 2* e com *perfomance* melhor do que o *Critério 4*.

Nos Apêndices B e C apresentaram-se estudos sobre as trajetórias das partículas para uma forma modificada do PSO (BISCAIA *et al.* [21]), baseada na solução analítica de um sistema dinâmico de 2^a ordem, e para a equação clássica de velocidade do algoritmo (VAN DEN BERGH e ENGELBRECHT [2]). Mostrou-se, na análise do PSO-modificado, que o mesmo pode ser escrito na forma clássica, resultando em uma redefinição dos parâmetros que podem garantir a estabilidade do enxame. Esse estudo pode motivar, portanto, novos desenvolvimentos no segmento de sintonia de parâmetros para o *PSO*.

O terceiro item é proposto devido a sua recorrência na literatura. Muitos estudos têm utilizado essa ideia de enxames independentes e cooperativos. Por exemplo, WAINTRAUB *et al.* [30] utilizou essa ideia no desenvolvimento das suas estratégias paralelas obtendo, segundo seus resultados, ganhos significativos em relação à estratégia *mestre-escravo*. Cabe ressaltar que a utilização de enxames cooperativos não restringem o uso de quaisquer variantes do PSO. Assim sendo, cada

sub-enxame de uma estratégia como essa, poderia utilizar a estratégia mestre-escravo assíncrona como desenvolvida nesta dissertação.

Outra estratégia interessante observada na literatura é a de redução do tamanho da população baseado em uma métrica sobre a disposição das partículas no espaço. Esses trabalhos partem do pressuposto de que próximo a convergência, quando as partículas tendem a se concentrar em uma pequena região do espaço, uma população grande já não é mais necessária. Assim sendo, propõem redução do tamanho da população de acordo com as métricas calculadas. Essa estratégia parece ser uma boa solução para melhoria de *perfomance* do PSO e sugere-se maiores estudos sobre a mesma.

Finalizando, seria muito importante avaliar quaisquer variantes já desenvolvidas e implementadas em problemas de elevada dimensão do espaço de busca. Como observado nos resultados da Secção 5.1.3, a *perfomance* do PSO foi degradada devido ao aumento, na ordem de potências de 2 a 5 da dimensão do espaço, do número de partículas necessárias para atingir determinada robustez. Dado que problemas de elevada dimensão são comuns na Engenharia, essa limitação pode levar a questionamentos sobre a aplicabilidade do PSO a esses problemas. Por outro lado, é bom deixar claro que, mesmo com uma população relativamente pequena, o PSO pode levar a resultados melhores dos que os atualmente conhecidos.

Sobre o melhoramento do código

Os problemas industriais são, em sua maioria, problemas de otimização com restrições no espaço de busca e multi-objetivo. Ainda, a formulação das funções objetivo pode utilizar informações advindas de outros *softwares*, sejam eles de código aberto ou fechado. Com isso, o cálculo da função objetivo se transforma em um problema do tipo *black-box*, no qual o usuário têm domínio apenas das entradas e saídas desses softwares.

Assim sendo, para tornar os códigos desenvolvidos nesta dissertação hábeis à aplicação a um leque maior de problemas industriais, alguns pontos devem ser estudados. Os itens a seguir mostram as sugestões para estudos e desenvolvimentos futuros.

1. desenvolver e/ou implementar técnicas para tratamento de restrições;

2. desenvolver o código para otimização multi-objetivo;

3. desenvolver *scripts*, interfaces ou outros artíficios para integração do otimizador desenvolvido com softwares simuladores, sejam eles abertos ou comerciais.

Particularmente, o terceiro item é um tema bastante aberto, pois a integração com os pacotes simuladores pode ser muito dependente do tipo de problema que se está querendo otimizar. Por exemplo, em problemas de otimização paramétrica para projeto de equipamentos que envolvam cálculos de Fluidodinâmica Computacional, seria necessário a integração do otimizador desenvolvido com ferramentas de geração de geometria, ferramentas de geração de malha numérica e com o próprio solver de CFD (Computational Fluid Dynamics) o que é um processo que tende a ser bastante trabalhoso.

Referências Bibliográficas

[1] ARAÚJO, J. F. M. *Modelos de quebra e coalescência de gotas para o escoamento de emulsões*. Tese de D.Sc., PEQ/COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2010.

[2] VAN DEN BERGH, F., ENGELBRECHT, A. "A study of particle swarm optimization particle trajectories", *Information Sciences*, v. 176, n. 8, pp. 937–971, 2006. ISSN: 0020-0255. doi: DOI:10.1016/j.ins.2005.02.003.

[3] SCHUTTE, J., REINBOLT, J., FREGLY, B., et al. "Parallel global optimization with the particle swarm algorithm", *INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING*, v. 61, n. 13, pp. 2296–2315, DEC 7 2004. ISSN: 0029-5981. doi: {10.1002/nme.1149}.

[4] KOH, B.-I., GEORGE, A. D., HAFTKA, R. T., et al. "Parallel asynchronous particle swarm optimization", *INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING*, v. 67, n. 4, pp. 578–595, JUL 23 2006. ISSN: 0029-5981. doi: {10.1002/nme.1646}.

[5] ZWOKAK, J. W., BOGGS, P. T., WATSON, L. T. *ODRPACK95*. Relatório técnico, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, USA, 2004.

[6] EDGAR, T. F., HIMMELBLAU, D. M., LASDON, L. S. *Optimization of chemical processes*. 2 ed. NY, USA, McGraw–Hill chemical engineering series, 1939. ISBN: 0-07-039359-1.

[7] SECCHI, A. R., BISCAIA, J. E. C. *COQ–897 – Otimização de Processos*. Programa de Engenharia Química. Coppe. Universidade Federal do Rio de Janeiro, Rio de Janeiro, BR, 2009.

[8] SAHINIDIS, N., TAWARMALANI, M. *BARON*. Carnegie Mellon University, Department of Chemical Engineering, Pittsburgh, USA, May 2011.

[9] BJORKMAN, M., HOLMSTROM, K. "Global Optimization Using the DIRECT Algorithm in Matlab", *AMO. Advanced Modeling and Optimization*, v. 1, n. 2, 1999. Disponível em: <http://tomopt.com/pubs/glbamo.pdf>.

[10] FINKEL, D. E. *DIRECT Optimization Algorithm User Guide*. Center for Research in Scientific Computation, North Carolina State University, mar 2003. Disponível em: <http://www4.ncsu.edu/~ctk/Finkel_Direct/DirectUserGuide_pdf.pdf>.

[11] LAND, A. H., DOIG, A. G. "An Automatic Method of Solving Discrete Programming Problems", *Econometrica*, v. 28, n. 3, pp. 497–520, 1960. Disponível em: <http://jmvidal.cse.sc.edu/library/land60a.pdf>.

[12] BOSSOIS, A. H., ZAMBON, E., GARCIA, B. B. "Geração atomática e resolução de problemas de programação matemática para a otimização da confiabilidade em redes de distribuição de energia". 2007. Disponível em: <http://ninfa.inf.ufes.br/public_files/reloc/monografia_debora.pdf>.

[13] HOLLAND, J. H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. 1 ed. MA, USA, MIT Press Cambridge, 1992. ISBN: 0-262-08213-6.

[14] METROPOLIS, N., ROSENBLUTH, A. W., TELLER, M., et al. "Equation of State Calculations by Fast Computing Machines", *The Journal of Chemical Physics*, v. 21, n. 6, pp. 1087–1092, June 1953.

[15] BONABEAU, E., DORIGO, M., THERAULAZ, G. "Inspiration for optimization from social insect behaviour", *NATURE*, v. 406, pp. 39–42, jul. 2000.

[16] SECCHI, A. R., PERLINGEIRO, C. A. "Busca Aleatória Adaptativa". In: *Proceedings of XII Congresso Nacional de Matematica Aplicada e Computacional*, pp. 49–52, São José do Rio Preto, SP, 1989.

[17] EBERHART, R., KENNEDY, J. "Particle Swarm Optimization", *Proceedings IEEE International Conference on Neural Networks*, pp. 1942–1948, 1995.

[18] SHI, Y., EBERHART, R. C. "Parameter selection in particle swarm optimization", *In Evolutionary programming VII: Proceedings of the EP98, New York: Springer-Verlag*, 1998.

[19] CLERC, M. "The swarm and the queen: toward a deterministic and adaptative particle swarm optimization", *Evolutionary Computation. CEQ 99*, pp. 1951–1957, 1999. doi: DOI:10.1109/CEC.1999.785513.

[20] SCHWAAB, M. *Avaliação de Algoritmos Heurísticos de Otimização em Problemas de Estimação de Parâmetros*. Dissertação de mestrado, PEQ/COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2005.

[21] BISCAIA, J. E. C., SCHWAAB, M., PINTO, J. C. "Um novo enfoque do método do enxame de partículas", *Proceedings os the Workshop em Nanotecnologia e Computação Inspirada na Biologia 2004*, 2004.

[22] HASSAN, R., COHANIM, B., WECK WECK, O., et al. *A Comparison of Particle Swarm Optimization and the Genetic Algorithm*. Relatório técnico, Massachusetts Institute of Technology, Cambridge, MA, 02139. Disponível em: <http://www.mit.edu/~deweck/PDF_archive/3%20Refereed%20Conference/3_50_AIAA-2005-1897.pdf>.

[23] ARUMUGAM, M. S., RAO, M. "On the improved performances of the particle swarm optimization algorithms with adaptive parameters, cross-over operators and root mean square (RMS) variants for computing optimal control of a class of hybrid systems", *Applied Soft Computing*, v. 8, n. 1, pp. 324–336, 2008. ISSN: 1568-4946. doi: DOI:10.1016/j.asoc.2007.01.010.

[24] CHEN, D., ZHAO, C. "Particle swarm optimization with adaptive population size and its application", *Applied Soft Computing*, v. 9, n. 1, pp. 39–48, 2009. ISSN: 1568-4946. doi: DOI:10.1016/j.asoc.2008.03.001.

[25] HE, S., WU, Q., WEN, J., et al. "A particle swarm optimizer with passive congregation", *Biosystems*, v. 78, n. 1-3, pp. 135–147, 2004. ISSN: 0303-2647. doi: DOI:10.1016/j.biosystems.2004.08.003.

[26] KALIVARAPU, V., FOO, J.-L., WINER, E. "Synchronous parallelization of Particle Swarm Optimization with digital pheromones", *Advances in Engineering Software*, v. 40, n. 10, pp. 975–985, 2009. ISSN: 0965-9978. doi: DOI:10.1016/j.advengsoft.2009.04.002.

[27] ENGELBRECHT, A., VAN DEN BERGH, F. "A Cooperative approach to particle swarm optimization", *Evolutionary Computation, IEEE Transactions on Issue*, v. 8, n. 2, pp. 225–239, 2004. ISSN: 8129285. doi: 10.1109/TEVC.2004.826069.

[28] POTTER, M. A., JONG, K. A. D. "A cooperative coevolucionary approach to function optimization", *III Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature*, pp. 249–257, 1994.

[29] JIANG, Y., HU, T., HUANG, C., et al. "An improved particle swarm optimization algorithm", *Applied Mathematics and Computation*, v. 193, n. 1, pp. 231–239, 2007. ISSN: 0096-3003. doi: DOI:10.1016/j.amc.2007.03.047.

[30] WAINTRAUB, M., SCHIRRU, R., PEREIRA, C. M. "Multiprocessor modeling of parallel Particle Swarm Optimization applied to nuclear engineering problems", *Progress in Nuclear Energy*, v. 51, n. 6-7, pp. 680–688, 2009. ISSN: 0149-1970. doi: DOI:10.1016/j.pnucene.2009.02.004.

[31] PEDERSEN, M., CHIPPERFIELD, A. "Simplifying Particle Swarm Optimization", *Applied Soft Computing*, v. 10, n. 2, pp. 618–628, 2010. ISSN: 1568-4946. doi: DOI:10.1016/j.asoc.2009.08.029.

[32] COELHO, L. S. "An efficient particle swarm approach for mixed-integer programming in reliability-redundancy optimization applications", *Reliability Engineering & System Safety*, v. 94, n. 4, pp. 830–837, 2009. ISSN: 0951-8320. doi: DOI:10.1016/j.ress.2008.09.001.

[33] HE, Q., WANG, L. "An effective co-evolutionary particle swarm optimization for constrained engineering design problems", *Engineering Applications of Artificial Intelligence*, v. 20, n. 1, pp. 89–99, 2007. ISSN: 0952-1976. doi: DOI:10.1016/j.engappai.2006.03.003.

[34] JIE, J., ZENG, J., HAN, C., et al. "Knowledge-based cooperative particle swarm optimization", *Applied Mathematics and Computation*, v. 205, n. 2, pp. 861–873, 2008. ISSN: 0096-3003. doi: DOI:10.1016/j.amc.2008.05.100. Special Issue on Advanced Intelligent Computing Theory and Methodology in Applied Mathematics and Computation.

[35] HU, J., WANG, Z., QIAO, S., et al. "The fitness evaluation strategy in particle swarm optimization", *Applied Mathematics and Computation*, v. 217, n. 21, pp. 8655–8670, 2011. ISSN: 0096-3003. doi: DOI:10.1016/j.amc.2011.03.108.

[36] REINBOLT, J. A., SCHUTTE, J. F., FREGLY, B. J., et al. "Determination of patient-specific multi-joint kinematic models through two-level optimiza-

tion", *Journal of Biomechanics*, v. 38, n. 3, pp. 621–626, 2005. ISSN: 0021-9290. doi: DOI:10.1016/j.jbiomech.2004.03.031.

[37] MENESES, A. A. M., MACHADO, M. D., SCHIRRU, R. "Particle Swarm Optimization applied to the nuclear reload problem of a Pressurized Water Reactor", *Progress in Nuclear Energy*, v. 51, n. 2, pp. 319–326, 2009. ISSN: 0149-1970. doi: DOI:10.1016/j.pnucene.2008.07.002.

[38] PATEL, V., RAO, R. "Design optimization of shell-and-tube heat exchanger using particle swarm optimization technique", *Applied Thermal Engineering*, v. 30, n. 11-12, pp. 1417–1425, 2010. ISSN: 1359-4311. doi: DOI:10.1016/j.applthermaleng.2010.03.001.

[39] SECCHI, A. R. *Simulação dinâmica de processos químicos pelo método da relaxação em forma de ondas em computadores paralelos*. Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 1992.

[40] AMDAHL, G. "Validity of the single processor approach to achieving large scale computer capabilities". In: *Proceedings of AFIPS Conference*, v. 30, pp. 483–485, 1967.

[41] GUSTAFSON, J. L. "Reavaluating Amdahl's Law", *Communications of the ACM*, v. 31, n. 5.

[42] FLYNN, M. J. "Very high speed computing systems". In: *Proceedings of IEEE*, v. 54, pp. 1901–1909, 1966.

[43] SNIR, M., OTTO, S., HUSS-LEDERMAN, S., et al. *MPI. The Complete Reference: Volume 1, The MPI Core*. 2 ed. MA, USA, MIT Press Cambridge, 1998. ISBN: 0-262-69215-5.

[44] BARNEY, B. *Message Passing Interface (MPI)*. Livermore, CA, USA, Lawrence Livermore National Laboratory. Disponível em: <`https://computing.llnl.gov/tutorials/mpi/`>.

[45] PACHECO, P. S. *A User's Guide to MPI*. Departament of Mathematics. University of San Francisco, San Francisco, CA 94117, mar. 1995.

[46] PACHECO, P. S., MING, W. C. *MPI User's Guide in Fortran*. Departament of Mathematics. University of San Francisco, San Francisco, CA 94117, mar. 1995.

[47] ALBA, E., TROYA, J. M. "Analyzing synchronous and asynchronous parallel distributed genetic algorithms", *Future Generation Computer Systems*,

v. 17, n. 4, pp. 451–465, 2001. ISSN: 0167-739X. doi: DOI:10.1016/S0167-739X(99)00129-6.

[48] ALBA, E., NEBRO, A. J., TROYA, J. M. "Heterogeneous Computing and Parallel Genetic Algorithms", *Journal of Parallel and Distributed Computing*, v. 62, n. 9, pp. 1362–1385, 2002. ISSN: 0743-7315. doi: DOI: 10.1006/jpdc.2002.1851.

[49] ALBA, E., LUNA, F., NEBRO, A. J., et al. "Parallel heterogeneous genetic algorithms for continuous optimization", *Parallel Computing*, v. 30, n. 5-6, pp. 699–719, 2004. ISSN: 0167-8191. doi: DOI:10.1016/j.parco.2003.12.011. Parallel and nature-inspired computational paradigms and applications.

[50] CARLISLE, A., DOZIER, G. "An off-the-shelf pso", *Proceedings of the Workshop on Particle Swarm Optimization*, 2001.

[51] PICCAND, S., O'NEILL, M., WALKER, J. "On the scalability of particle swarm optmisation", *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence)*, pp. 2505–2512, 2008. doi: DOI:10.1109/CEC.2008.4631134.

[52] RAMKRISHNA, D. *Population Balance. Theory and Applications to Particulate Systems in Engineering*, v. 1. 1 ed. Indiana, USA, Academic Press, 2000. ISBN: 0-12-576970-9.

[53] PINTO, J. C., SCHWAAB, M. *Análise de Dados Experimentais. I. Fundamentos de Estatística e Estimação de Parâmetros*, v. 1. 1 ed. RJ, Brasil, e-papers, 2007. ISBN: 978-85-7650-136-7.

[54] PETZOLD, L. R. *A description of DASSL*. Applied Mathematics Division, Sandia National Laboratories, Livermore, USA, 1982.

Apêndice A

Calculo do desvio padrão do produto e da divisão de eventos independentes

Seja $f(x)$ uma distribuição contínua de probabilidade de uma variável $x \in X$, onde X é um conjunto de medidas independente. O momento de ordem n da distribuição é definido pela equação A.1. PINTO e SCHWAAB [53]

$$\mu_X^{(n)} = \int_X x^n f_X(x) dx \tag{A.1}$$

Sendo $x \in X$ e $y \in Y$ duas variáveis independentes definidas, respectivamente, nos conjuntos independentes, X e Y, então:

$$\begin{aligned} \mu_X^{(n)} \mu_Y^{(n)} &= \left[\int_X x^n f_X(x) dx \right] \left[\int_Y y^n f_Y(y) dy \right] \\ &= \int_X \int_Y (xy)^n f_X(x) f_Y(y) dx dy = \mu_Z^{(n)} \end{aligned} \tag{A.2}$$

ou seja, para dois experimentos independentes x e y:

$$\mu_Z^{(n)} = \mu_X^{(n)} \mu_Y^{(n)} \tag{A.3}$$

onde Z é o conjunto das variáveis dependentes $z = xy$, com $x \in X$ e $y \in Y$.

Seja ainda a variância de um conjunto X, definido de acordo com a equação A.4. PINTO e SCHWAAB [53]

$$V_X = \sigma_X^2 = \int_X [x - E_X]^2 f_X(x) dx \tag{A.4}$$

onde E_X é a média do conjunto X, definida como:

$$E_X = \mu_X^{(1)} = \int_X x f_X(x) dx \tag{A.5}$$

Expandindo o quadrado na integral da equação A.4, obtemos:

$$V_X = \int_X [x^2 - 2xE_X + (E_X)^2] f(x) dx = \mu_X^{(2)} - (E_X)^2 \tag{A.6}$$

ou seja:

$$V_X = \mu_X^{(2)} - (E_X)^2 \tag{A.7}$$

Utilizando a equação A.3 juntamente com a equação A.7 podemos obter a variância do conjunto Z, ou seja, a variância do produto entre duas medidas independentes, como segue:

$$\begin{aligned} V_Z = \mu_Z^{(2)} - (E_Z)^2 = \mu_Z^{(2)} - (\mu_Z^{(1)})^2 = \mu_X^{(2)} \mu_Y^{(2)} - [E_X E_Y]^2 \\ = [V_X + (E_x)^2][V_Y + (E_Y)^2] - [E_X E_Y]^2 \\ = V_X V_Y + V_X (E_Y)^2 + V_X (E_X)^2 \end{aligned} \tag{A.8}$$

$$V_Z = V_X V_Y + V_X (E_Y)^2 + V_X (E_X)^2 \tag{A.9}$$

Seja $Q = \{q = \frac{x}{y} | x \in X \quad e \quad y \in Y\}$. Fazendo a mudança de variável $\overline{y} = \frac{1}{y}$, com $\overline{y} \in \overline{Y}$, então $Q = \{q = x\overline{y} | x \in X \quad e \quad \overline{y} \in \overline{Y}\}$. Para obter a média e a variância do conjunto Q, precisamos, primeiramente, calcular a média e a variância do conjunto $\overline{Y}$ em termos da média e variância do conjunto Y.

A média do conjunto $\overline{Y}$ é dada pela equação A.19.

$$\mu_{\overline{Y}}^{(1)} = E_{\overline{Y}} = \int_{\overline{Y}} \overline{y} f_{\overline{Y}}(\overline{y}) d\overline{y} = \int_{Y} \frac{1}{y} f_Y(y) dy \tag{A.10}$$

onde $f_{\overline{Y}}(\overline{y})d\overline{y} = f_Y(y)dy$ pois, sendo Y e $\overline{Y}$ interdependentes, eles possuem a mesma distribuição de probabilidades.

A expansão em série de Taylor da fração $\frac{1}{y}$ em torno da média do conjunto Y até o termo de 2^a ordem fornece:

$$\frac{1}{y} \cong \frac{1}{E_Y} - \frac{1}{(E_Y)^2}[y - E_Y] + \frac{1}{E_Y^3}[y - E_Y]^2 \tag{A.11}$$

Substituindo a equação A.11 na equação A.10, temos:

$$E_{\overline{Y}} \cong \frac{1}{E_Y} - \frac{1}{E_Y^2} \int_Y (y - E_y) f_Y(y) dy + \frac{1}{E_Y(y)^3} \int_Y (y - E_Y)^2 f_Y(y) dy \tag{A.12}$$

O segundo termo do lado direito é nulo e a integral do terceiro termo é igual à definição de variância dada pela equação A.4. Assim sendo, a média do conjunto $\overline{Y}$ é dada, em termos da média e variância do conjunto Y, pela equação A.13.

$$E_{\overline{Y}} \cong \frac{1}{E_Y} + \frac{V_Y}{E_Y^3} \tag{A.13}$$

A variância do conjunto $\overline{Y}$ é definida pela equação A.14.

$$V_Y = \mu_{\overline{Y}}^{(2)} - E_{\overline{Y}}^2 = \int_{\overline{Y}} \overline{y}^2 f_{\overline{Y}}(y) d\overline{y} - E_{\overline{Y}}^2 \tag{A.14}$$

$$V_Y + E_{\overline{Y}}^2 = \int_{\overline{Y}} \overline{y}^2 f_{\overline{Y}}(y) d\overline{y} = \int_Y \frac{1}{y^2} f_Y(y) dy \tag{A.15}$$

Aproximando a fração do integrando por uma série de Taylor truncada no termo de 2^a ordem, temos:

$$\frac{1}{y^2} \cong \frac{1}{(E_Y)^2} - \frac{2}{(E_Y)^3}[y - E_Y] + \frac{3}{(E_Y)^4}[y - E_Y]^2 \tag{A.16}$$

Portanto, substituindo a equação A.16 na A.15 e resolvendo a integral, a variância do conjunto $\overline{Y}$ é dada pela equação A.17.

$$V_Y = \frac{1}{E_Y^2} + \frac{3V_Y}{E_{\overline{Y}}^4} - E_{\overline{Y}}^2 \tag{A.17}$$

Sustituindo a equação A.13 na equação A.17, obtemos a expressão para a variância do conjunto $\overline{Y}$ em termos da variância do conjunto Y, dada pela equação A.18.

$$V_Y = \frac{V_Y}{E_Y^4} - \frac{V_Y^2}{E_Y^6} \tag{A.18}$$

Por fim, efetuando as substituições e algebrismos necessários, a média e a variância do conjunto Q são dadas pelas equações A.19 e A.20.

$$\begin{aligned} E_Q = \mu_Q^{(1)} &= \mu_X^{(1)} \mu_{\overline{Y}}^{(1)} = E_X E_{\overline{Y}} \\ &\cong \frac{E_X}{E_Y}\Big[1 + \frac{V_Y}{E_Y^2}\Big] \end{aligned} \tag{A.19}$$

$$\begin{aligned} V_Q = \mu_Q^{(2)} - E_Q^2 &= \mu_X^{(2)} \mu_{\overline{Y}}^{(2)} - E_Q^2 \\ &= \Big[V_X + E_X^2\Big]\Big[V_{\overline{Y}} + E_{\overline{Y}}^2\Big] - E_Q^2 \\ &\cong \frac{\Big[V_X + E_X^2\Big]\Big[V_Y + E_Y^2\Big]}{E_Y^4} - \frac{E_X^2}{E_Y^2} + \frac{V_Y}{E_Y^4}\Big[2V_X - V_Y \frac{E_X^2}{E_Y^2}\Big] \end{aligned} \tag{A.20}$$

Apêndice B

Algoritmo modificado do enxame de partículas. Sintonia dos parâmetros do PSO–Clássico

O algoritmo clássico do enxame de partículas pode ser interpretado como um sistema dinâmico no qual cada um dos indivíduos está sujeito a forças de atração aos pontos de melhor posição individual e melhor posição global do enxame e a uma força de amortecimento da velocidade dada pelo termo inercial. Partindo desse pressuposto, BISCAIA *et al.* [21] propuseram que o movimento das partículas pode ser descritos por um sistema dinâmico linear de 2^a ordem amortecido de acordo com a equação B.1

$$\frac{d^2x_i}{dt^2} = -K[(x_i - X_i)] - 2\xi V_i \tag{B.1}$$

em que:

$$\frac{dx_i}{dt} = V_i \tag{B.2}$$

sujeita às condições iniciais:

$$x_i|_{t=t_k} = x_i^k \tag{B.3}$$

$$\frac{dx_i}{dt}|_{t=t_k} = V_i^k \tag{B.4}$$

sendo os dois termos ao lado direito da equação B.1 correspondentes à atração das partículas ao ponto X_i e ao amortecimento das partículas proporcional à sua velocidade. Os termos K e ξ correspondentes, portanto, ao ganho e ao fator de amortecimento do sistema.

Esta equação pode ser escrita para a velocidade das partículas de acordo com a equação B.5.

$$\frac{dV_i}{dt} + 2\xi V_i + Kx_i = KX_i \tag{B.5}$$

Resolvendo a equação B.5 com o método de Euler explícito e a equação B.2 com o método de Euler implícito, as equações B.6 e B.7 são obtidas.

$$\frac{V_i^{k+1} - V_i^k}{\Delta t} + 2\xi V_i^k + Kx_i^k = KX_i^k \tag{B.6}$$

$$\frac{x_i^{k+1} - x_i^k}{\Delta t} = V_i^{k+1} \tag{B.7}$$

resultando nas equações B.8 e B.9 para a atualização da velocidade e posição das partículas:

$$V_i^{k+1} = (1 - 2\xi\Delta t)V_i^k + K\Delta t[X_i^k - x_i^k] \tag{B.8}$$

$$x_i^{k+1} = x_i^k + \Delta t V_i^{k+1} \tag{B.9}$$

Definido-se $V_i = \frac{v_i}{\Delta t}$ obtém-se:

$$v_i^{k+1} = (1 - 2\xi\Delta t)v_i^k + K\Delta t^2[X_i^k - x_i^k] \tag{B.10}$$

$$x_i^k = x_i^k + v_i^k \tag{B.11}$$

Definindo-se ainda:

$$w = 1 - 2\xi\Delta t \tag{B.12}$$

$$K\Delta t^2 = \phi_1 + \phi_2 \tag{B.13}$$

$$X_i^k = \frac{\phi_1 x_{m,i}^k + \phi_2 x_g^k}{\phi_1 + \phi_2} \tag{B.14}$$

obtém-se:

$$v_i^{k+1} = w v_i^k + \phi_1[x_{m,i}^k - x_i^k] + \phi_2[x_g^k - x_i^k] \tag{B.15}$$

que corresponde exatamente a equação de deslocamento das partículas no algoritmo clássico do PSO (PSO–Clássico) com peso de inércia, conforme proposta por SHI e EBERHART [18]. Nestas equações, $\phi_1 = c_1 r_1$ e $\phi_2 = c_2 r_2$ onde c_1 e c_2 são os parâmetros cognitivo e social e r_1 2 r_2 são dois números sorteados aleatoriamente em cada no intervalo [0,1].

Um procedimento alternativo, proposto por BISCAIA *et al.* [21], é resolver a equação B.1 analiticamente considerando fixos os valores do ganho, do amortecimento e da posição de atração (X_i) em cada passo de tempo ou iteração do algoritmo. Sob essas premissas, a equação B.1 constitui-se em uma equação diferencial ordinária linear de 2^a ordem com coeficientes constantes (em cada passo de tempo ou iteração do algoritmo) e como tal apresenta a seguinte solução:

$$\begin{aligned} x_i^{k+1} = \; & X_i^k + \exp(-\xi_k \Delta t)\{(x_i^k - X_i^k)\cos(\omega_k \Delta t) \\ & + [\xi_k (x_i^k - X_i^k) + V_i^k]\frac{\sin(\omega_k \Delta t)}{\omega_k}\} \end{aligned} \tag{B.16}$$

$$\begin{aligned} V_i^{k+1} = \; & \exp(-\xi_k \Delta t)\Big\{V_i^k \cos(\omega_k \Delta t) \\ & - [(\xi_k^2 + \omega_k^2)(x_i^k - X_i^k) + \xi_k V_i^k]\frac{\sin(\omega_k \Delta t)}{\omega_k}\Big\} \end{aligned} \tag{B.17}$$

em que:

$$\omega_k = \sqrt{K_k - \xi_k^2} \tag{B.18}$$

Esta forma modificada de atualização da posição e velocidade das partículas foi denominada pelo pelos autores de PSO-Modificado. A forma de atualização de SHI e EBERHART [18] é referenciada neste trabalho como PSO-Clássico.

Utilizando ainda as definições B.12 e B.13 realizadas para obtenção da forma clássica de atualização, obtemos:

$$x_i^{k+1} = X_i^k + \exp(\frac{w_k - 1}{2})\Big\{(x_i^k - X_i^k)\cos(\theta_k) + \Big[\frac{1 - w_k}{2}(x_i^k - X_i^k) + v_i^k\Big]\frac{\sin(\theta_k)}{\theta_k}\Big\} \quad \text{(B.19)}$$

$$v_i^{k+1} = \exp(\frac{w_k - 1}{2})\Big\{v_i^k\cos(\theta_k) - \Big[(\phi_1 + \phi_2)(x_i^k - X_i^k) + \frac{1 - w_k}{2}v_i^k\Big]\frac{\sin(\theta_k)}{\theta_k}\Big\} \quad \text{(B.20)}$$

em que:

$$\theta_k = \sqrt{(\phi_1 + \phi_2) - \frac{(1 - w_k)^2}{4}} \quad \text{(B.21)}$$

A condição de estabilidade do sistema é:

$$w_k < 1 \quad \text{(B.22)}$$

E as partículas terão movimento oscilatório se:

$$\phi_1 + \phi_2 > \frac{(1 - w_k)^2}{4} \quad \text{(B.23)}$$

o que é uma região maior do que a apresentada na Figura C.1[1]. Além disso, as partículas convergem para o mesmo ponto que o definido pela equação C.17, ou seja, para média ponderada entre a sua melhor posição e a melhor posição do enxame. A vantagem dessa forma modificada das equações de atualização está na redefinição dos parâmetros em relação ao PSO–Clássico. De fato, a equação B.20 pode ser escrita na mesma forma que a equação 2.7, ao se considerar as redefinições dos parâmetros dadas pelas equações B.24, B.25 e B.26.

[1]No apêndice C será discutida a estabilidade do PSO–Clássico de acordo com o trabalho de VAN DEN BERGH e ENGELBRECHT [2]

$$\hat{w_k} = e^{(\frac{w_k - 1}{2})}\Big[cos\theta_k - (\frac{1 - w_k}{2})\frac{sen\theta_k}{\theta_k}\Big] \tag{B.24}$$

$$\hat{\phi_1} = \phi_1 e^{(\frac{w_k - 1}{2})} \tag{B.25}$$

$$\hat{\phi_2} = \phi_2 e^{(\frac{w_k - 1}{2})} \tag{B.26}$$

Definindo ainda:

$$\alpha_k = e^{(\frac{w_k - 1}{2})}\Big[cos\theta_k + (\frac{1 - w_k}{2})\frac{sen\theta_k}{\theta_k}\Big] \tag{B.27}$$

$$\beta_k = e^{(\frac{w_k - 1}{2})}\frac{sen\theta_k}{\theta_k} \tag{B.28}$$

As equações do PSO–Modificado podem ser reescritas de acordo com a equação B.29 e B.30.

$$v_i^{k+1} = \hat{w_k} v_i^k + \hat{\phi_1}[x_{m,i}^k - x_i^k] + \hat{\phi_2}[x_g^k - x_i^k] \tag{B.29}$$

$$x_i^{k+1} = \alpha_k x_i^k + \beta_k v_i^k + (1 - \alpha_k) X_i^k \tag{B.30}$$

Apêndice C

Análise das trajetórias das partículas

A chave para o sucesso alcançado pelo algoritmo do enxame de partćulas está a forma como as partículas se movimentam através do espaço de busca. A forma clássica do algoritmo com peso de inércia vêm sendo amplamente estudada e utilizada desde que foi introduzida por SHI e EBERHART [18]. Entretanto, grande parte dos estudos (e talvez a maior parte deles) são empíricos, concentrando-se fundamentalmente na análise da influência dos parâmetros do algoritmo (o peso de inércia, o parâmetro cognitivo e o parâmetro social) sobre a eficiência e a robustez do método. SHI e EBERHART [18] observaram, por exemplo, que altos valores do peso de inércia promovem uma busca mais global, com elevados deslocamentos das partículas pelo espaço de busca, ao passo que baixos valores desse parâmetro conduzem a uma busca mais localizada e acelera a convergência do enxame.

Apesar da grande contribuição para o entendimento e melhoramento do método, os estudos empíricos não oferecem informações fundamentadas e tão pouco exatas sobre a influência desses parâmetros. Em outras palavras, esses estudos não fornecem a informação exata sobre o tipo de trajetórias e sobre estabilidade do enxame para um dado conjunto de parâmetros. O entendimento exato de como as partículas se movimentam pelo espaço pode ser de grande auxílio para o melhoramento do método e para a determinação do conjunto de parâmetros que conduzem a trajetórias estáveis.

Neste sentido, VAN DEN BERGH e ENGELBRECHT [2] analisaram teoricamente o comportamento das partículas para o algoritmo clássico com peso de inércia

e demonstraram que o enxame sempre converge para um ponto estável no domínio se escolhas adequadas dos parâmetros forem realizadas. A partir das análises, os autores apresentaram ainda a região de parâmetros para os quais as trajetórias das partículas são estáveis.

A prova da convergência fornecida em VAN DEN BERGH e ENGELBRECHT [2] passa pela determinação dos autovalores da matriz associada à equação de atualização de posição das partículas. A sua demonstração é integralmente fornecida na referência citada e será resumida aqui. Assim sendo, sejam as equações de atualização de velocidade e posição do PSO–Clássico para cada partícula em uma dada dimensão do espaço de busca.

$$v_i^{k+1} = wv_i^k + \phi_1[x_{m,i}^k - x^k] + \phi_2[x_g^k - x_i^k] \tag{C.1}$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \tag{C.2}$$

onde ϕ_1 e ϕ_2 são dois números aleatórios sorteados, respectivamente, nos intervalo $[0, c_1]$ e $[0, c_2]$, sendo c_1 e c_2 os parâmetros cognitivo e social.

Essas equações podem ser adequadamente manipuladas, resultando na equação C.3 de atualização de posição das partículas

$$x_i^{k+1} = (1 + w - \phi_1 - \phi_2)x_i^k - wx_i^{k-1} + \phi_1 x_{m,i}^k + \phi_2 x_g^k \tag{C.3}$$

A forma de recorrência expressa pela equação C.3 pode ser escrita como o produto matricial dado pela equação C.4.

$$\begin{bmatrix} x_i^{k+1} \\ x_i^k \\ 1 \end{bmatrix} = \begin{bmatrix} 1 + w - \phi_1 - \phi_2 & -w & \phi_1 x_{m,i}^k + \phi_2 x_g^k \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i^k \\ x_i^{k-1} \\ 1 \end{bmatrix} \tag{C.4}$$

O equação característica associada à matriz é dada pela equação C.5

$$(1-\lambda)[w-\lambda(1+w-\phi_1-\phi_2)+\lambda^2]=0 \tag{C.5}$$

cujas soluções são:

$$\lambda_1 = 1 \tag{C.6}$$

$$\lambda_2 = \frac{1+w-\phi_1-\phi_2+\gamma}{2} \tag{C.7}$$

$$\lambda_3 = \frac{1+w-\phi_1-\phi_2-\gamma}{2} \tag{C.8}$$

em que

$$\gamma = \sqrt{(1+w-\phi_1-\phi_2)^2-4w} \tag{C.9}$$

Desta forma, dada as condições iniciais $x_i(0) = x_i^0$ e $x_i(1) = x_i^1$, a relação de recorrência explícita é dada pela equação C.10

$$x_i^k = k_1 + k_2(\lambda_2)^k + k_3(\lambda_3)^k \tag{C.10}$$

onde

$$k_1 = \frac{\phi_1 x_{m,i}^k + \phi_2 x_g^k}{\phi_1+\phi_2} \tag{C.11}$$

$$k_2 = \frac{\phi_2(x_i^0 - x_i^1) - x_i^1 + x_i^2}{\gamma(\lambda_1 - 1)} \tag{C.12}$$

$$k_3 = \frac{\phi_1(x_i^1 - x_i^0) + x_i^1 - x_i^2}{\gamma(\lambda_1 - 1)} \tag{C.13}$$

sendo x_i^2 calculado a partir da relação de recorrência da equação C.3

$$x_i^2 = (1+w-\phi_1-\phi_2)x_i^1 - wx_i^0 + \phi_1 x_{m,i}^k + \phi_2 x_g^k \tag{C.14}$$

Observando que quando o radicando de γ é negativo, os autovalores λ_1 e λ_2

são complexos conjugados. Assim sendo, a relação de recorrência dada pela equação C.10 pode ser escrita de acordo com a equação C.15.

$$x_i^k = k_1 + |\lambda|^k[(k_2 + k_3)\cos(\theta k) + j(k_2 - k_3)\sin(\theta k)] \tag{C.15}$$

onde $\theta = arg(\lambda)$ e $j = \sqrt{-1}$

A equação C.15 é divergente para $max(|\lambda_2|, |\lambda_3|) \geq 1$ e convergente para $max(|\lambda_2|, |\lambda_3|) < 1$. No caso convergente, o limite da relação é dado pela equação C.16

$$\lim_{k\to\infty} x_i^k = k_1 = \frac{\phi_1 x_{m,i}^k + \phi_2 x_g^k}{\phi_1 + \phi_2} \tag{C.16}$$

mostrando que o enxame converge para a média ponderada entre a melhor posição da partícula e a melhor posição global do enxame.

Quando o radicando de γ é positivo, a convergência do enxame para o ponto médio definido pelo limite da equação C.16 é dada em termos do maior valor entre os autovalores, sendo que a condição de estabilidade do enxame continua sendo $max(|\lambda_2|, |\lambda_3|) < 1$.

A análise apresentada anteriormente mostra a estabilidade do algoritmo em função dos valores característicos da matriz de recorrência dada na equação C.4, que são funções apenas dos parâmetros do algoritmo, ou seja, o peso de inércia e os parâmetros cognitivo e social. A figura C.1 mostra o conjunto de parâmetros para os quais o enxame é convergente ou divergente. A intensidade dos pontos indica a magnitude do maior valor entre os autovalores definido pelas equações C.7 e C.8. A região em preto corresponde ao conjunto de parâmetros para os quais o enxame é divergente, ou seja $max(|\lambda_2|, |\lambda_3|) \geq 1$, enquanto as região mais clara correspondem àquela para qual o enxame é convergente (estável). Para o conjunto de parâmetros próximos à concavidade da zona esquerda o enxame apresenta maior velocidade de convergência.

O PSO é convergente, portanto, para o conjunto de parâmetros que satisfazem a seguinte relação:

$$w > \frac{1}{2}(c_1 + c_2) - 1 \tag{C.17}$$

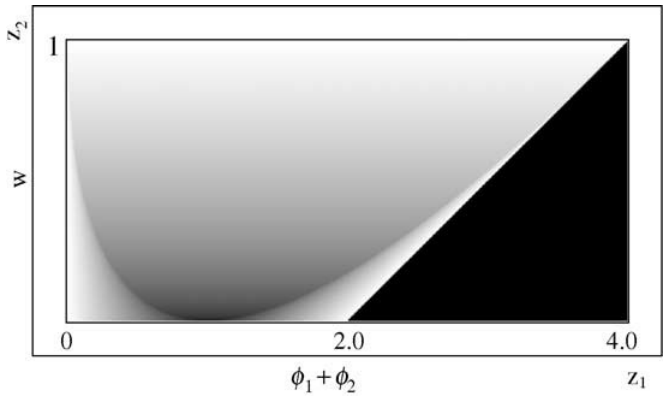

Figura C.1: *Regiões de estabilidade do PSO. O triângulo negro (*$w < \frac{1}{2}(c_1 + c_2) - 1$*) corresponde ao conjunto de parâmetros para os quais o PSO é divergente. Na região à esquerda (*$w > \frac{1}{2}(c_1 + c_2) - 1$*) o PSO é convergente e a velocidade de convergência é maior para as áreas mais escuras do gráfico. Fonte: VAN DEN BERGH e ENGELBRECHT [2]*

sendo w, c_1 e c_2, respectivamente, o peso de inércia, o parâmetro cognitivo e o parâmetro social.

Apêndice D

Modelo de quebra e coalescência de gotas

A modelagem utilizada por ARAÚJO [1] para o escoamento de emulsões com quebra e coalescência de gotas é fundamentada na *equação de balanço populacional* (PBM - *Population Balance Equation*), que descreve a evolução no espaço e no tempo de uma função de distribuição de densidade numérica de partículas. Esta equação é obtida pela integração da equação de Boltzmann aplicada a problemas de muitos corpos, forncendo a conservação da função de distribuição de densidade numérica de partículas , $f_\alpha(\mathbf{z}, \mathbf{v}, t)$ em um processo que inclui as interaçõe possíveis entre as mesmas (ARAÚJO [1]).

A equação de balanço populacional é dada por (RAMKRISHNA [52]):

$$\begin{aligned}\frac{\partial f_\alpha(\mathbf{z},\mathbf{v},t)}{\partial t} &= -\nabla_z \cdot \Big[\dot{\mathbf{Z}} f_\alpha(\mathbf{z},\mathbf{v},t)\Big] + \Big[\mathbf{D}_z\Big(\nabla_z \cdot \mathbf{D}_z^T f_\alpha(\mathbf{z},\mathbf{v},t)\Big)\Big] \\ &- \nabla_v \cdot \Big[\dot{\mathbf{V}} f_\alpha(\mathbf{z},\mathbf{v},t)\Big] + \Big[\mathbf{D}_v\Big(\nabla_v \cdot \mathbf{D}_v^t f_\alpha(\mathbf{z},\mathbf{v},t)\Big)\Big] \\ &+ H\Big(f_\alpha; \mathbf{z},\mathbf{v},t\Big)\end{aligned} \tag{D.1}$$

Nesta equação, $\mathbf{v}$ são as chamadas *variáveis internas* e designam as características segundo as quais as partículas estão distribuídas, como tamanho, composição, energia interna, idade, etc. e $\mathbf{z}$ são as coordeanadas espaciais, chamadas de *variáveis externas*. A função $f_\alpha(\mathbf{z}, \mathbf{v}, t)$ é chamada de *função de distribuição de*

densidade numérica de partículas e mede a o número de partículas por unidade de volume com determinadas propriedades e por unidade destas propriedades internas, sendo que o índice α representa a respectiva fase.

Na equação D.1 os dois primeiros termos ao lado direito da igualdade representam a movimentação das partículas no espaço físico. O primeiro considera a movimentação por efeitos determinísticos, sendo $\dot{Z}$ a taxa de variação da variável externa (a velocidade). O segundo termo considera a movimentação por efeitos estocásticos, onde $\mathbf{D}_z \cdot \mathbf{D}_z^T$ representa o coeficiente de difusão anisotrópica de $f_\alpha()$. O terceiro e quarto termos representam a movimentação das partículas no espaço de variáveis internas por efeitos determinísticos e estocásticos (convecção e difusão da $f_\alpha()$ em relação às vari'aveis internas). Por fim, o último termo ao lado direito da igualdade representa o termo fonte que modela efeitos de quebra e de agregação das partículas. Este termo é dado pela equação D.2.

$$H\Big(f_\alpha; \mathbf{z}, \mathbf{v}, t\Big) = B_B - D_B + B_C - D_C \tag{D.2}$$

sendo B e D os termos de nascimento e morte das partículas causados por mecanismos de quebra e agregação respectivamente.

Em sua tese, ARAÚJO [1] descreve os termos de nascimento e morte para o caso de uma distribuição monovariada, em que $\mathbf{v} = [v]$ é dado por uma única variável aditiva (v) que descreve o tamanho, massa ou volume, da partícula. As equações D.3 a D.6 apresentam os termos de nascimento e morte por quebra (B - Break) e coalescência (C - Coalescence).

$$B_C\left(t, v, \mathbf{z}, \mathbf{y}\right) = \frac{1}{2}\int_0^v f_\alpha\left(t, v - v', \mathbf{z}\right) f_\alpha\left(t, v', \mathbf{z}\right) a\left(v - v', v', \mathbf{y}\right) \mathrm{d}v' \tag{D.3}$$

$$D_C\left(t, v, \mathbf{z}, \mathbf{y}\right) = \int_0^\infty f_\alpha\left(t, v, \mathbf{z}\right) f_\alpha\left(t, v', \mathbf{z}\right) a\left(v, v', \mathbf{y}\right) \mathrm{d}v' \tag{D.4}$$

$$B_B\left(t, v, z, \mathbf{y}\right) = \int_{v'}^\infty \varsigma\left(v', \mathbf{y}\right) P\left(v|v', \mathbf{y}\right) b\left(v', \mathbf{y}\right) f_\alpha\left(t, v', \mathbf{z}\right) \mathrm{d}v' \tag{D.5}$$

$$D_B\left(t, v, \mathbf{z}, \mathbf{y}\right) = b\left(v, \mathbf{y}\right) f_\alpha\left(t, v, \mathbf{z}\right) \tag{D.6}$$

onde o vetor $\mathbf{y} = \mathbf{y}(t, \mathbf{z})$ representa as variáveis da fase contínua que interferem nos processos de quebra e agregação, $a\left(v, v', \mathbf{y}\right)$ é a frequência de agregação local das

partículas de propriedades v e v', $b\left(v', \mathbf{y}\right)$ é a frequência de quebra da partícula de de propriedade v', $\varsigma\left(v', \mathbf{y}\right)$ é o número de filhas produzidas na quebra da partícula de propriedade v' e $P\left(v|v', \mathbf{y}\right)$ é a probabilidade condicional de uma partícula de propriedade v ser gerada quando da quebra de uma partícula de propriedade v'. Nestas equações, assumiu-se a hipótese de que a quebra e a coalescência são fenômenos locais. Dessa forma as funções de quebra e coalescência (a, b, ς, P) somente dependem de (t, $\mathbf{z}$) através das variáveis da fase contínua.

Para fechamento das equações é necessário modelar as funções $b\left(v', \mathbf{y}\right)$,$\varsigma\left(v', \mathbf{y}\right)$ e $P\left(v|v', \mathbf{y}\right)$ para a quebra das partículas e a frequência de agragação $a\left(v, v', \mathbf{y}\right)$. Os capítulos 4 e 5 de ARAÚJO [1] apresentam uma extensa revisão bibliográfica de modelos disponíveis para esta modelagem.

O modelagem do acidente resolvido por ARAÚJO [1] para a realização da estimação de parâmetros considerou as seguintes simplificações:

- a distribuição é monovariada no tamanho, ou seja, apenas uma única variável interna, o volume da partícula, $\mathbf{v} = v$;
- a movimentação da partícula é fundamentalmente devido a movimentação convectiva, ou seja, as partículas não se deslocam devido a efeitos difusivos;
- o termo convectivo e estocástico da variável interna podem ser desprezados, pois pode-se dizer que não crescimento da partícula;
- o problema é estacionário.

O que transforma a equação D.1 na equação D.7

$$\nabla_{\mathbf{z}} \cdot \left[\dot{\mathbf{Z}} f_\alpha(\mathbf{z}, v, t)\right] = H(\{f_\alpha\}; \mathbf{z}, v, t) \tag{D.7}$$

ARAÚJO [1] considerou ainda, baseado no custo computacional do acoplamento PB-CFD (dado em termos do campo de velocidades ($\dot{Z}$) advindo da solução das equações de dinâmica dos fluidos) e no fato dos dados experimentais não fornecerem uma caracterização espacial de $f_\alpha(\mathbf{z}, \mathbf{v}, t)$ (os dados foram obtidos para a entrada e saída do equipamento estudado), simplificações que transformam o sistema em *0-dimensional*. Para a fase contínua incompressível, o modelo resolvido é dado pela equação D.8

$$\frac{\mathrm{D}f_\alpha(v,t)}{\mathrm{D}t} \doteq \dot{\mathbf{Z}} \cdot \nabla_{\mathbf{z}} f_\alpha(\mathbf{z},v,t) = H(\{f_\alpha\};v,t) \tag{D.8}$$

onde $H(\{f_\alpha\};v,t)$ é dado por:

$$B_C(t,v) = \frac{1}{2}\int_0^v f_\alpha(t,v-v')\, f_\alpha(t,v')\, a(v-v',v')\,\mathrm{d}v' \tag{D.9}$$

$$D_C(t,v) = \int_0^\infty f_\alpha(t,v)\, f_\alpha(t,v')\, a(v,v')\,\mathrm{d}v' \tag{D.10}$$

$$B_B(t,v) = \int_{v'}^\infty \varsigma(v')\, P(v|v')\, b(v')\, f_\alpha(t,v')\,\mathrm{d}v' \tag{D.11}$$

$$D_B(t,v) = b(v)\, f_\alpha(t,v) \tag{D.12}$$

Este modelo simplicado foi resolvido utilizando o método das classes (RAMKRISHNA [52]) para a discretização das integrais e integrando o sistema de equações diferenciais ordinárias resultantes com a DASSL (PETZOLD [54]).