MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA

PROGRAMAÇÃO PARALELA E SEQUENCIAL APLICADA À OTIMIZAÇÃO DE ESTRUTURAS METÁLICAS COM O ALGORITMO PSO

por

Adelano Esposito

Dissertação para obtenção do Título de
Mestre em Engenharia

Porto Alegre, dezembro de 2012

PROGRAMAÇÃO PARALELA E SEQUENCIAL APLICADA À OTIMIZAÇÃO DE ESTRUTURAS COM O ALGORITMO PSO

por

Adelano Esposito
Engenheiro Mecânico

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Mecânica, da Escola de Engenharia da Universidade Federal do Rio Grande do Sul, como parte dos requisitos necessários para a obtenção do Título de

Mestre em Engenharia

Área de Concentração: Mecânica dos Sólidos

Orientador: Profa. Dra. Letícia Fleck Fadel Miguel
Coorientador: Prof. Dr. Herbert Martins Gomes

Comissão de Avaliação:

Prof. Dr. Moacir Kripka, PPGEng/UPF

Prof. Dr. João Ricardo Masuero, DECIV/UFRGS

Prof. Dr. Ignacio Iturrioz, PROMEC/UFRGS

Prof. Ph.D. Francis Henrique Ramos França
Coordenador do PROMEC

Porto Alegre, 07 de dezembro de 2012.

AGRADECIMENTOS

Manifesto meus sinceros agradecimentos às seguintes pessoas e instituições:

- Aos Professores Letícia Fadel Miguel e Herbert Martins Gomes, pela amizade e valiosa orientação que tornou possível a conclusão deste trabalho. Eu me considero privilegiado por ter estudado e trabalhado com estes grandes pesquisadores e Doutores;
- A Universidade Federal do Rio Grande do Sul, através do Grupo de Mecânica Aplicada – GMAP, pela oportunidade de capacitação;
- A CAPES, pelo auxílio;
- A minha namorada Daniela Fátima Giarollo, pelo amor, carinho, incentivo, paciência e tolerância, que mesmo distante me permitiu encontrar paz e tranquilidade, condições fundamentais para execução deste trabalho;
- Aos meus familiares pelo incentivo e carinho, em especial ao meu pai Luiz Esposito, minha mãe Lourdes Esposito, meus irmãos Arlei Esposito e Claudio A. Esposito;
- A Deus, pela oportunidade de estar aqui e pela ajuda que sempre tive ao longo desta vida;
- Aos colegas e todas as pessoas que, direta ou indiretamente, contribuíram para a realização deste trabalho.

RESUMO

Um dos métodos heurísticos bastante explorados em engenharia é o PSO (Otimização por enxame de partículas). O PSO é uma meta-heurística baseada em populações de indivíduos, na qual candidatos à solução evoluem através da simulação de um modelo simplificado de adaptação social. Este método vem conquistando grande popularidade, no entanto, o elevado número de avaliações da função objetivo limita a sua aplicação em problemas de grande porte de engenharia. Por outro lado, esse algoritmo pode ser facilmente paralelizado, o que torna a computação paralela uma alternativa atraente para sua utilização. Neste trabalho, são desenvolvidas duas versões seriais do algoritmo por enxame de partícula e suas respectivas extensões paralelas. Os algoritmos paralelos, por meio de funções disponíveis na biblioteca do *MATLAB*®, utilizam os paradigmas mestre-escravo e múltiplas populações, diferindo entre si pela forma de atualização das partículas do enxame (revoada ou pseudo-revoada) bem como pelo modo de comunicação entre os processadores (síncrono ou assíncrono). Os modelos propostos foram aplicados na otimização de problemas clássicos da engenharia estrutural, tradicionalmente encontrados na literatura (*benchmarks*) e seus resultados são comparados quanto às métricas utilizadas na literatura para avaliação dos algoritmos. Os resultados obtidos demonstram que a computação paralela possibilitou uma melhora no desempenho do algoritmo sequencial assíncrono. Também são registrados bons ganhos de tempo de processamento para as duas extensões paralelas do algoritmo, salvo que o algoritmo paralelo síncrono, diferentemente da versão paralela assíncrona, demonstrou um crescente desempenho computacional à medida que mais processadores são utilizados.

Palavras-chave: meta-heurística; sequencial; paralelização; otimização estrutural.

ABSTRACT

Amongst heuristic algorithms, PSO (Particle Swarm Optimization) is one of the most explored. PSO is a metaheuristic based on a population of individuals, in which solution candidates evolve by simulating a simplified model of social adaptation. This method has becoming popular, however, the large number of evaluations of the objective function limits its application to large-scale engineering problems. On the other hand, this algorithm can easily be parallelized, which makes parallel computation an attractive alternative to be used. In this work, two versions of the serial particle swarm algorithm and their parallel extensions are developed. The parallel algorithms, by means of available MATLAB® functionalities, use the master-slave paradigm and multiple populations, differing from each other by the way the particle swarm is updated (flocking or pseudo-flocking) as well as by the communication between processors (synchronous or asynchronous). The proposed models were applied to the optimization of classical structural engineering problems found in the literature (benchmarks) and the results are compared in terms usual metrics used for algorithm evaluation. The results show that parallel computing has enabled an improvement in the performance of asynchronous parallel algorithm. Good time savings were recorded for the two parallel extensions, except that the synchronous parallel algorithm, unlike the asynchronous parallel version, demonstrated a growing performance as more processors are used.

Keywords: metaheuristic, sequential, parallelization, structural optimization.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS E ABREVIATURAS

Caracteres Latinos

A_j	Área da seção transversal da barra *j*	[m^2]
A_j^{inf}	Área mínima da seção transversal da barra *j*	[m^2]
A_j^{Sup}	Área máxima da seção transversal da barra *j*	[m^2]
c_1	Constante cognitiva da partícula i	[Adimensional]
c_2	Constante social do enxame	[Adimensional]
c	Coeficiente de penalização	[Unitário]
C_c	Índice de esbeltez que divide a flambagem elástica em inelástica	[Adimensional]
COV	Coeficiente de variação	[Adimensional]
E	Módulo de Elasticidade do material	[N/m^2]
$Eficiência$	Eficiência	[Adimensional]
FP	Fator de Penalização	[Adimensional]
$f_{máx}(x^*)$	Valor ótimo máximo da função objetivo	[Unitário]
$f_{média}(x^*)$	Valor ótimo médio da função objetivo	[Unitário]
$f_{mín}(x^*)$	Valor ótimo mínimo da função objetivo	[Unitário]
$f_p(x, c)$	Função penalizada	[Unitário]
Fy	Tensão de escoamento do material	[N/m^2]
FP_{pr+1}	Fator de penalização futuro	[Adimensional]
FP_{pr}	Fator de penalização atual	[Adimensional]
$f(x_i^k)$	Função objetivo da partícula *i* na iteração *k*	[Unitário]
$gbest$	Melhor valor da função objetivo encontrado pelo enxame	[Unitário]
g_j	Restrição de desigualdade da barra *j*	[Unidade]
h_k	Restrição de igualdade	[Unidade]
K	Constante determinada pela geometria transversal	[Adimensional]
k	Número de iterações	[Unitário]
k_{max}	Número máximo de iterações	[Unitário]
$lbest_i$	Melhor valor da função objetivo encontrado pela partícula *i*	[Unitário]
L_j	Comprimento da barra *j*	[m]

m	Número total de variáveis de projeto	[Adimensional]
M	Número total de barras	[Unidade]
n	Número total de partículas	[Adimensional]
n_A	Número total de variáveis de projeto	[Unidade]
n_{coord}	Número total de coordenadas nodais	[Adimensional]
n_g	Número de restrições de desigualdade	[Unidade]
n_h	Número de restrições de igualdade	[Unidade]
N_p	Número total de processadores para processamento paralelo	[Adimensional]
pr_j	Valor associado à violação da restrição da barra *j*	[Adimensional]
r_1	Parcela de aleatoriedade	[Unitário]
r_2	Parcela de aleatoriedade	[Unitário]
R	Reinicialização do algoritmo	[Adimensional]
r_j	Raio de giração da seção transversal da barra *j*	[Unitário]
$Speedup$	Ganho efetivo com o processamento paralelo	[Adimensional]
T_{exec}	Tempo total de execução do algoritmo	[s]
T_{final}	Tempo em que o último processador finalizou sua tarefa	[s]
$T_{inicial}$	Tempo inicial registrado pelo primeiro processador	[s]
$tol_{\ cov}$	Tolerância para a covariação	[Adimensional]
tol_{FP}	Tolerância para o fator de penalização	[Adimensional]
T_{par}	Tempo de execução do algoritmo paralelo	[s]
T_{ser}	Tempo de execução do algoritmo serial	[s]
$v_{i,j}(t)$	Velocidade atual da partícula i da variável de projeto *j*	[m/s]
$v_{i,j}(t+1)$	Velocidade futura da partícula i da variável de projeto *j*	[m/s]
$V_{máx}$	Velocidade máxima associada à partícula	[Adimensional]
$V_{mín}$	Velocidade mínima associada à partícula	[Adimensional]
w	Momento ou inércia	[Adimensional]
$w_{máx}$	Momento ou inércia máximo	[Adimensional]
$w_{mín}$	Momento ou inércia mínimo	[Adimensional]
$\vec{x}^*$	Vetor de coordenadas da solução ótima conhecida	[m]
$x_{i,j}(t)$	Posição atual da partícula i da variável de projeto *j*	[m]
$x_{i,j}(t+1)$	Posição futura da partícula i da variável de projeto *j*	[m]

$xgbest_j(t)$	Melhor posição do enxame da variável de projeto *j*	[m]
x_j^{inf}	Limite inferior para a variável de projeto *j*	[m]
x_q^{inf}	Limite inferior para a coordenada do nó *q*	[m]
$xlbest_{i,j}(t)$	Melhor posição da partícula *i* da variável de projeto *j*	[m]
x_q	Coordenada do nó *q*	[m]

Caracteres Gregos

λ_j	Índice de esbeltez da barra *j*	[Adimensional]
μ	Acurácia do algoritmo	[Adimensional]
μ_{xgbest}	Média das melhores posições do enxame	[m]
ω_j	Frequência natural da barra *j*	[rad/s]
$\omega_j^{máx}$	Valor limite para a frequência natural da barra *j*	[rad/s]
ρ	Densidade do material	[kg/m^3]
$\boldsymbol{\sigma}$	Robustez do algoritmo	[Adimensional]
σ_j	Tensão atuante na barra *j*	[N/m^2]
σ_j^F	Tensão admissível quanto à flambagem da barra *j*	[N/m^2]
σ_j^T	Tensão admissível quanto à tração da barra *j*	[N/m^2]
χ	Fator de constrição associado à partícula	[Adimensional]

1 INTRODUÇÃO

Cada vez mais cresce o interesse pela otimização numérica em diversos campos da engenharia. Pode-se dar como exemplo projeto de equipamentos, controle de operações, controle de processos, estimação de variáveis em modelos estruturais, onde frequentemente são desenvolvidos algoritmos com características globais somadas à capacidade da solução de problemas de grande porte e dimensão. Neste contexto, ha muito vem se estudando os métodos heurísticos aplicados à resolução de problemas de elevado nível de complexidade computacional. Dentre os diversos algoritmos de busca global existentes, o algoritmo do enxame de partículas (PSO) esta se destacando entre as demais heurísticas por sua reconhecida robustez, eficiência e simplicidade de desenvolvimento. Uma deficiência desse método é quanto à necessidade de um elevado número de avaliações da função objetivo durante o processo de busca. Essa característica se torna crítica quando o PSO é exposto à solução de problemas de grande porte, nos quais o cálculo da função objetivo demanda elevado esforço computacional.

A necessidade de se resolver problemas complexos em tempo hábil, utilizando o algoritmo por enxame de partículas, tem motivado estudos para a criação de suas extensões paralelas. Além das vantagens supracitadas do PSO em comparação com as demais heurísticas, a característica do método de realizar o cálculo da função objetivo de maneira independente para cada partícula do enxame torna-o facilmente paralelizável, possibilitando o desenvolvimento das versões paralelas como um campo aberto para pesquisas com o algoritmo PSO.

No presente trabalho, são desenvolvidas duas versões seriais do algoritmo por enxame de partícula e suas respectivas extensões paralelas. Quanto às versões seriais implementadas neste trabalho, SSPSO (Otimização por Enxame de Partículas Sequencial Síncrono) e SAPSO (Otimização por Enxame de Partículas Sequencial Assíncrono), a distinção entre elas consiste no modo como a atualização das partículas é realizado, que pode ser de forma dinâmica a cada pseudo-revoada (SAPSO) ou em uma única etapa a cada revoada do enxame (SSPSO). As respectivas extensões paralelas (PSPSO e PAPSO) implementadas mantém a natureza das suas versões seriais.

Os algoritmos são implementados em funções padrão, conhecidamente utilizadas na literatura, e problemas de otimização de estruturas metálicas, os quais são comumente

analisados por diversos pesquisadores. Casos deste tipo, também chamados de *benchmark*, são de grande importância para a avaliação dos algoritmos, pois possibilitam testar a capacidade do algoritmo em atingir a solução ótima conhecida. Além disso, com o *benchmark*, é possível avaliar as métricas que determinam a disposição das partículas no espaço de busca (acurácia, robustez e desempenho do algoritmo) e o desempenho computacional (tempo de execução, *Speedup* e eficiência) apresentado pelos algoritmos para chegar a uma solução próxima ou melhor que a obtida pelo *benchmark*.

Os códigos são desenvolvidos em ambiente *MATLAB*® e as extensões paralelas são submetidas ao processamento em multiprocessadores de até oito núcleos. Com essa plataforma computacional, é possível avaliar o desempenho dos algoritmos propostos na otimização de estruturas metálicas investigando as vantagens e dificuldades apresentada por cada modelo. Também se procura demonstrar os ganhos decorrentes do paralelismo em termos de velocidade de processamento bem como nos resultados da otimização.

1.1 JUSTIFICATIVA

Embora os processadores modernos apresentem alto desempenho computacional, muitos problemas de engenharia permanecem computacionalmente custosos para alguns métodos heurísticos de otimização. Buscando reduzir tais efeitos indesejáveis destas heurísticas, várias formas de hibridização foram propostas. Contudo, apesar dos constantes avanços das pesquisas em melhorar a velocidade de convergência dos métodos heurísticos, as modernas e potentes arquiteturas multiprocessadas permanecem pouco exploradas. O uso ineficiente dos multiprocessadores disponíveis justifica os altos tempos gastos com a execução dos métodos heurísticos, tornando-os inviáveis para determinadas aplicações. A característica supracitada denuncia o grande interesse da atual pesquisa, o qual consiste em tornar estas heurísticas computacionalmente atraentes mesmo quanto aplicadas em problemas de elevada complexidade. Dessa forma, devido a sua reconhecida robustez, escolheu-se o algoritmo de otimização por enxame de partículas – PSO, para ser implementado na forma paralela como uma alternativa para a utilização mais eficiente do método. Duas versões seriais do algoritmo PSO são desenvolvidas para servirem de comparação a suas extensão paralelas e assim demonstrarem seus ganhos em termos de velocidade de convergência do

algoritmo bem como quanto ao desempenho computacional pelo aproveitamento dos processadores.

1.2 OBJETIVOS

Este trabalho tem com objetivo principal implementar a forma serial e paralela do algoritmo de otimização por enxame de partículas síncrono e assíncrono, submetendo-os a otimização de problemas clássicos da mecânica estrutural para avaliar os ganhos obtidos com os algoritmos desenvolvidos em termos da solução ótima e desempenho. Para tanto são utilizadas função presentes no próprio *MATLAB*® (*PARFOR* e *SPMD*) para execução das etapas em paralelo dos códigos desenvolvidos.

Sendo assim, pode-se especificar os objetivos específicos como sendo:

a) Compreender as etapas que compõem a solução de um problema de otimização estrutural;
b) Estudar o método de otimização por enxame de partícula para a solução de problemas de otimização estrutural;
c) Desenvolver e implementar duas formas de atualização da posição e velocidade da partícula;
d) Estudar a técnica de processamento paralelo voltado à aplicação em algoritmos por enxame de partículas;
e) Apresentar as respectivas extensões paralelas aos algoritmos seriais desenvolvidos;
f) Submeter os algoritmos desenvolvidos a exaustivos experimentos em casos de *benchmark*;
g) Avaliar os algoritmos empregados neste trabalho segundo métricas de avaliação comumente utilizadas na literatura.

1.3 ORGANIZAÇÃO DO TRABALHO

O conteúdo da presente dissertação está dividido em sete capítulos:

No Capítulo 1, apresenta-se uma introdução ao tema abordado, expondo a motivação do autor para realizar a pesquisa e como o trabalho é organizado;

No Capítulo 2, é realizada uma revisão bibliográfica referente ao assunto, discutindo: o problema da otimização estrutural, o algoritmo de otimização por enxame de partículas e suas variações, a programação paralela disponível no ambiente *MATLAB*®, os modelos de algoritmos por enxame de partículas paralelos existentes, suas topologias de comunicação e variações;

No Capítulo 3, desenvolvem-se os modelos de algoritmos propostos, descrevendo todas as etapas dos códigos implementados com auxilio de pseudocódigos e fluxogramas;

No Capítulo 4, é apresentada uma metodologia para avaliação dos algoritmos desenvolvidos, que tradicionalmente é utilizada por diversos autores;

No Capítulo 5, é realizada a validação dos algoritmos expondo-os a otimização de funções padrão da literatura;

No Capítulo 6, os algoritmos são aplicados à otimização de estruturas metálicas, descrevendo-se a análise dos resultados;

No Capítulo 7, são apresentadas as conclusões e as sugestões para trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

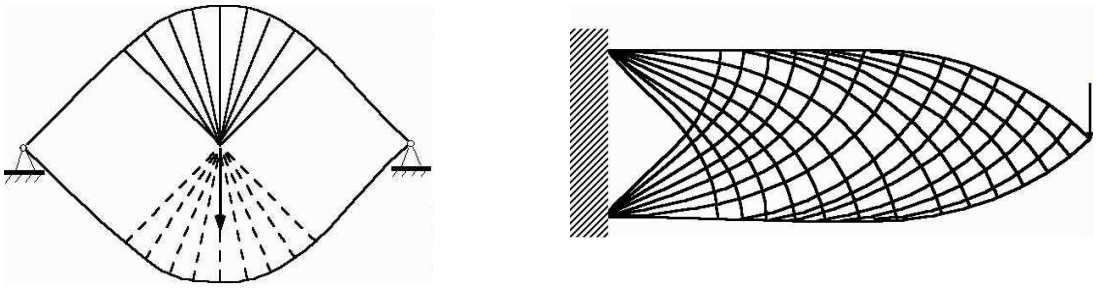
Neste capítulo é realizada uma revisão de literatura com base em pesquisas já desenvolvidas, explorando o tema em nível de aprendizado para auxiliar no desenvolvimento do presente trabalho.

2.1 OTIMIZAÇÃO ESTRUTURAL

O conceito de otimização estrutural foi introduzido pela primeira vez por Maxwell em 1872 buscando obter projetos de estruturas civis, principalmente pontes, que utilizassem a menor quantidade de material e não falhassem. Este método consistia em usar conceitos da teoria de elasticidade da seguinte forma:

a) Dado um carregamento atuando num domínio infinito cujos pontos onde este domínio estivesse apoiado (pontos de apoio da ponte), calcular o campo de tensões principais mecânicas usando teoria da elasticidade;
b) As direções das tensões principais correspondem às direções onde não ocorrem tensões de cisalhamento, apenas tensões normais;
c) Uma vez obtida essas direções, Maxwell sugeriu que a estrutura ótima, que utilizasse menos material, seria constituída de elementos de treliça alinhados com essas direções principais.

A ideia de Maxwell foi retomada por Michell em 1904 aplicando o método a projeto de vários tipos de estruturas a fim de obter o menor volume de material. A Figura 2.1 ilustra alguns dos resultados obtidos por Michell. A Figura 2.1(a) ilustra a estrutura ótima considerando-se dois apoios, um em cada extremidade e um carregamento central. As linhas sólidas representam os elementos de treliças em tração e as linhas tracejadas representam os elementos de treliça em compressão. A Figura 2.1(b) ilustra o resultado ótimo para uma estrutura engastada com uma carga aplicada na extremidade [Rozvany, 1995].



(a) (b)

Figura 2.1 – (a) Estrutura bi apoiada com carga central, (b) Estrutura engastada com carga na extremidade. Fonte: Michell, 1904.

No entanto, de 1904 até a década de 60, somente problemas acadêmicos de estruturas simples foram estudados. Apenas nos anos 60, com a disposição dos computadores e do método de elementos finitos (MEF), que problemas práticos de otimização estrutural passaram a ganhar mais atenção. Na década de 70, vários algoritmos para problemas de otimização não-lineares são desenvolvidos e implementados.

2.2 DEFINIÇÕES DO PROBLEMA DE OTIMIZAÇÃO

Um problema onde se procura pelo máximo ou mínimo de uma função de diversas variáveis, devendo estas satisfazer alguns requisitos impostos, é chamado problema de programação matemática ou otimização. A função recebe o nome de função objetivo, e os requisitos impostos o de restrições do problema [Fritzsche, 1978].

Trazendo o conceito citado no parágrafo anterior para problemas práticos de otimização estrutural, a função objetivo é relacionada com o parâmetro que queremos minimizar ou maximizar. No caso de uma estrutura a função objetivo pode ser, por exemplo, a rigidez, a frequência de ressonância, o volume, etc.. Na busca pelo ótimo da função objetivo é necessário que alguns limites sejam respeitados, como por exemplo, ao diminuir a área da seção transversal de uma barra, a tensão máxima suportada pelo material não pode ser excedida. Essa limitação é conhecida como restrição de projeto.

As restrições impostas à solução da otimização podem ser simples, como limites superiores e inferiores para os valores admissíveis das variáveis de projeto (restrições laterais ou primárias), bem como mais complexas, relacionadas aos critérios de falhas do material, como tensão e flexibilidade (restrições secundárias). A essa última categoria, também se enquadram as restrições de comportamento, as quais limitam o deslocamento de um

determinado ponto da estrutura a um valor específico. Sendo assim, de maneira geral, as restrições impõem uma solução de compromisso às variáveis de projeto na melhora da função objetivo.

As variáveis de projeto são os parâmetros que podem ser alterados na otimização. Podem representar certa dimensão que será alterada, como área da seção transversal de uma viga, espessura de uma chapa, orientação das fibras de um material compósito, etc.. Elas são classificadas em contínuas e discretas. Quando se utilizam variáveis contínuas, por exemplo, para minimizar a massa de uma estrutura, a área da seção transversal de um perfil pode assumir qualquer valor real dentro do intervalo $[A_j^{inf},\ A_j^{Sup}]$. Considerando para o mesmo exemplo a utilização de variáveis discretas, estas podem assumir apenas valores compreendidos dentro de um determinado conjunto fixo. Na prática isto significa que a área da seção transversal do perfil deve ser selecionada de acordo com a disponibilidade comercial.

De uma forma geral, um problema de otimização com restrições é apresentado da seguinte forma [Haftka e Gürdal, 1991; Arora, 2004].

Minimizar ou Maximizar: *Função Objetivo:* $f(x)$

Variáveis de Projeto: $x = \{x_1, x_2, \dots x_m\}$

Sujeito a: *Restrições:* $g_j(x) \leq 0, \quad j = 1, \dots, n_g$

$$h_k(x) = 0, \quad k = 1, \dots, n_h$$

$$A_j^{inf} \leq A_j \leq A_j^{Sup}, \quad i = 1, \dots, n_A$$

Onde x é o vetor das variáveis de projeto, g_j e h_k são respectivamente, funções que contém restrições de desigualdade (tensão em cada ponto da estrutura deve ser, por exemplo, inferior a um limite admissível) e igualdade (equação de equilíbrio). Os valores assumidos pelas variáveis de projeto A_j devem estar dentro de um intervalo limitado pelas restrições laterais A_j^{inf} e A_j^{Sup}. Os números de restrições de desigualdade e igualdade são representados por n_g e n_h respectivamente.

Basicamente existem três abordagens que tratam dos problemas de otimização estrutural. São elas: otimização paramétrica, otimização de forma e otimização topológica. A

Figura 2.2 ilustra esses conceitos aplicados ao projeto de treliças, o qual é de interesse do presente trabalho.

Num problema de otimização paramétrica, as variáveis de projeto descrevem características geométricas da estrutura, no caso de treliças, procura-se uma distribuição ótima da área da seção transversal de cada barra, de modo que minimize (ou maximize) uma grandeza física, tal como flexibilidade ou tensão, enquanto as restrições são satisfeitas.

Na otimização de forma, busca-se a forma ótima para o modelo de estrutura. As variáveis de projeto, em treliças, são geralmente as posições dos nós resultando na distorção da malha, Por este motivo, alguns autores destacam que esse método pode muitas vezes gerar resultados inválidos devido a problemas de convergência da solução de elementos finitos.

A otimização topológica visa uma distribuição ótima de material na estrutura. Nesta técnica é retirado/acrescentado material em diferentes regiões ao longo da estrutura formando assim uma nova topologia para a treliça.

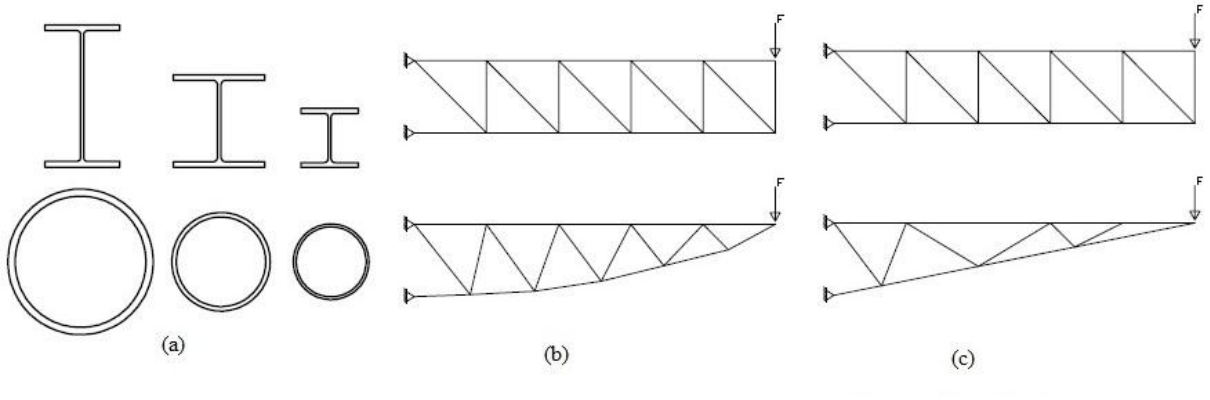


Figura 2.2 – Tipos de otimização estrutural: (a) paramétrica, (b) forma, (c) topológica

Definido o problema de otimização, é necessário determinar o espaço de localização de sua solução ótima, que é o espaço de busca. Essa região compreende as possíveis soluções sobre as variáveis de projeto do problema a ser otimizado, sendo delimitada pelas funções de restrição, ou seja, as restrições são responsáveis por dividir o espaço de busca em domínio viável e domínio inviável. No domínio viável, a busca pela solução ótima parte de uma solução previamente conhecida; já no domínio inviável, não se tem uma estimativa inicial de solução ótima.

A solução ótima encontrada pode ser de natureza global ou local. A solução da otimização é dita global quando o ponto obtido é o melhor dentro da totalidade do espaço de busca, ou seja, a melhor solução dentre todas as soluções existentes para aquele determinado

problema, diferentemente da solução ótima local, a qual define o melhor ponto (solução) dentro de um subespaço específico.

Na seção 3.3 é apresentada uma definição mais detalhada para o problema de otimização no contexto do presente trabalho.

2.3 ALGORITMOS PARA OTIMIZAÇÃO ESTRUTURAL

A solução de um problema de otimização estrutural consiste na utilização de métodos de programação matemática, definidos de acordo com as características da função objetivo e das restrições que caracterizam o problema em questão.

Uma vez definida a função objetivo, o próximo passo consiste na sua minimização ou maximização através do ajuste de parâmetros, representados no presente trabalho pela área da seção transversal dos perfis.

Um dos métodos tradicionais de ajuste de parâmetros são os métodos determinísticos, que a partir de uma estimativa inicial de parâmetros, buscam o mínimo da função objetivo. São métodos baseados em gradientes e necessitam que as funções sejam continuas e diferenciáveis. Dentre suas qualidades é possível citar sua rápida convergência, desde que se tenha informação a respeito do comportamento da função a ser otimizada em termos analíticos e que a mesma atenda a critérios de convexidade e de suavidade.

No entanto, é comum em problemas reais de engenharia a não linearidade de seus modelos matemáticos. Por esta razão, a minimização da função objetivo pode apresentar uma série de dificuldades, dentre as quais é possível destacar: a presença de mínimos locais, a alta correlação entre os parâmetros, incerteza dos valores das variáveis e comportamento do modelo utilizado para representar o problema, falta de convexidade por limitações das funções de restrições, o desconhecimento de uma boa estimativa inicial dos parâmetros e a falta de suavidade da função a ser otimizada pela presença de variáveis de projeto não contínuas. Este conjunto de características comumente encontrado em problemas reais justifica a dificuldade do método em realizar a busca por um ótimo global.

Perante as dificuldades associadas aos métodos tradicionais determinísticos, aplicam-se as estratégicas heurísticas para contornar tais problemas. “Heurística é uma estratégia de solução por tentativa e erro que produz soluções aceitáveis a problemas complexos em um tempo razoável e prático. A complexidade do problema de interesse o torna impossível de

uma procura do ótimo em cada solução possível ou combinação delas, de forma que o objetivo é encontrar uma solução boa e viável numa escala de tempo aceitável [Yang, 2010]".

Nos métodos heurísticos, o procedimento de busca aleatória somado ao grande número de avaliações da função objetivo contribuem para que a busca não fique presa em um ótimo local, aumentando a probabilidade de encontrar o ótimo global. Além disso, estes métodos não dependem de uma boa estimativa inicial dos parâmetros e não utilizam gradientes durante a otimização. Estas características heurísticas permitem a estes métodos trabalharem tanto com funções contínuas como discretas.

Entretanto, tem como limitação a necessidade de um grande número de avaliações da função objetivo, de modo que, estes métodos demandam um alto tempo de processamento [Schwaab, 2005].

Uma forma de tentar melhorar a eficiência dos algoritmos heurísticos consiste em identificar as regiões mais prováveis de conter o ótimo global e concentrar a busca nestas regiões. Esta forma é conhecida como Busca Adaptativa Pura [Patel, 1988, Zabinsky e Smith, 1992].

Outra alternativa interessante, que vem despertando o interesse dos pesquisadores em função de seu bom desempenho, consiste na utilização em conjunto dos métodos determinísticos e heurísticos em uma mesma estrutura algorítmica. Esses métodos são híbridos. Algumas destas técnicas utilizam, por exemplo, um método heurístico para gerar uma solução e refinar esta solução com um método de busca local. Hibbert, 1993, avaliou a utilização do heurístico AG (Algoritmo Genético) seguido de um método determinístico. Price, 1976, propôs um algoritmo híbrido que combina a heurística Busca Aleatória Pura com o método Simplex; este novo método foi chamado de Busca Aleatória Controlada. Kvasnicka e Pospíchal, 1997, propuseram uma técnica híbrida utilizando o Recozimento Simulado para tornar aleatória uma busca realizada com base no método Simplex. Parsopoulos e Vrahatis, 2002, fizeram a inicialização do algoritmo de Otimização por Enxame de Partículas (PSO), que geralmente é feita de forma aleatória, utilizando o método Simplex, procurando detectar mais rapidamente regiões promissoras e acelerar a busca pelo mínimo global. Esmin, 2005, apresentou um novo algoritmo híbrido, chamado de algoritmo de Otimização por Enxame de Partícula Híbrido com Mutação (HPSOM), que combina a tradicional velocidade e regras de atualização de posição do algoritmo PSO com o conceito da mutação numérica dos AGs.

Recentemente, a heurística que vem se destacando como uma boa alternativa para solução de problemas complexos na engenharia é o algoritmo de Otimização por Enxame de Partículas (PSO).

O algoritmo PSO esta sendo amplamente estudado na literatura devido a algumas características particulares que o tornam mais popular que outros métodos, sendo elas: insensível à mudança de escala de variáveis; fácil implementação; adaptável à computação paralela; não requer cálculo de derivadas; poucos parâmetros heurísticos para serem definidos pelo usuário; entre outras.

Na seção subsequente, é realizada uma descrição do método de Otimização por Enxame de Partículas.

2.4 ALGORITMO DE OTIMIZAÇÃO POR ENXAME DE PARTÍCULAS

Nesta seção é realizada uma revisão sobre o algoritmo de enxame de partículas, expondo o método clássico e algumas de suas variantes.

2.4.1 PSO Padrão

O algoritmo padrão de otimização por enxame de partículas (*Particles Swarm Optimization*) foi inicialmente proposto por James Kennedy e Russel Eberhart, 1995, como sendo uma técnica inspirada no comportamento social de bandos de pássaros. A busca por alimento e a interação entre os pássaros ao longo do voo são modeladas como um mecanismo de otimização.

Fazendo uma analogia, o termo partícula foi adotado para simbolizar os pássaros e representar as possíveis soluções do problema a ser resolvido. A topologia ou região percorrida pelos pássaros é equivalente ao espaço de busca e encontrar o local com comida corresponde a encontrar a solução ótima (Figura 2.3). Em problemas de otimização, o espaço de busca é dado pela função objetivo do problema. É a função que se quer otimizar, nela estão presentes as características do problema as quais o PSO necessita para atingir seu objetivo.

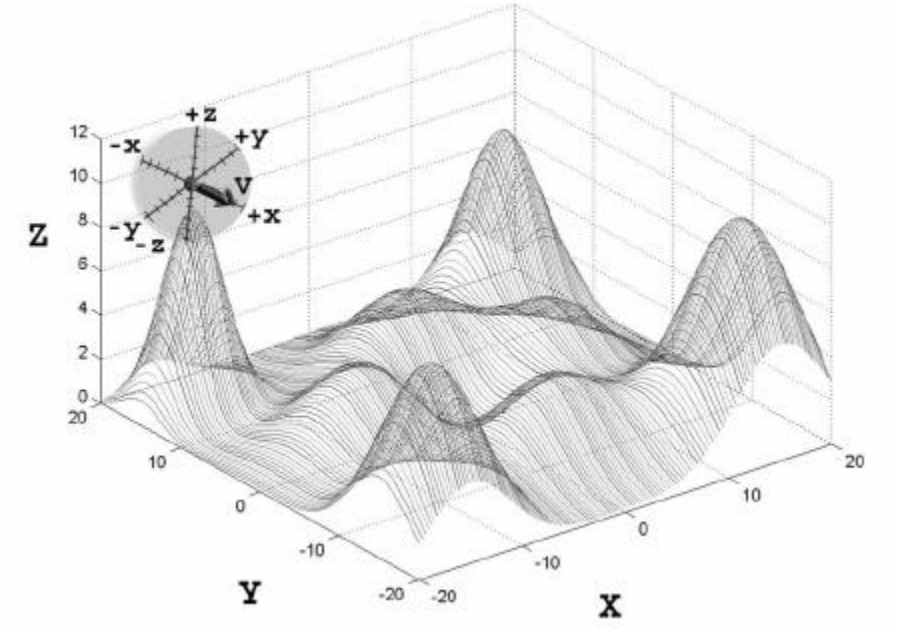


Figura 2.3 – Partícula observando o espaço de busca (função objetivo)
Fonte: Kaewkamnerdpong e Bentley, 2005.

Para que as partículas sempre se aproximem do ponto ótimo, ao invés de se perderem e nunca alcançá-lo enquanto estiverem percorrendo a função objetivo, utiliza-se a função de aptidão *fitness*, responsável por avaliar o desempenho de cada partícula e informar com respeito às outras partículas quão boa é a sua posição. O termo que indica a experiência ou conhecimento individual de cada partícula, representada pela melhor solução encontrada individualmente e gravada para uso posterior é o *lbest*. O responsável por representar o conhecimento do enxame como um todo e a melhor solução encontrada até então por algum dos indivíduos que formam o enxame é representado na literatura por *gbest*.

Dois parâmetros guiam a direção de procura que cada partícula apresentada ao longo do processo iterativo: a solução atual da partícula e a sua velocidade. Na inicialização do algoritmo PSO a posição e velocidade de cada partícula são especificadas de forma aleatória (randômica), como demonstrado na Figura 2.4.

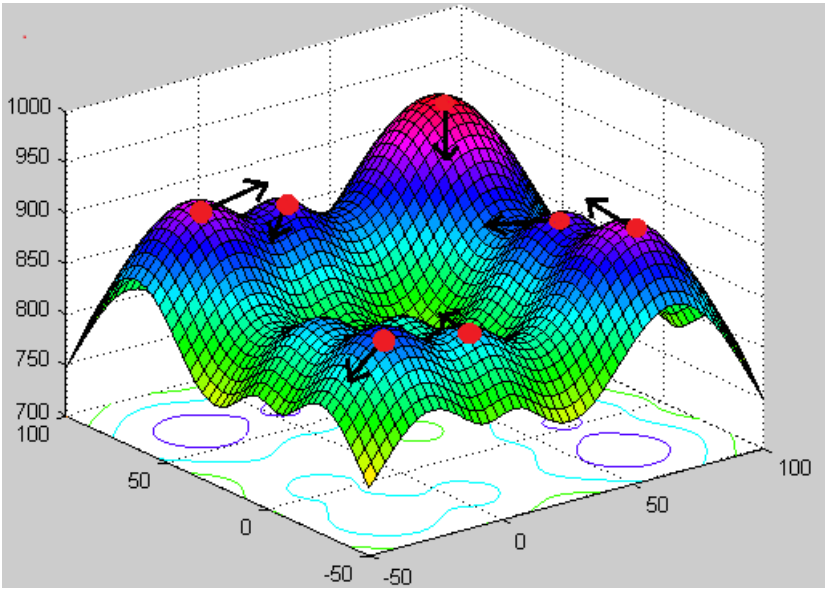


Figura 2.4 – Distribuição das partículas (enxame) no espaço de busca: pontos vermelhos são a posição e setas são as velocidades

Cada partícula do enxame representa uma possível solução para o problema de otimização, que pode ser representada por um objeto que executa uma ação influenciada por três "forças" representadas matematicamente em forma de vetor: inércia, memória e cooperação. A inércia representa a força que impulsiona a partícula para seguir em direção ao ótimo pretendido, a memória faz com que a partícula se direcione de acordo com suas experiências individuais passadas e a cooperação direciona a partícula para a melhor direção já conhecida pelo enxame [Miranda, 2005].

A atualização da posição de uma partícula em um espaço de busca de forma simplificada (não considerando o vetor inércia) corresponde ao ilustrado na Figura 2.5. Cada partícula modifica sua velocidade ($v_{i,j}(t+1)$) levando em conta a sua distância atual para a melhor posição já encontrada pela mesma partícula $\left(xlbest_{i,j}(t) - x_{i,j}(t)\right)$ e a distância da sua posição atual à melhor posição encontrada pelo grupo $\left(xgbest_j(t) - x_{i,j}(t)\right)$. Esses vetores são ponderados por fatores $c_1 r_1$ e $c_2 r_2$ os quais serão descritos a seguir.

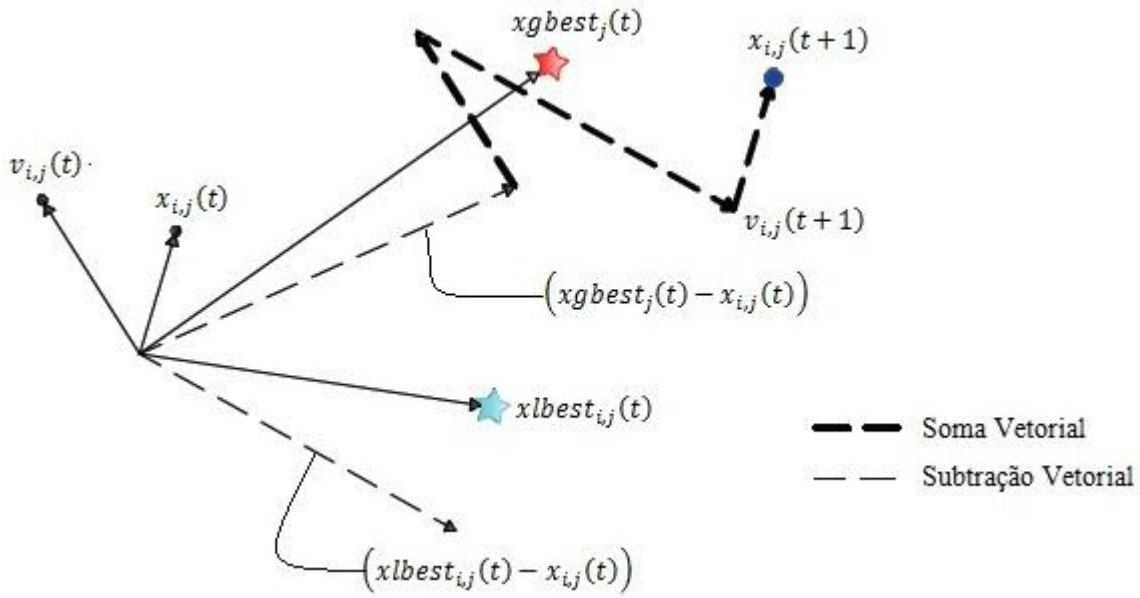


Figura 2.5 – Atualização da partícula

Pode se verificar com a Figura 2.5 que a nova posição da partícula $x_{i,j}(t+1)$ resulta da adição do vetor posição atual $x_{i,j}(t)$ com o novo vetor de velocidade $v_{i,j}(t+1)$. A nova posição das partículas é calculada pela Equação 2.2.

O Vetor velocidade é um dos itens mais importantes no algoritmo PSO. As partículas "voam" pelo espaço de busca tendo suas velocidades atualizadas dinamicamente de acordo com o histórico das experiências individuais e coletivas de todo o enxame [Shi, 2004]. Portanto, a evolução do PSO está associada à trajetória percorrida pelo enxame e ao tempo gasto para encontrar a melhor solução do problema. A atualização ao longo das iterações (*t*) do vetor velocidade é representada pela Equação 2.1.

$$v_{i,j}(t+1) = v_{i,j}(t) + c_1 r_1 \left(xlbest_{i,j}(t) - x_{i,j}(t)\right) + c_2 r_2 \left(xgbest_j(t) - x_{i,j}(t)\right) \quad (2.1)$$

$$x_{i,j}(t+1) = x_{i,j}(t) + v_{i,j}(t+1) \quad (2.2)$$

Onde:

- $v_{i,j}(t+1)$: é a velocidade atualizada da partícula *i* correspondente a variável de projeto *j*;
- $v_{i,j}(t)$: é a velocidade atual da partícula *i* correspondente a variável de projeto *j*;
- $x_{i,j}(t+1)$: é a posição atualizada da partícula *i* correspondente a variável de projeto *j*;
- $x_{i,j}(t)$: é a posição atual da partícula *i* correspondente a variável de projeto *j*;

- $xlbest_{i,j}(t)$: é a melhor posição já encontrada pela partícula i correspondente a variável de projeto j;
- $xgbest_j(t)$: é a melhor posição já encontrada pelo enxame de partículas correspondente a variável de projeto j;
- c_1: constante de aceleração cognitiva (individual), referente à $xlbest_{i,j}(t)$;
- c_2: constante de aceleração social (enxame), referente à $xgbest_j(t)$;
- r_1 e r_2: números aleatórios que podem estar entre zero e um.

2.4.2 Modificações do PSO

Após o surgimento do PSO, algumas melhorias foram propostas na tentativa de aumentar sua velocidade de convergência. Dentre as várias modificações sugeridas, algumas das mais relevantes para o presente estudo são apresentadas nos itens a seguir.

I. Limitação de Velocidade e Restrições Laterais

Como pode ser verificado nas Equações 2.1 e 2.2, não há um mecanismo que limite a velocidade da partícula, desta forma, altos valores de velocidade fazem com que a partícula tende a ultrapassar os limites do espaço de busca. Para evitar que isso possa ocorrer algumas estratégias podem ser adotadas. Uma delas consiste em "parar" a partícula nos limites do espaço de busca, ou refletir a partícula para dentro do espaço de busca (Figura 2.6) [Waintraub, 2009].

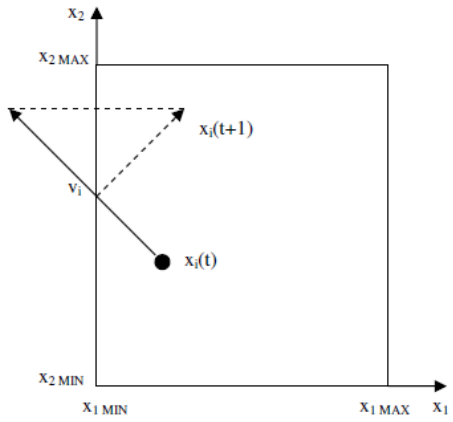


Figura 2.6 – Reflexão de uma partícula nas bordas de um espaço de busca

Fonte: Waintraub, 2009.

Entretanto, com esta última alternativa, a partícula tende a realizar várias reflexões dentro do espaço de busca, esse comportamento tende a baixar a eficiência do PSO, ou seja, o PSO localiza rapidamente a região do ótimo, mas uma vez dentro desta região, ele pode enfrentar dificuldades em ajustar sua velocidade para prosseguir numa busca mais refinada.

Para evitar o problema da reflexão da partícula, Eberhart e Kennedy, 1995, propuseram um mecanismo que limite a velocidade da partícula para um valor entre $V_{máx}$ e V_{min}. Caso a velocidade da partícula tente ultrapassar estes intervalos, haverá uma atualização para o valor limite mínimo ou máximo de velocidade.

II. Peso de Inércia (*w*)

A determinação do valor de $V_{máx}$ e $V_{mín}$ não é nada trivial, e a escolha errada para este valor pode implicar em uma perda de w. O peso de inércia indica o quanto da velocidade da iteração atual será mantido na velocidade da próxima iteração. Ao adicionar a inércia (*w*) na Equação 2.1 tem-se:

$$v_{i,j}(t+1) = wv_{i,j}(t) + c_1 r_1 \left(xlbest_{i,j}(t) - x_{i,j}(t)\right) + c_2 r_2 \left(xgbest_j(t) - x_{i,j}(t)\right) \qquad (2.3)$$

Sendo assim, a nova velocidade da partícula será dada em função dos vetores memória, cooperação e o vetor inércia associado à velocidade anterior. Na Figura 2.7, Hassan *et al.*, 2005, apresentam uma maneira simples para entender o procedimento de atualização da velocidade e posição da partícula.

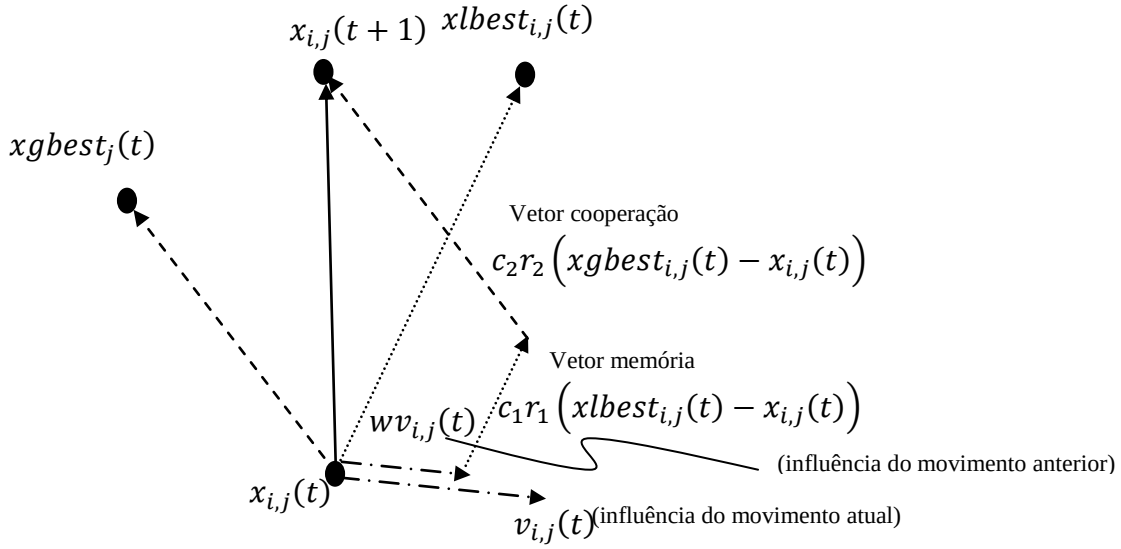


Figura 2.7 – Representação vetorial da atualização da velocidade e posição do PSO

Fonte: Adaptado de Gomes, 2011.

III. Fator de Constrição (χ)

Conceito similar ao peso de inércia pode ser dado ao o fator de constrição χ, proposto por Clerc e Kennedy, 2002. Segundo Berci, e Botura, 2008, esse fator possibilita a concepção de um algoritmo sem a necessidade da utilização do peso de inércia e o limite superior de velocidade. Entretanto, os autores salientam que apesar do fator resultar em uma convergência mais rápida do método, podem ocorrer perdas da capacidade exploratória, fazendo com que o método falhe quando iniciado muito distante do ponto ótimo. A Equação 2.4 é proposta por Bergh e Engelbrecht, 2006, para o calculo do parâmetro.

$$\chi = \frac{1.6}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|} \tag{2.4}$$

Onde, $\varphi = c_1 + c_2$

Clerc e Kennedy, 2002, demonstraram que atribuir valores para c_1e c_2 de forma a resultar em $\varphi < 4$, o enxame tende a convergir mais lentamente, ao passo que do $\varphi > 4$, a convergência torna-se mais rápida e garantida. Por esta razão, alguns autores assumem valores para c_1e c_2 iguais a 2,05 para assegurar convergência.

Ao aplicar o fator de constrição à Equação 2.3, a atualização do vetor velocidade proposta no presente trabalho é implementada no código de acordo com a Equação 2.5.

$$v_{i,j}(t+1) = \chi\left(wv_{i,j}(t) + c_1 r_1\left(xlbest_{i,j}(t) - x_{i,j}(t)\right) + c_2 r_2\left(xgbest_j(t) - x_{i,j}(t)\right)\right) \tag{2.5}$$

De acordo com Koh *et al*, 2006, mesmo que várias modificações no algoritmo PSO original tenham sido realizadas para aumentar a robustez e velocidade de processamento, uma questão muito importante que deve ser levado em consideração é quanto a abordagem síncrona ou assíncrona utilizada para atualizar as posições e velocidades das partículas. As duas seções subsequentes descrevem essas variações do PSO padrão.

2.5 ALGORITMO PSO SÍNCRONO

O algoritmo de otimização por enxame de partículas síncrono possui uma implementação bastante simples. O enxame é inicializado com as posições e velocidades das partículas atribuídas de maneira aleatória. Após, as partículas entram em um laço. Dentro deste, enquanto um critério de parada ou convergência não seja atingido, uma nova *revoada* (ou iteração) é reinicializada e a posição e velocidade de cada partícula são atualizadas de acordo com os melhores valores das partículas e do enxame, previamente calculados. Em outras palavras, pode-se dizer que, primeiramente todas as partículas do enxame devem ser avaliadas para só então, (não satisfeita a condição de parada) atualizarem suas posições e velocidades. Isto pode ser visto no pseudocódigo da Figura 2.8.

```
Inicialização do algoritmo
        Informar constantes do algoritmo
        Gerar posição e velocidade das partículas randomicamente
Inicialização da otimização
        Para k=1, número de iterações
                Para i=1, número de partículas
                        Avaliar função f(x_k^i)
                Fim
                Testar convergência
                Atualizar posição e velocidade
        Fim
```

Figura 2.8 – Pseudocódigo do algoritmo PSO síncrono

A descrição do algoritmo de otimização por enxame de partículas síncrono de forma mais detalhada, é apresentada na subseção 3.1.1, a qual detalha o algoritmo desenvolvido para o presente trabalho.

2.6 ALGORITMO PSO ASSÍNCRONO

Conforme verificado nos algoritmos síncronos, as partículas da nova *revoada* assumem posições influenciadas pela melhor partícula da *revoada* anterior. Este

comportamento impede que o enxame evolua dinamicamente. Além disso, esses algoritmos necessitam de um ponto de sincronização no final da *revoada*, tornando o código limitado à partícula mais lenta, ou no caso de processamento paralelo (seção 2.7), ao processador mais lento. Surge assim, uma estratégia assíncrona como alternativa para contornar esses problemas.

O algoritmo PSO assíncrono é muito semelhante ao algoritmo síncrono, exceto que a atualização de cada partícula é realizada logo após sua avaliação, ou conforme Koh *et al*., 2006, em algoritmos PSO assíncronos, a atualização da velocidade e posição da partícula ocorrem continuamente com base em dados atualmente disponíveis. Assim, uma partícula que antes (forma síncrona) era alterada apenas após a *revoada* do enxame, agora é alterada a cada *pseudo-revoada*.

Portanto, a *pseudo-revoada* possibilita ao algoritmo que ao encontrar um novo ótimo global (*gbest*), que ele seja imediatamente ajustado e disponibilizado (informado) às partículas ainda não atualizadas naquela iteração. Sendo assim, conforme Souza e Gomes, 2011, a implementação assíncrona é superior à implementação síncrona.

O pseudocódigo do algoritmo PSO assíncrono pode ser representado conforme a Figura 2.9.

```
Inicialização do algoritmo
        Informar constantes do algoritmo
        Gerar posição e velocidade das partículas randomicamente
Inicialização da otimização
        Para k=1, número de iterações
                Para i=1, número de partículas
                        Avaliar função f(x_k^i)
                        Testar convergência
                        Atualizar posição e velocidade
                Fim
        Fim
Apresentação dos resultados
```

Figura 2.9 – Pseudocódigo para um algoritmo PSO assíncrono

Fonte: Adaptado de Souza *et al.*, 2011.

A descrição do algoritmo de otimização por enxame de partículas assíncrono de forma mais detalhada segundo a programação desenvolvida no presente trabalho, é apresentada na subseção 3.1.2.

2.7 PROCESSAMENTO PARALELO

As diversas áreas da engenharia requerem cada vez mais computação intensiva, em virtude dos algoritmos complexos para análise de sistemas e do tamanho do conjunto de dados a serem processados. A fim de atender estas e outras finalidades, em 2005, a Intel e a AMD, desenvolveram as plataformas multiprocessadas. Segundo Masuero, 2009, na versão mais simples de um processador com múltiplos núcleos (*multi core*), cada núcleo ou *core* corresponde a circuitos eletrônicos de um processador completo, encapsulados juntos em um mesmo invólucro e compartilhando as vias de conexão com a placa mãe. Em sua versão mais sofisticada, os núcleos estão interconectados através de vias especiais de alta velocidade e tem um "cachê" e uma controladora de memória comum.

No entanto, ainda que a alta tecnologia dos novos computadores venha aumentando sua capacidade de processamento, os algoritmos continuam limitados a uma tarefa decomposta em um conjunto de instruções transmitidas e executadas de forma sequencial (ou serial). Nessa situação, uma única instrução é executada após a outra em um dado instante de tempo, como pode ser verificado na Figura 2.10(a). Dentro desse contexto, pode-se caracterizar a computação paralela como alternativa para executar múltiplas instruções ao mesmo instante de tempo, como mostrado na Figura 2.10(b).

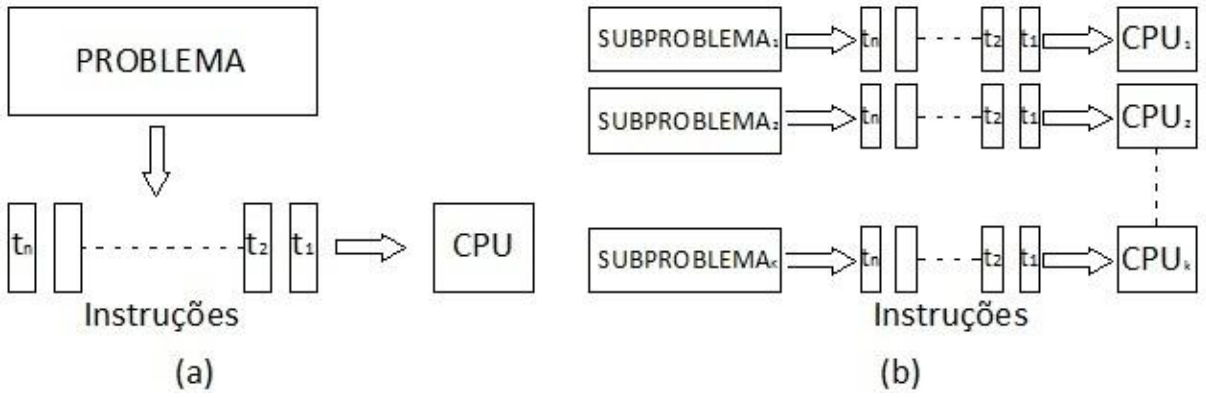


Figura 2.10 – (a) Processamento sequencial, (b) Processamento paralelo

Na computação paralela, como pode ser verificado na Figura 2.10(b), um problema é dividido em vários problemas menores (subproblemas). Esses subproblemas são distribuídos entre os vários processadores disponíveis e executados simultaneamente. Ao final da computação, cada subproblema origina um resultado, que é combinado com os demais resultados obtidos, gerando a solução para o problema tratado inicialmente. Sendo assim, uma característica evidente do paralelismo consiste em possibilitar resolver problemas de larga escala, com maior desempenho do que o processamento sequencial. Além disso, com a disponibilidade cada vez maior de plataformas com arquiteturas computacionais de múltiplos processadores, a resolução desses problemas vem se tornando cada vez mais eficiente, incentivando o desenvolvimento de algoritmos e métodos adequados à utilização nestas plataformas.

O ambiente para processamento paralelo pode ser representado pela Figura 2.11.

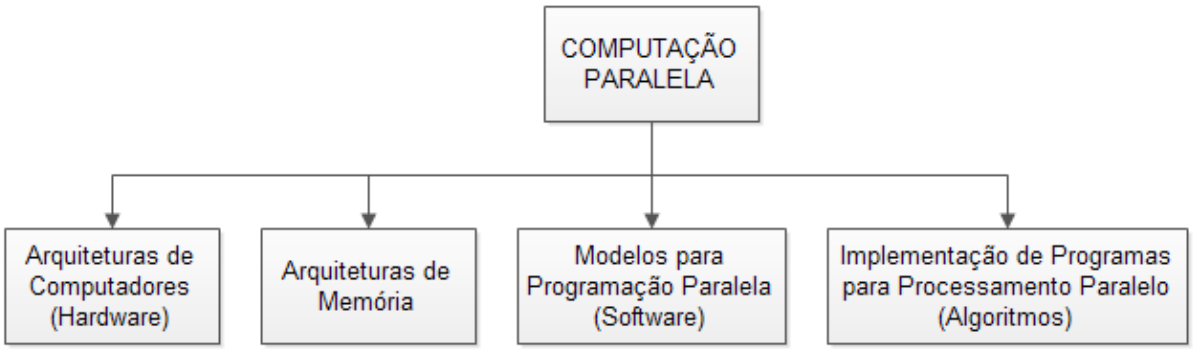


Figura 2.11 – Ambiente para processamento paralelo

As partes que compõem um ambiente paralelo, representados na Figura 2.11, são descritos nas seções subsequentes.

2.7.1 Arquitetura de Computadores

As arquiteturas de computadores compreendem os tipos de máquinas existentes, utilizadas no processamento paralelo. Flynn, 1966, propôs uma classificação baseada nos fluxos de instruções e dados dos processadores, apresentadas a seguir [Foster, 1995].

a. SISD (*Single Instruction, Single Data*): São computadores seriais tradicionais, com fluxo único de instruções para operar em dados em memória.

b. SIMD (*Single Instruction, Multiple Data*): Nesta arquitetura tem-se um fluxo único de instruções e um fluxo múltiplo de dados. Todas as unidades de processamento executam a mesma instrução, porém em diferentes conjuntos de dados, permitindo, portanto, um tipo de execução em paralelo;
c. MISD (*Multiple Instruction, Single Data)*: Esta arquitetura possui um fluxo múltiplo de instruções e somente um fluxo de dados, ou seja, os vários processadores desta arquitetura executam operações distintas num mesmo conjunto de dados,
d. MIMD (*Multiple Instruction, Multiple Data)*: Diferentes dados e programas podem ser alocados a diferentes processadores. Essa característica permite que cada processador possa executar instruções diferentes em dados diferentes num mesmo instante de tempo. A categoria MIMD é aplicada na maioria dos computadores com multiprocessadores desenvolvidos atualmente.

2.7.2 Arquiteturas de Memória

Um algoritmo paralelo pode ser organizado de várias maneiras, de acordo com o tipo de plataforma onde será executado. Genericamente, os modelos computacionais paralelos podem ser divididos em duas categorias principais: memória compartilhada e memória distribuída.

a. Memória Compartilhada: Na memória compartilhada podem ser discriminadas algumas características tais como: múltiplos processadores operam de maneira independente, mas compartilham os recursos de uma única memória; somente um processador por vez pode acessar um endereço na memória compartilhada;

b. Memória Distribuída: Nesta categoria, múltiplos processadores operam independentemente, sendo que cada um possui sua própria memória. Os dados são compartilhados através de uma rede de comunicação (*Message-Passing*), esta é utilizada pelo usuário que é responsável pela sincronização das tarefas, ou seja, o usuário é o responsável pelo envio e recebimento dos dados.

2.7.3 Modelos para Programação Paralela

Diversos modelos para geração de programas em paralelo podem ser implementados, independente da arquitetura de *hardware*, utilizando tanto memória compartilhada quanto memória distribuída. Dentre os principais modelos existentes, bem como atendendo ao interesse do presente trabalho, na subseção 2.8.1 serão detalhados os modelos presentes no *MATLAB*®, os quais utilizam o *MPI* para comunicação entre os processadores.

O modelo *MPI* (*Message Passing Interface* – Interface de Passagem de Mensagens) é uma biblioteca constituída de um conjunto de regras, especificações e funções escritas em Fortran, C e C++ para permitir que os processos troquem dados por meio do envio e recepção de mensagens entre si. Esta abordagem possibilita a criação de programas paralelos do tipo *SPMD*.

No contexto do presente trabalho, o *MPI* permite a troca de informações entre as partículas pertencentes aos enxames, essa característica será tratada posteriormente no paralelismo assíncrono o qual utiliza o comando *SPMD* do *MATLAB*®. Com isso, os enxames tendem a melhorar pela interação das partículas entre eles. Outra aplicação do *MPI* no presente trabalho esta presente no comando *PARFOR* do *MATLAB*®, o qual é utilizado para implementação do paralelismo síncrono, discutido posteriormente. No *PARFOR*, o *MPI* não possibilita a troca de partículas de um enxame.

2.7.4 Implementação de Programas para Processamento Paralelo

Ao se implementar um código essencialmente serial em uma extensão paralela, o procedimento inicial consiste em:

a. Escolher um algoritmo que permita a paralelização desejada;
b. Otimizar o código sequencial;
c. Realizar a paralelização.

Realizada esta etapa, segundo Foster, 1995, são necessárias algumas fases para análise do algoritmo, a fim de expor oportunidades de paralelização. Esta metodologia é conhecida como PCAM, descrita nos itens abaixo e ilustrada na Figura 2.12 [Foster, 1995].

a. Particionamento: O algoritmo deve ser “quebrado” em partes pequenas de código que podem ser executadas de forma independente. Na fase inicial da metodologia será necessário analisar todo o código fonte a fim de agrupar em grupos que possam processar em paralelo;

b. Comunicação: Etapa que consiste em verificar as dependências das variáveis compartilhadas e o fluxo de dados entre os processos, como variáveis globais e locais do código original. Com uma visão prática, procura-se estabelecer um caminho de comunicação entre os pequenos grupos de código particionado.
c. Aglomeração: Após particionar o algoritmo e realizar a comunicação, é obtido um conjunto de procedimentos que podem ser aglomerados para formar blocos sólidos.
d. Mapeamento: Destina-se a direcionar onde cada processo será executado, distribuindo cargas elevadas de processamento para nodos com maior poder computacional.

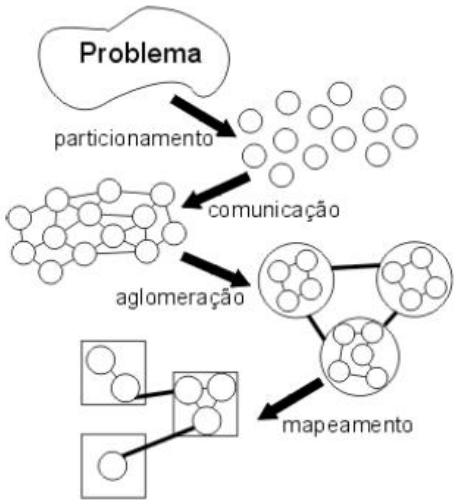


Figura 2.12 – Estrutura da Metodologia PCAM

Fonte: Foster, 1995.

O ambiente de desenvolvimento escolhido para a implementação dos programas é o *MATLAB*®, devido à facilidade para programação da lógica envolvida nos algoritmos (há um conjunto de funções para determinação de parâmetros necessários ao algoritmo utilizado) e permitir explorar o paralelismo das máquinas que vão hospedar o processamento. Entretanto, se tratando de velocidade de processamento, sabe-se que o ambiente escolhido não consiste na melhor opção se comparado a outros tipos de linguagens compiladas, como Fortran, C, C++ entre outras. Maiores detalhes quanto à utilização do *MATLAB*® em processamento paralelo são apresentados nas seções a seguir.

2.8 PROCESSAMENTO PARALELO NO MATLAB®

Um programa desenvolvido em *MATLAB*®, dentre outras características, possui uma linguagem interpretada e operada de forma sequencial, o que justifica, em partes, seu baixo desempenho computacional. Dessa forma, a utilização do *MATLAB*® em paralelo pode acelerar o tempo de resposta do programa mantendo a vantagem da facilidade de uso provida pelo ambiente.

Para se paralelizar o *MATLAB*®, inicialmente deve-se configurá-lo, informando o número de processadores existentes naquele ambiente de trabalho. Essa configuração (consultar APÊNDICE A) permite que a função *matlabpool*, identifique a disponibilidade de processadores para executar os trabalhos em paralelo. Todo o código que tenha algum processo realizado em paralelo deve inicialmente ser implementado com a função *matlabpool open* e finalizado com *matlabpool close*. Essas funções realizam a entrada e saída do laço para processamento paralelo do *MATLAB*®.

Outro fator a ser considerado, para transformar um algoritmo serial num algoritmo paralelo na linguagem *MATLAB*®, consiste em utilizar funções que o paralelizem concorrentemente. Sendo assim, dentre outras possibilidades, uma das bibliotecas que contém as funções de interesse para esse trabalho é a Caixa de Ferramentas de Computação Paralela (*Parallel Computing Toolbox - PCT*) do próprio *MATLAB*®.

2.8.1 *Parallel Computing Toolbox* (PCT)

A *Parallel Computing Toolbox* (*PTC*) é uma biblioteca de funções de propriedade da *Mathworks*, desenvolvida para aplicações em computação paralela utilizando o *MATLAB*®, possibilitando que o usuário execute uma tarefa em paralelo no ambiente de trabalho de uma máquina, usando até 8 processos para auxiliar o programa principal.

O programa principal é executado em um dos 8 processadores que estiver disponível, ou seja, após a realização de uma determinada tarefa pelos processadores, estes ficam disponíveis e a espera de uma posterior ordem de trabalho, a qual é delegada pelo programa principal. Esta decisão é tomada no primeiro processador que estiver disponível no intervalo entre uma tarefa e outra, o qual varia a depender desta disponibilidade a cada nova decisão.

Algumas das funções presentes na *PTC* são o *SPMD* e o *PARFOR*. Para que o código acesse estas funções, é necessário realizar a entrada no laço *matlabpool* (descrito na seção anterior), essa operação demanda um tempo de execução notadamente competitivo

comparado com outras bibliotecas (por exemplo: *MATLABMPI*, *PMATLAB* e etc.). Portanto, a utilização da biblioteca *PCT* é indicada para processamento de grandes quantidades de dados (como é o caso desta pesquisa), do contrário, a paralelização com o *PCT* possivelmente não apresente grandes vantagens.

Os itens a seguir detalham as funções necessárias ao desenvolvimento do trabalho.

a) *SPMD* (*Single Program Multiple Data* – Um Único Programa e Múltiplos Dados)

As funções *SPMD* permitem que sejam definidos blocos de código executados simultaneamente em vários processadores comunicando-se entre si, ou seja, um programa é executado no processador mestre e partes dele, definidas como blocos *SPMD*, são executadas nos processadores escravos com a possibilidade de comunicação entre eles. Quando todos os blocos *SPMD* estiverem concluídos, o programa continua a ser executado no processador mestre. A estrutura de comunicação *SPMD* pode ser verificada na Figura 2.13, a qual se enquadra na topologia de comunicação em estrela, a ser apresentada na subseção 2.9.2.

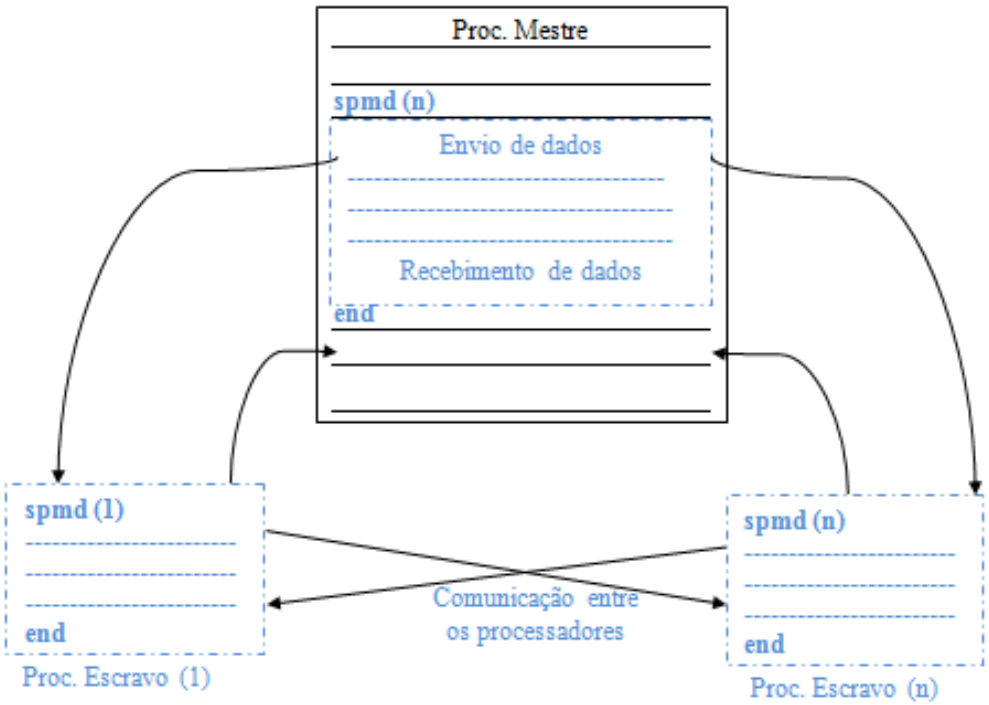


Figura 2.13 – Etapas do processamento paralelo com *SPMD*

Um aspecto importante do método consiste em possibilitar que um único programa seja executado em todos os processadores, gerando diferentes dados em cada um deles e permitindo que o conjunto desses dados seja acomodado no próprio processador para ficar a disposição dos demais processadores do grupo. Trazendo para o contexto deste trabalho, o *SPMD* permite que cada processador receba uma determinada partícula de um único enxame, e os dados gerados para aquela partícula possam ser armazenados nele mesmo, bem como, ficarem disponíveis aos demais processadores do bloco *SPMD.*

A função que possibilita ao *SPMD* armazenar os dados no processador que os gerou bem como os deixar disponíveis aos demais processadores do grupo é a *labBroadcast*. De forma análoga, pode-se dizer que os dados gerados por cada processador são enviados, pela função *labBroadcast*, para sua "nuvem" a qual pode ser acessada (por meio de funções como *labReceive*) por outros processadores. No entanto, dependendo da aplicação, como é o caso deste trabalho, esta operação aumenta os riscos de *deadlock*. Este fenômeno ocorre quando um determinado processador k_1 tenta acessar os dados presentes na "nuvem" do processador k_2, entretanto, esses dados estão sendo acessados pelo processador k_3 neste mesmo instante. Isto pode ser parcialmente contornado com a sincronização de operações e colocação de barreiras na programação paralela.

Outra característica que torna o método bastante atraente consiste na transferência de dados entre os processadores durante a execução das tarefas. A disposição do método em permitir este tipo de operação, trazendo para o contexto deste trabalho, possibilita a implementação da estratégia paralela assíncrona (a ser discutida na subseção 3.2.2). Para que esta estratégia possa ser desenvolvida, deve haver a comunicação entre os processadores que estão trabalhando em paralelo. Dessa forma, cada processador, ao selecionar sua melhor partícula, pode disponibilizá-la para os demais processadores do grupo. Dentre as várias funções do *SPMD* responsáveis pela realização deste tipo de operação, para a abordagem da pesquisa realizada, foi escolhida a função *labSendReceive*. Esta função, além de atender as condições necessárias à implementação proposta neste trabalho, permite que se estabeleça uma certa ordenação nas ordens de chamada das funções de comunicação, reduzindo os riscos de *deadlock* na execução dos processos.

As aplicações *SPMD* podem ser muito eficientes se tiverem seus dados bem distribuídos e o sistema hospedeiro for homogêneo. Do contrário, se os processos apresentam diferentes cargas de trabalho ou se a capacidade de processamento entre os processadores for

diferente, então o paradigma pode exigir algum esquema de balanceamento de carga capaz de adaptar a distribuição de dados em tempo de execução. Como é o caso da referente pesquisa, pois se tratando de grandes e complexos problemas de engenharia, as tarefas delegadas aos processadores podem assumir diferentes proporções as quais demandam diferentes cargas de trabalho.

A função *labBarrier* apresenta-se como uma alternativa ao inconveniente citado. Ela permite ao programa prosseguir somente após o término das operações de todos os processadores, ou seja, o algoritmo pode prosseguir somente se os dados de todas as partículas estiverem disponíveis. Isso garante que durante os processos seja mantida a distribuição uniforme dos dados entre os processadores, exigência para o funcionamento do *SPMD*.

Aplicações típicas para o *SPMD* são utilizadas quando for necessária a comunicação e sincronização entre processadores. Alguns casos são:

- Programas que levam muito tempo para serem executados – *SPMD* permite que vários processadores calculem soluções simultaneamente;
- Programas que operam com grandes conjuntos de dados – *SPMD* distribui os dados para vários processadores.

b) *PARFOR* (*Loop* Paralelo)

A paralelização de um bloco de códigos com múltiplos processadores simultaneamente, pode ser realizada utilizando operadores de *loops* paralelos. Neste contexto, utilizando o prefixo "*PAR*" seguido do nome do operador de *loop* normal "*FOR*" compõe-se o comando *PARFOR*, presente na biblioteca *PTC* do *MATLAB*®, responsável por promover a paralelização de um determinado código com uma implementação relativamente simples (comparado com *SPMD*).

As funções presentes no comando *PARFOR* possibilitam ao algoritmo realizar a paralelização próxima à que ocorre utilizando o comando *SPMD*, exceto quanto à comunicação entre os processadores, ou seja, o *PARFOR* não permite que os processadores se comuniquem entre si. A Figura 2.14 ilustra a lógica envolvida durante a execução do *PARFOR*. Nela, verifica-se que um programa, executado no processador mestre, é repartido em blocos *PARFOR* e distribuídos entre os processadores escravos. Quando todos os blocos *PARFOR* estiverem concluídos, o programa continua a ser executado no processador mestre. Cabe salientar que neste caso não pode haver comunicação entre os processadores escravos,

devendo ser as execuções completamente independentes entre si. O comando *PARFOR* se enquadra na topologia mestre-escravo a ser apresentada, segundo o contexto deste trabalho, na subseção 2.9.1.

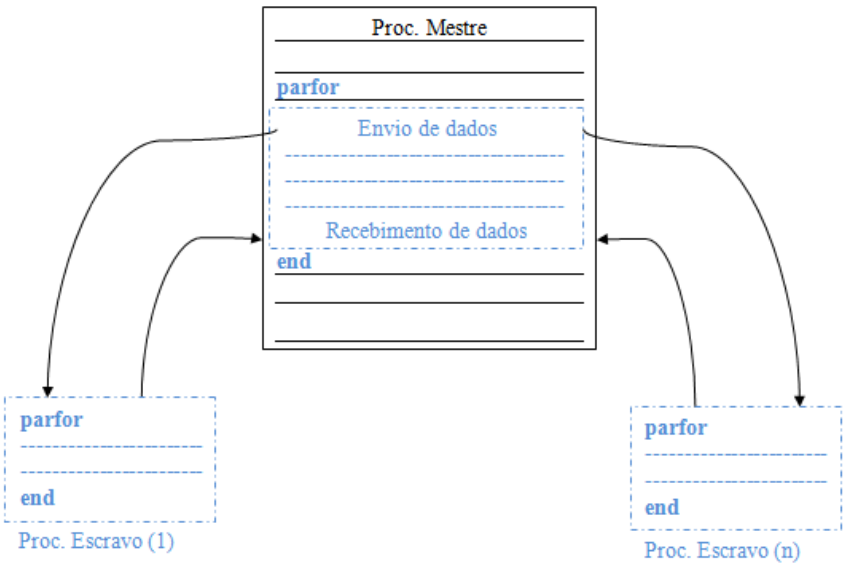


Figura 2.14 – Etapas do processamento paralelo com *PARFOR*

Um grande atrativo do método é a sua fácil implementação. Consiste apenas na substituição do *loop FOR* por *PARFOR* na parte do código que deva ser processado em paralelo, entretanto, algumas restrições e cuidados devem ser seguidos.

Uma propriedade que deve ser mantida para paralelizar um bloco *PARFOR*, consiste em utilizar apenas variáveis reconhecidas pelo *loop PARFOR*, do contrário (havendo dependência de variáveis desconhecidas) o *MATLAB*® não reconhece a variável como um dos tipos pré-determinados e consequentemente fica incapaz de executar o código, notificando-o com um erro.

A Figura 2.15 apresenta um código com os tipos de variáveis que o *MATLAB*® é capaz de reconhecer dentre de um *loop PARFOR*.

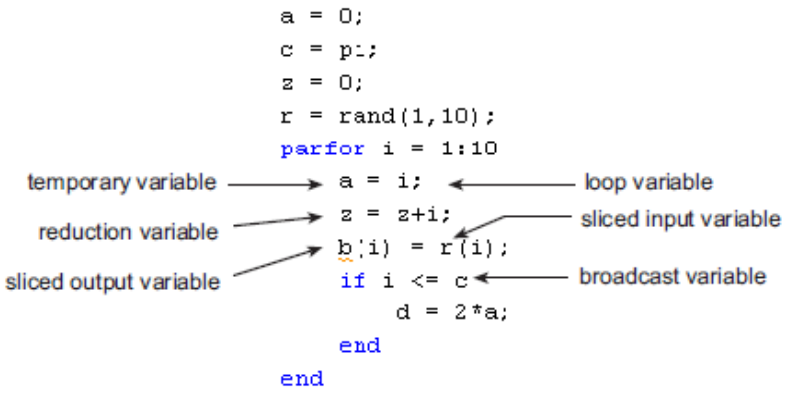


Figura 2.15 – Código implementado com variáveis reconhecidas pelo *PARFOR*

Fonte: Matwhorks, 2011.

As definições das respectivas variáveis, apresentadas a seguir, podem ser encontradas no *Help* do *MATLAB*®.

- *Loop variable*: variável que cresce dentro do *loop*, e serve de índice para os vetores;
- *Sliced input variable*: é a variável que possui os dados acessados conforme o índice indicado pelo *PARFOR*, ou seja, varia a cada iteração.
- *Sliced output variable*: tem seus elementos definidos conforme o índice indicado pelo *PARFOR* a cada iteração;
- *Broadcast*: esta variável está definida fora do *loop* e pode se acessada dentro dele, mas sem alterar o valor nela armazenado;
- *Reduction*: esta variável vai incrementando seu valor a cada iteração, e independente da ordem das iterações;
- *Temporary*: esta variável é criada dentro do *loop PARFOR*, mas não pode ser acessada fora dele.

Além das variáveis, outras restrições quanto ao seu uso devem ser atendidas, como por exemplo: ambiguidade entre o nome das variáveis, transparência, variáveis distribuídas como referência para uma função com laço e etc., as quais podem ser encontradas no *Help* do *MATLAB*®.

2.9 MODELOS PARALELOS DO PSO

O alto desempenho computacional dos modernos computadores somado as atuais pesquisas realizadas com métodos determinísticos, heurísticos e híbridos buscam constantemente melhorar o tempo de processamento de inúmeros problemas de engenharia, no entanto, alguns destes permanecem computacionalmente custosos. Segundo Yang, 2010, não há um método universal que forneça garantidamente a solução ótima com o melhor desempenho para todos os problemas de otimização global.

Yang, 2010, denomina os métodos existentes como '*No Free Lunch Theorems'* (*NLF theorems*). Este teorema afirma que se um determinado algoritmo A superar um algoritmo B na busca do extremo de uma determinada função de custo, então o algoritmo B superará o algoritmo A ao extremizar uma outra função de custo. De acordo com Yang, o teorema *NFL* prova que o desempenho médio sobre todas as funções de custo é o mesmo para todos os algoritmos de busca. O autor afirma que o desempenho independe do algoritmo, ou seja, todos os algoritmos de otimização apresentarão o mesmo desempenho se considerar a média do desempenho de várias funções diferentes.

É importante ressaltar que um algoritmo pode apresentar melhor desempenho para um determinado problema de otimização, entretanto, não significa que para funções diferentes seu desempenho continue sendo melhor do que os demais algoritmos. Isso dá indícios que não existe um método universal que garanta um melhor desempenho sob todos os problemas de otimização existentes.

Sendo assim, inspirados em encontrar uma forma de melhorar o desempenho destes algoritmos, pesquisadores de diversas áreas veem desenvolvendo modelos paralelos aos algoritmos seriais desenvolvidos.

Um dos métodos heurísticos frequentemente explorados na computação paralela são os AGs (AGP) [Cantú-Paz, 2000], apresentando-se basicamente em três formas paralelas: mestre-escravo, onde apenas a função de aptidão *fitness* é distribuída entre os processadores; Ilhas (malha grossa), onde subpopulações evoluem paralelamente trocando informações através de migrações, segundo uma dada estratégia; Celular (malha fina), onde cada candidato (indivíduo) à solução é alocado em um processador/processo (célula) e geralmente segue uma arquitetura de malha 2D, interligada em forma de toroide.

Outro método que vem conquistando grande popularidade devido a sua robustez, eficiência e simplicidade, é o PSO. Entretanto, como as demais heurísticas, quando deparado com problemas grandes e complexos sua eficiência pode ser reduzida drasticamente. Esse

comportamento é devido ao aumento na quantidade de avaliações da função de aptidão *fitness* (forma pela qual o enxame evolui em direção ao ótimo) causado pela dificuldade do algoritmo de encontrar o ótimo. Por outro lado, o PSO pode ser facilmente paralelizado, o que torna a computação paralela uma alternativa atraente e dependendo do problema, indispensável para utilização deste método.

O primeiro trabalho de paralelização de PSO foi desenvolvido por Schutte *et al*., 2004, propondo uma implementação síncrona com o modelo mestre-escravo, denominado por eles de *PSPSO* (*Parallel Syncronous Particle Swarm Optimization*). Desde então, arquiteturas paralelas do PSO têm sido largamente empregadas em problemas de otimização. Aplicações deste tipo são encontradas nos trabalhos de Belal e El-Ghazawi, 2004, Mostaghim *et al*, 2007, dentre outros, ressaltando-se desde já que, na seção 2.10, serão abordados os métodos de paralelização do PSO propostos por Koh *et al*. 2006 e Venter, 2005, úteis ao desenvolvimento deste trabalho.

A ideia básica da maioria dos programas paralelos, de acordo com estudo realizado (seção 2.7) consiste em dividir uma tarefa em partes menores e solucioná-las simultaneamente, usando múltiplos processadores. Esta definição pode ser aplicada a programação paralela do PSO, que de um modo geral consiste em distribuir as partículas de um enxame entre os processadores disponíveis. A topologia ou estrutura de comunicação destas partículas caracterizam o modelo de programação adotado para paralelizar o PSO.

Os modelos de programação paralela do PSO podem ter diversas classificações, estas divergem de autor para autor. A classificação feita neste trabalho estabelece dois modelos principais de implementação do PSO em paralelo, o modelo de paralelização global e modelo de múltiplas populações.

2.9.1 Modelo de Paralelização Global

Os modelos de paralelização global geralmente são implementados em algoritmos com a topologia do tipo mestre-escravo (Figura 2.16). Nesses algoritmos, existe um processador principal (ou mestre) que controla a evolução do enxame. Os demais processadores secundários (ou escravos) recebem frações do enxame e executam as operações pertinentes ao algoritmo. Por analogia, o processador mestre decide qual partícula possui a maior contribuição para o enxame encontrar o ótimo global. Já os processadores escravos são

responsáveis apenas pelo cálculo da função de aptidão *fitness* das partículas. A comunicação entre mestre e escravos ocorre quando os escravos enviam ao mestre seus valores individuais e recebem deste o melhor valor global. O papel do mestre é procurar a melhor dentre as partículas (a que possuir a melhor posição no espaço de busca) e atualizar o ótimo global.

Alguns autores argumentam que a falta de comunicação entre os processadores escravos no processamento paralelo, impede o algoritmo de apresentar alguma melhora no ótimo global e seu único ganho consiste na redução do tempo computacional.

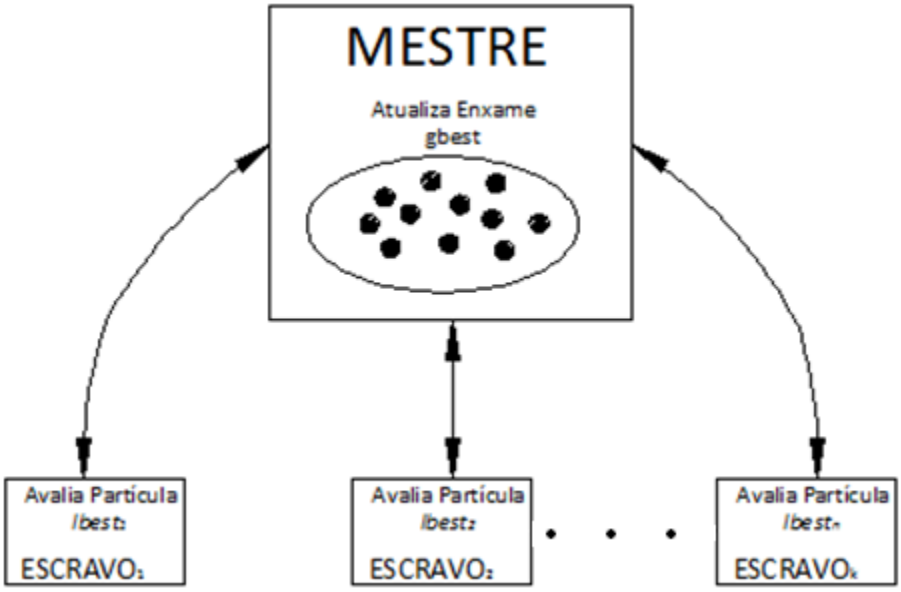


Figura 2.16 – Topologia de comunicação mestre-escravo

2.9.2 Modelo de Múltiplas Populações

No modelo de múltiplas populações, também conhecido como modelo de granularidade grossa, várias subpopulações são distribuídas nas várias unidades de processamento disponíveis, onde evoluem isoladamente e em paralelo.

O conceito de granularidade grossa para este método foi dado devido ao alto índice de comunicação entre os enxames durante sua execução, ou seja, neste modelo, os enxames se comunicam e a troca de informações entre partículas pode acontecer entre quaisquer processadores.

Nesta estratégia de comunicação, cada um dos enxames seleciona a sua melhor partícula e envia para cada um dos demais enxames. Dessa forma as piores partículas de cada enxame são substituídas pela melhor partícula de cada um dos demais enxames, evitando a

perda de tempo com uma subpopulação cujo resultado seja ruim, pois esta será melhorada com a chegada de partículas dos enxames vizinhos.

A Figura 2.17 apresenta o modelo de múltiplas populações com topologia tipo estrela, que pode ser representado por um grafo completamente conectado. Nesta topologia, quaisquer dos enxames podem receber ou enviar partículas para qualquer outro enxame.

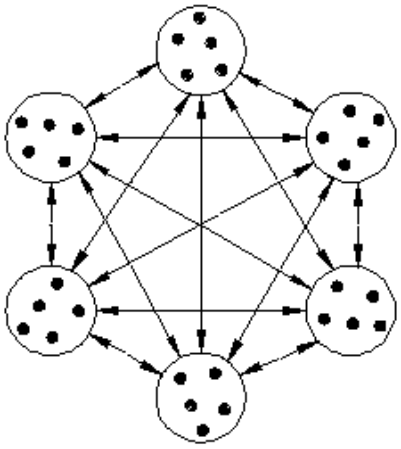


Figura 2.17 – Topologia de comunicação em estrela

Uma desvantagem deste método é que o alto índice de comunicação aumenta o tempo gasto com a troca de dados entre os processadores à medida que aumentam o número de processadores, o que pode impactar nos tempos de processamento dos algoritmos.

2.10 IMPLEMENTAÇÃO PARALELA DO PSO

Muitas pesquisas são desenvolvidas no campo da computação paralela utilizando abordagens síncronas do algoritmo PSO [Schutte *et al.*, 2004], no entanto, Koh *et al.*, 2006, propuseram uma nova estratégia adaptativa assíncrona. Na subseção 2.10.1, são apresentadas as implementações, síncrona e assíncrona, aplicadas na computação paralela segundo seus respectivos autores.

2.10.1 Implementação Paralela do PSO Síncrona e Assíncrona

De acordo com Venter, 2005, o modelo básico de implementação paralela síncrona do PSO consiste em avaliar as partículas dentro de uma iteração em paralelo, sem alterar a lógica global do algoritmo em si. Nesta implementação, todas as partículas dentro de uma *revoada*

são enviadas para o ambiente de computação paralela, e o algoritmo deve esperar que todas as partículas sejam analisadas antes de prosseguir para a próxima iteração.

Embora a implementação síncrona forneça uma extensão fácil para um ambiente em paralelo, não é uma solução ideal e geralmente resulta em baixa eficiência. Segundo o mesmo autor, o problema com esta implementação é a dificuldade em manter todos os processadores trabalhando no final de cada iteração, antes de iniciar nova iteração.

Venter, 2005, cita três situações que resultam na ociosidade de alguns ou na maioria dos processadores no final de cada iteração.

I. O número de partículas não é múltiplo inteiro do número de processadores;
II. Um ambiente heterogêneo de computação distribuída, onde processadores com diferentes velocidades computacionais são combinados em um ambiente de computação paralela;
III. Usando uma simulação numérica para avaliar cada partícula, onde o tempo de simulação necessário depende da partícula a ser analisada.

Sendo assim, é possível perceber que o aumento no número de processadores ociosos resultará na redução da eficiência do algoritmo paralelo do PSO síncrono.

Buscando resolver o problema da baixa eficiência do algoritmo paralelo do PSO síncrono, Koh *et al.*, 2006, propuseram uma implementação assíncrona para o PSO paralelo (*Parallel Assyncronous Particle Swarm Optimization – PAPSO*). O objetivo da estratégia era não ter processadores ociosos com o avanço das iterações.

De acordo com o autor, além da paralelização, esta estratégia envolve uma modificação no algoritmo sequencial assíncrono do PSO. Uma vez que os vetores velocidade e posição da partícula *i*, são calculados usando a melhor posição local $xlbest_{i,j}(t)$ e a melhor posição global $xgbest_j(t)$ até k interações, a ordem da avaliação da função de aptidão "*fitness"* da partícula afeta o resultado da otimização. No entanto, o algoritmo PSO não restringe a ordem em que as avaliações das partículas são executadas pela *fitness*, consequentemente, isso permite que a ordem da partícula seja alterada continuamente, dependendo da velocidade com que cada processador completa a avaliação da *fitness*, eliminando assim a utilização do contador de interação *k* no algoritmo assíncrono do PSO.

Para melhor entender o algoritmo proposto por Koh *et al.*2006, a Figura 2.18 apresenta um fluxograma que compara o *PAPSO* com o *PSPSO*.

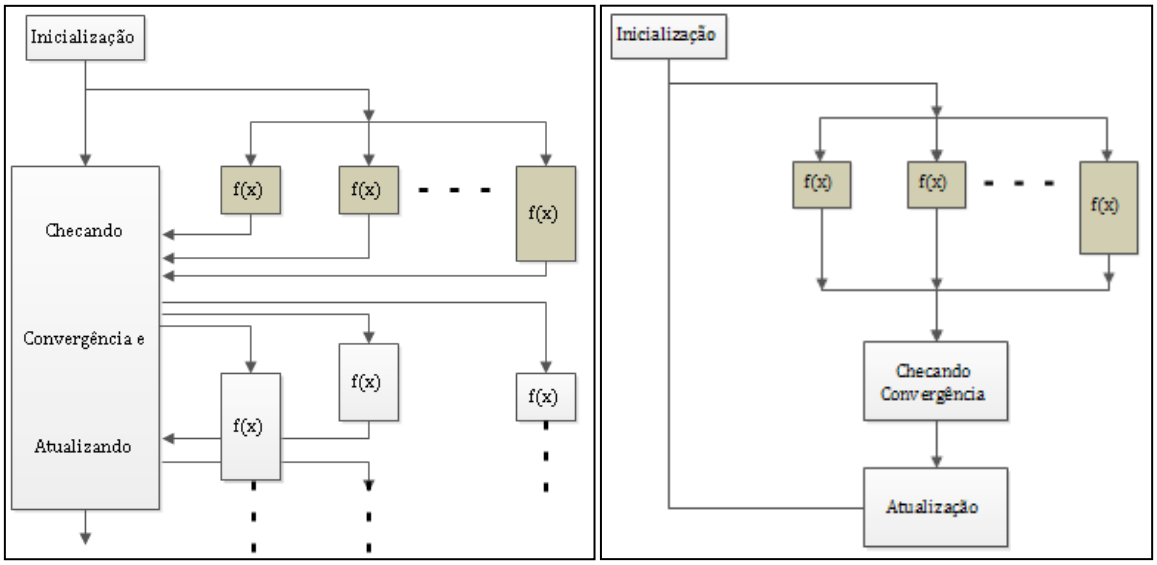


Figura 2.18 – Diagrama de blocos para: (a) *PAPSO*; (b) *PSPSO*

Fonte: Koh *et al.*, 2006.

As caixas em cinza que aparecem no fluxograma da figura acima indicam o primeiro conjunto de partículas avaliadas por cada algoritmo.

A Figura 2.18(a) mostra que a atualização da posição e velocidade da partícula ocorre continuamente baseada em informações atualmente disponíveis. Já na figura 2.18(b), a atualização da posição e velocidade da partícula ocorre apenas após o termino da iteração de todas as partículas, usando sincronização global.

Sendo assim, de acordo com os autores, enquanto o *PSPSO* usa balanceamento de carga estática, para determinar a carga de trabalho do processador durante o tempo de execução, o *PAPSO* usa balanceamento de carga dinâmica, reduzindo assim o desequilíbrio da carga.

O algoritmo *PAPSO* proposto por Koh *et al.*, 2006, segue o paradigma mestre-escravo. O processador mestre detém a fila de partículas prontas para enviar aos processadores escravos, ele executa todos os processos de tomada de decisão, tais como: atualização de velocidade/posição e verificação da condição de parada do algoritmo. Ele não executa quaisquer avaliações da função objetivo. Os processadores escravos avaliam repetidamente a função objetivo utilizando as partículas que lhes são atribuídas.

As tarefas desempenhadas pelos processadores mestre/escravos podem ser organizadas da seguinte forma [Koh, *et al.*, 2006]:

- Processador Mestre

1. Inicializa todos os parâmetros de otimização, posições e velocidades iniciais das partículas;
2. Mantém uma fila de partículas para os processadores escravos avaliar;
3. Atualiza posição e velocidade da partícula com base em informações atualmente disponíveis ($xlbest_{i,j}(t)$, $xgbest_j(t)$);
4. Envia a posição da partícula seguinte da fila ($x_{i,j}(t)$) para um processador escravo disponível;
5. Recebe valores da *fitness* analisada pelos processadores escravos;
6. Verifica convergência.

- Processador Escravo
  1. Recebe a posição da partícula ($x_{i,j}(t)$) do processador mestre;
  2. Analisa a posição da partícula ($x_{i,j}(t)$) com a função de aptidão *fitness*;
  3. Envia o valor da *fitness* para o processador mestre.

3 DESENVOLVIMENTO DOS ALGORITMOS PROPOSTOS

Neste capítulo, é realizado o detalhamento dos códigos desenvolvidos e implementados neste trabalho. Nas seções 3.1 a 3.2, a apresentação dos códigos assume o comportamento do algoritmo do ponto de vista da partícula. Já na seção 3.3, procurou-se abordar o algoritmo ao contexto da otimização de estruturas treliçadas.

3.1 DETALHAMENTO DOS CÓDIGOS SEQUENCIAIS PROPOSTOS

Para um melhor entendimento da implementação desenvolvida nos algoritmos seriais, síncrono e assíncrono, desta seção, aconselha-se rever as seções 2.5 e 2.6, as quais serviram de instrução para os códigos propostos.

3.1.1 Detalhamento do Código Sequencial Síncrono do PSO (SSPSO)

No desenvolvimento do código SSPSO, procurou-se manter a características apresentadas na seção 2.5. Os itens a seguir descrevem as etapas do algoritmo e posteriormente, nas Figuras 3.1 e 3.2, estão ilustrados os respectivos pseudocódigo e fluxograma.

1. Inicialmente realiza-se a entrada de dados no algoritmo, alimentando-o com os seguintes parâmetros:
   - Dimensão do espaço de busca (m);
   - Número de partículas do enxame (n);
   - w, c_1e c_2;
   - Valores máximos e mínimos para o espaço de busca (restrições laterais);
   - Condição de parada do algoritmo (tol_{COV} - tol_{FP}), detalhada na subseção 3.2.1, pseudocódigo_6;
2. O algoritmo é inicializado com valores de posição e velocidades gerados aleatoriamente;
3. O passo seguinte consiste na avaliação das partículas pela função de aptidão *fitness*. A determinação do valor dado pela *fitness* leva em consideração as restrições secundárias atribuídas à função objetivo;

4. Adota-se a melhor posição encontrada pela partícula $xlbest_i$, como sendo, a melhor posição do enxame *xgbest*;
5. Definidos os valores iniciais para posição e velocidade da partícula, o algoritmo entre em um laço de otimização até atingir a condição de parada;
6. É então realizada a atualização das velocidades e posições de cada partícula. A posição deve respeitar os limites (restrições laterais) do espaço de busca. Em caso do não cumprimento desta condição, seus valores são corrigidos para os respectivos valores mínimos e máximos do espaço de busca. A atualização da velocidade e posição é obtida pelas Equações 2.6 e 2.2 respectivamente;
7. Novamente a função *fitness* é recalculada, agora utilizando as posições atualizadas pela Equação 2.2;
8. Em seguida, é realizada a atualização dos ótimos locais $lbest_i$ e globais *gbest*, levando em conta os novos valores da função *fitness*, conforme descrito a seguir:
    - Comparar o valor da *fitness* da partícula com o $lbest_i$da partícula. Se o valor corrente é melhor que $lbest_i$, então o valor de $lbest_i$ passa a ser igual ao valor da *fitness*, e a posição do $xlbest_i$ passa a ser igual a posição atual;
    - Comparar o valor da *fitness* com o prévio melhor valor do enxame. Se o valor atual é melhor que o *gbest*, atualizar o valor de *gbest* para o índice e valor da *fitness* da partícula atual.
9. Diante das novas posições ao final da *revoada*, aplica-se o critério de parada. Ao ser satisfeito, a solução do problema é apresentada. Do contrário, reinicializa-se uma nova *revoada* (com a atualização do enxame), até que a condição de parada seja atingida.

Insira os dados de entrada: número de partículas n, dimensão do espaço de busca (ou variáveis de projeto) m, parâmetro cognitivo c_1, parâmetro social c_2, momento de inércia w, fator de constrição χ, tolerância para as iterações tol_{COV} - tol_{FP}, limites superiores e inferiores para o espaço de busca x_j^{Sup} e x_j^{Inf};

Gere um Enxame inicial aleatório e inicialize os melhores valores locais:

Para cada partícula i no Enxame faça

Para cada variável de projeto *j* faça

$r = uniforme[0,1]$

$x_{i,j}^0 = x_j^{Inf} + r\left(x_j^{Sup} - x_j^{Inf}\right)$

$v_{i,j}^0 = 0$

Selecione o ótimo local das variáveis de projeto como o valor atual

$xlbest_{i,j} = x_{i,j}$

Fim

Selecione o ótimo local da função objetivo como o valor atual

Determine $f(xlbest_i) = f(x_i^k)$

Fim

Faça iteração no Enxame para encontrar a partícula com as variáveis de projeto que tenham o mínimo da função objetivo: faça o laço com o critério de convergência;

Para cada partícula i no Enxame faça

$r_1 = uniforme[0,1]$

$r_2 = uniforme[0,1]$

$v_i^{k+1} = \chi[wv_i^k + c_1 r_1(xlbest_i^k - x_i^k) + c_2 r_2(xgbest_i^k - x_i^k)]$

$\boldsymbol{x}_i^{k+1} = \boldsymbol{x}_i^k + \boldsymbol{v}_i^{k+1}$

Fim

Para cada partícula i até o critério de convergência escolhido faça

Avalie para cada partícula i a função objetivo $f_i(\boldsymbol{x}_i^{k+1})$

Atualize o ótimo local e a sua respectiva função objetivo

Para cada partícula i faça

Se $f(\boldsymbol{x}_i^{k+1}) < f(xlbest_i^k)$

Então $f(xlbest_i^{k+1}) = f(\boldsymbol{x}_i^{k+1})$ e $xlbest_i^{k+1} = \boldsymbol{x}_i^{k+1}$, sucessivamente

Fim

Encontre a partícula mínima da função objetivo $min(f(\boldsymbol{x}_i^{k+1}))$

Se $min\left(f(\boldsymbol{x}_i^{k+1})\right) < f(xgbest^k)$

Então $f(xgbest^{k+1}) = min\left(f(\boldsymbol{x}_i^{k+1})\right)$ e $xgbest^k = \boldsymbol{x}_{index(min(f_i(x_i)))}$, sucessivamente

Fim

Figura 3. 1 – Pseudocódigo do algoritmo SSPSO

Fonte: Adaptado de Gomes, 2009.

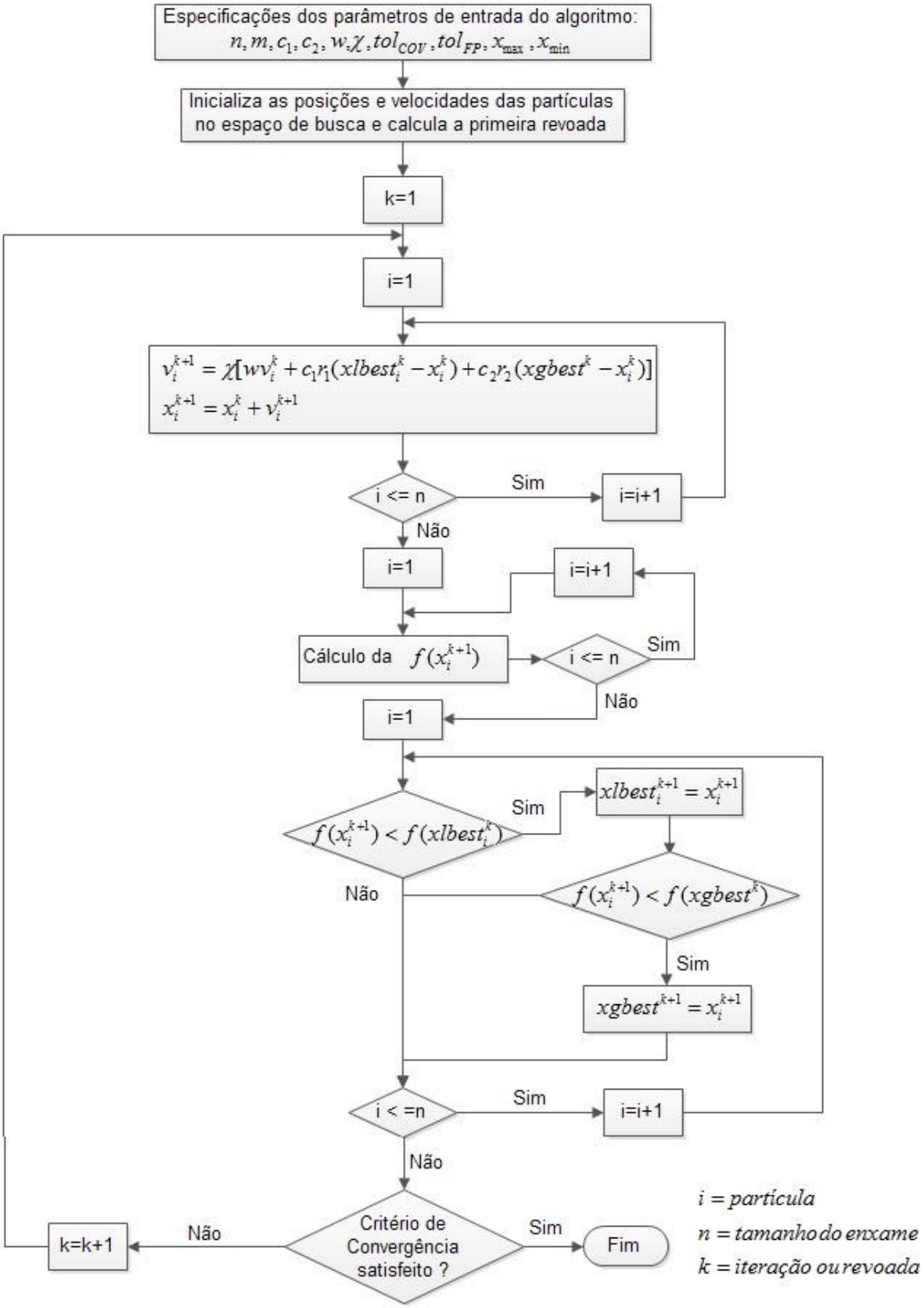


Figura 3. 2 – Fluxograma da implementação do SSPSO

3.1.2 Detalhamento do Código Sequencial Assíncrono do PSO (SAPSO)

Análogo ao método assíncrono, apresentado na seção 2.6, desenvolveu-se o algoritmo SAPSO proposto no presente trabalho. A diferença entre o método síncrono e assíncrono está na atualização do ótimo global, velocidade e posição das partículas quando o algoritmo entra no laço de otimização, portanto, em posse do código proposto para o PSO síncrono (SSPSO), realizou-se a alteração da etapa pertinente à implementação do método assíncrono. Como pode ser verificado no pseudocódigo e fluxograma das Figuras 3.3 e 3.4.

Insira os dados de entrada: número de partículas n, espaço de busca (ou número de variáveis de projeto) m, parâmetro cognitivo c_1, parâmetro social c_2, momento de inércia w, fator de constrição χ, tolerância para as iterações tol_{COV} - tol_{FP}, limites superiores e inferiores para o espaço de busca x_j^{Sup} e x_j^{Inf};

Gere um Enxame inicial aleatório e inicialize os melhores valores locais:

Para cada partícula i no Enxame faça

Para cada variável de projeto j faça

$r = uniforme[0,1]$

$x_{i,j}^0 = x_j^{Inf} + r\left(x_j^{Sup} - x_j^{Inf}\right)$

$v_{i,j}^0 = 0$

Selecione o ótimo local das variáveis de projeto como o valor atual

$xlbest_{i,j} = x_{i,j}$

Fim

Selecione o ótimo local da função objetivo como o valor atual

Determine $f(xlbest_i) = f(x_i)$

Fim

Faça iteração no Enxame para encontrar a partícula com as variáveis de projeto que tenham o mínimo da função objetivo: faça o laço com o critério de convergência;

Para cada partícula i no Enxame faça

$r_1 = uniforme[0,1]$

$r_2 = uniforme[0,1]$

$v_i^{k+1} = w_p v_i^k + c_1 r_1 (xlbest_i^k - x_i^k) + c_2 r_2 (xgbest_i^k - x_i^k)$

$x_i^{k+1} = x_i^k + \chi v_i^{k+1}$

Avalie a partícula i com a função objetivo $f_i(\boldsymbol{x}_i)$

Atualize o ótimo local e a sua respectiva função objetivo

Se $f\left(\boldsymbol{x}_i^{k+1}\right) < f(xlbest_i^k)$

Então $f\left(xlbest_i^{k+1}\right) = f(\boldsymbol{x}_i^{k+1})$ e $xlbest_i^{k+1} = \boldsymbol{x}_i^{k+1}$sucessivamente

Encontre a partícula mínima da função objetivo $min(f(\boldsymbol{x}_i))$

Se $min\left(f\left(\boldsymbol{x}_i^{k+1}\right)\right) < f(xgbest^k)$

Então $f(xgbest^k) = min\left(f\left(\boldsymbol{x}_i^{k+1}\right)\right)$ e $xgbest^{k+1} = \boldsymbol{x}_{index(min(f_i(x_i)))}$, sucessivamente

Fim

Fim

Figura 3. 3 – Pseudocódigo do algoritmo SAPSO

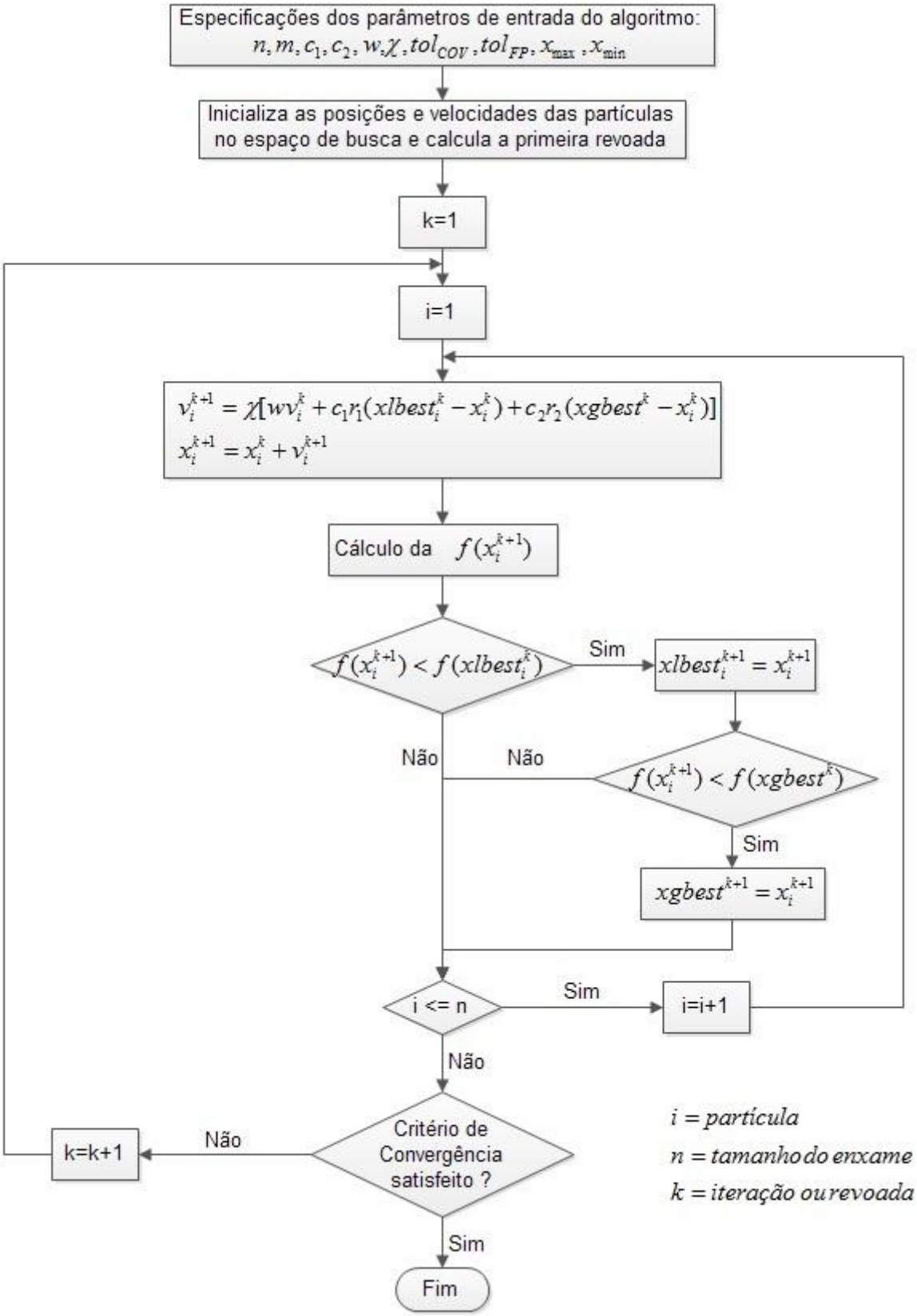


Figura 3. 4 – Fluxograma da implementação do SAPSO

Ao analisar o fluxograma e pseudocódigo do SAPSO, é possível perceber que o método assíncrono permite ao algoritmo encontrar um novo ótimo global a cada *pseudo-revoada*, e se necessário, ajustar seu valor para aplicá-lo nas partículas ainda não atualizadas da atual iteração. Esta característica tende a fornecer informações adicionais ao enxame, melhorando seu desempenho em termos de número de avaliações da função objetivo.

3.2 DETALHAMENTO DOS CÓDIGOS PARALELOS PROPOSTOS

Os algoritmos paralelos desenvolvidos neste trabalho utilizaram os paradigmas descritos na seção 2.9. Para a versão paralela síncrona, foi aplicado o modelo mestre-escravo para comunicação entre os processadores. A versão paralela assíncrona utilizou o modelo mestre-escravo, para a comunicação entre o processador mestre e os processadores escravos, e o modelo de múltiplas populações para a comunicação entre os processadores escravos. A tarefa desempenhada pelo processador mestre será desempenhada em qualquer um dos processadores disponíveis da plataforma multiprocessada, o qual é selecionado segundo a disponibilidade dos processadores existentes, ou seja, quando um processador escravo realizou sua tarefa, ele esta disponível para desempenhar o papel de processador mestre. Esta etapa pode ser realizada em qualquer um dos processadores a cada nova tomada de decisão do mestre, a depender apenas do primeiro processador que estiver disponível. Este comportamento ocorre durante a execução dos códigos paralelos síncronos e assíncronos.

3.2.1 Detalhamento do Código Paralelo Síncrono do PSO (PSPSO)

O código paralelo desenvolvido para versão síncrona do PSO é simples de ser obtido a partir da sua versão serial. No presente trabalho, utilizamos a forma descrita por Venter, 2005, na subseção 2.10.1, cuja ordem de execução das tarefas permanece a mesma, com a diferença de que a função objetivo é calculada paralelamente aos demais cálculos do algoritmo. Neste contexto há uma coerência entre a forma serial e paralela do algoritmo, ou seja, as operações do algoritmo paralelo ocorrem na mesma ordem da implementação do algoritmo serial original, sendo assim, a paralelização não deve afetar o resultado final obtido quando uma mesma semente de números aleatórios for utilizada. Ressalta-se desde já que, nos algoritmos

desenvolvidos neste trabalho, a parte sequencial do código será executada no processador mestre enquanto a parte paralela nos processadores escravos.

A Figura 3.5 apresenta o fluxograma da implementação do algoritmo PSPSO. Para a comunicação entre os processadores mestre e escravos foi utilizado o operador de *loop* paralelo *PARFOR* do *MATLAB*®, detalhado na subseção 2.8.1.

A paralelização do problema proposto, utilizando o comando *PARFOR*, consiste em paralelizar um bloco de códigos com múltiplos processadores simultaneamente. Durante o processamento o processador mestre equilibra a carga entre os processadores escravos para que nenhum processador fique ocioso, ou seja, mantém a carga de trabalho uniforme entre os processadores escravos. Em outras palavras pode-se dizer que, quando um processador escravo estiver disponível, este receberá mais tarefas do processador mestre, de modo que sua carga de trabalho seja próxima a carga de trabalho presente nos demais processadores escravos.

A geração das posições iniciais das partículas é feita aleatoriamente na forma sequencial do algoritmo como pode ser visto no pseudocódigo_1.

Pseudocódigo_1**:** Inicialização da Posição das Partículas Aleatoriamente

Para cada partícula *i*, faça

Gera posições x_i de maneira aleatória

Determina $f(lbest_i) = f(x_i)$

$xlbest_i = x_i$

Fim_*Para*

Em seguida, ocorre a atualização dos valores ótimos *gbest* da primeira *revoada* e sua respectiva melhor posição $xgbest$, também segundo a implementação sequencial, como é possível ser verificado no pseudocódogo_2.

Pseudocódigo_2: Atualização da Melhor Posição do Enxame

Para cada partícula *i* faça

Se ($lbest_i < gbest_k$)

Então $gbest_k = lbest_i$ e $xgbest = x_i$

Fim_*Se*

Fim_*Para*

A primeira parte do laço de otimização é responsável pela atualização da posição e velocidade de todas as partículas. Para esta etapa, foi mantida a forma sequencial do algoritmo, como está representado no pseudocódigo_3.

Pseudocódigo_3: Atualização da Posição e Velocidade das Partículas

Para cada partícula *i* faça

Para cada variável de projeto *j* faça

Atualiza velocidade $vnew_{i,j}$ e posição $xnew_{i,j}$

Verifica *Se* a posição $xnew_{i,j}$ respeita as restrições $xmax_{i,j}$ e $xmin_{i,j}$ Fim_*Se*

Fim_*Para* (variáveis de projeto)

Fim_*Para*

A etapa posterior (a avaliação da função objetivo) ocorre de forma paralela. Assim que a atualização de todas as partículas esteja finalizada, o processador mestre delega as tarefas aos processadores disponíveis (N_p), também chamados de processadores escravos. Neste contexto, uma tarefa consiste em avaliar a posição da partícula atribuindo-a uma nota, no algoritmo PSO esta avaliação se dá pelo cálculo da função objetivo do problema de otimização ($f(x)$).

Conforme citado, a paralelização com o comando *PARFOR* possibilita, ao processador mestre controlar a carga de trabalho dos N_p. Isso só é possível graças à independência entre o tempo de resposta dos N_p, ou seja, assim que concluído o trabalho em um processador escravo, este pode solicitar nova tarefa ao mestre, independente dos demais processadores terem finalizados suas tarefas.

O pseudocódigo_4 apresenta a implementação, de forma paralela, da avaliação das posições das partículas, dada pela função objetivo $f(x_i)$.

Pseudocódigo_4: Avaliação da Função Objetivo

Para cada partícula *i*, de forma paralela faça

Avaliação da função$f(x_i)$ em N_p processadores

Fim_*Para*

Após as posições de todas as partículas serem avaliadas, realiza-se a atualização dos ótimos locais e globais no processador mestre. Para esta etapa, mantêm-se a forma sequencial do algoritmo, como pode ser verificada no pseudocódigo_5.

Pseudocódigo_5: Verificação Quanto ao Ótimo Local e Global

Para cada partícula *i* faça

Se ($f(x_i) < lbest_i$)

Então $lbest_i = f(x_i)$ e $xlbest_i = xnew_i$ respectivamente

Fim_*Se*

Se($f(x_i) < gbest_k$)

Então $gbest_k = f(x_i)$ e $xgbest = xnew_i$ respectivamente

Fim_*Se*

Fim_*Para*

Na etapa final do código é implementado o critério de convergência e o critério de parada para o algoritmo. O critério de convergência consiste em satisfazer duas condições concomitantemente, uma delas, a tolerância quanto à covariância (uma medida de dispersão) pode ser definida como a razão entre o desvio padrão e a média referente ao valor da função de custo para todas as partículas, a qual deve ser menor ou igual a um valor de tolerância especificada (tol_{COV}). A outra condição consiste em estabelecer uma tolerância para a violação das restrições, a qual não comprometa a validação da solução do projeto, esta condição recebe o nome de tolerância para o fator de penalização (tol_{FP}). Atendendo as duas condições simultaneamente, o algoritmo é considerado convergido, do contrário, o contador *k* (número de iterações) é incrementado e o algoritmo volta a iterar a partir dos últimos resultados obtidos.

Quanto ao critério de parada, como o próprio nome diz, é possível parar o algoritmo limitando-o a um número máximo de iterações k_{max}, quando atingido, independente de satisfeito ou não o critério de convergência citado no parágrafo anterior, o algoritmo é finalizado. Em outras palavras, o algoritmo pode parar atingindo um número total de iterações k_{max} sem atender o critério de convergência ou vice e versa.

O pseudocódigo_6 apresenta a implementação desta etapa, realizada de forma serial no processador mestre.

Pseudocódigo_6: Condição de parada para o laço de otimização do PSPSO

Se ($COV < tol_{COV}$) e ($FP < tol_{FP}$)

Então $Finaliza\ o\ algoritmo$

Fim_*Se*

Se $k = k_{max}$

Então $Finaliza\ o\ algoritmo$

Fim_*Se*

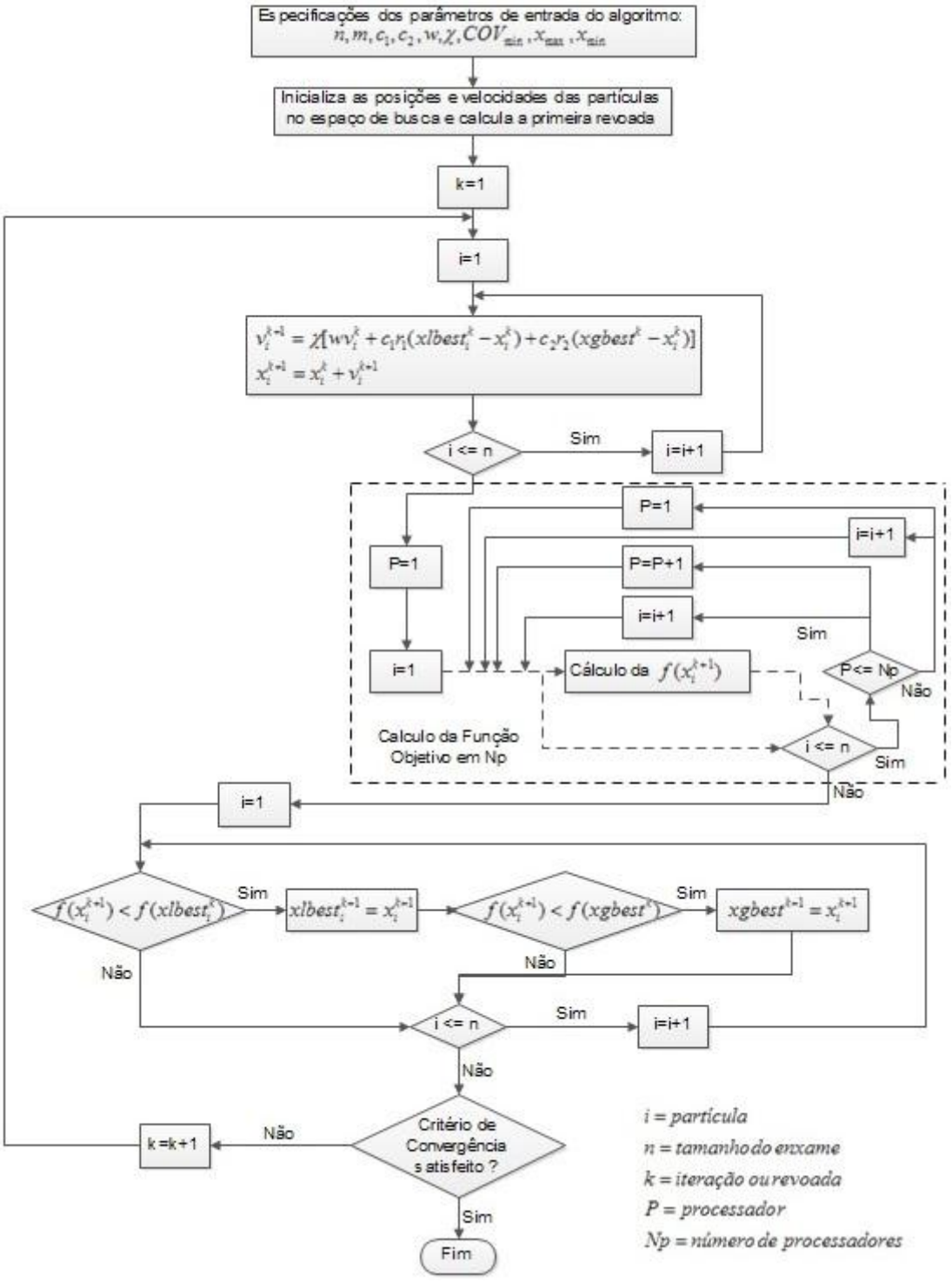


Figura 3. 5 – Fluxograma da implementação do PSPSO

De um modo geral, as tarefas desempenhadas pelos processadores mestre/escravos, as quais foram apresentadas nos pseudocódigos_1 a 6 bem como no fluxograma da Figura 3.5, podem ser organizadas da seguinte forma:

- Processador Mestre

1. Inicializa todos os parâmetros de otimização, posições e velocidades iniciais das partículas;
2. Atualiza posição e velocidade das partículas;
3. Mantém uma fila de partículas (ou seja: posição das partículas) para os processadores escravos avaliarem;
4. Envia a posição da partícula seguinte da fila para um processador escravo disponível;
5. Recebe o valor da função *fitness* e o guarda numa fila até que todas as partículas sejam avaliadas;
6. Atualiza *xlbest*, *xgbest* e seleciona a melhor partícula do enxame;
7. Verifica convergência.

- Processadores Escravos

1. Recebe a posição da partícula do processador mestre;
2. Analisa a posição da partícula com a função *fitness*;
3. Envia o valor da *fitness* para o processador mestre.

3.2.2 Detalhamento do Código Paralelo Assíncrono do PSO (PAPSO)

Conforme dito na seção anterior, o comando *PARFOR* possibilita ao PSO síncrono a independência entre os processadores escravos durante a avaliação da função objetivo de cada partícula. No entanto, o processador mestre deve esperar o calculo da função objetivo de todas as partículas para que ele possa prosseguir com os demais cálculos de atualização dos ótimos globais e da velocidade e posição das partículas. Esta característica torna evidente um ponto fraco do método, que é a dependência entre as etapas do algoritmo, ou seja, o processador mestre só pode prosseguir para uma nova etapa após receber todas as informações dos N_p.

Sendo assim, de um modo geral, apesar do comando *PARFOR* possibilitar ao PSO síncrono a independência entre os N_p, na avaliação das $n_{partículas}$ em paralelo (cálculo de função objetivo), as demais etapas que compõem o algoritmo serial continuam dependentes umas das outras. E conforme descrito na subseção 2.10.1, o não cumprimento de algumas condições quanto à utilização deste método, tendem a gerar desbalanceamento de carga, degradando o desempenho desses algoritmos.

Para contornar esses problemas, foi implementado o código paralelo assíncrono (PAPSO) utilizando as topologias global e mestre escravo para a comunicação entre os processadores em formato de estrela. Diferente do PSPSO, na abordagem paralela assíncrona o processador mestre é responsável apenas pela verificação quanto a convergência do método, as demais etapas do processo ocorrem isoladamente em cada processador. As tarefas desempenhadas pelos processadores mestre/escravos no algoritmo PAPSO podem ser organizadas da seguinte forma:

- Processador Mestre

1. Inicializa todos os parâmetros de otimização, posições e velocidades iniciais das partículas;
2. Envia uma fila de partículas para os processadores escravos;
3. Recebe as informações da melhor partícula do enxame;
4. Verifica convergência.

- Processadores Escravos

1. Cada processador escravo recebe e mantém uma parcela da fila de partículas para serem avaliadas;
2. Atualiza posição e velocidade da partícula com base em informações atualmente disponíveis (*xlbest*, *xgbest*);
3. Analisa a posição da partícula com a função *fitness*;
4. Realiza a troca de informações entre os demais processadores;
5. Atualiza *xlbest*, *xgbest* e seleciona a melhor partícula do enxame;
6. Mantém uma fila com as melhores partículas do enxame até que todas as partículas sejam avaliadas;
7. Envia as informações da melhor partícula do enxame.

Outro cuidado quanto à implementação paralela, consistiu em manter a característica de *pseudo-revoada* do algoritmo serial assíncrono, pois assim que as partículas, presentes nos N_p, receberem sua nota, esta é repassada (pela topologia de comunicação global) entre os N_p, os quais selecionam (isoladamente) a melhor partícula daquela rodada.

Em termos de implementação do código, o PAPSO difere do PSPSO apenas no laço responsável pela execução da otimização, onde é atribuído o comando *SPMD* do *MATLAB*®, para realizar a comunicação assíncrona entre os processadores. Para auxiliar no entendimento

da implementação do código, na Figura 3.7 é apresentado o fluxograma do PAPSO, correspondente a etapa responsável pela execução da otimização.

Internamente ao laço *SPMD*, são aplicadas as funções necessárias para a sua implementação, apresentadas na subseção 2.8.1 e tratadas, a partir de agora, ao contexto da presente pesquisa.

A primeira consideração a ser feita quanto à paralelização com o *SPMD*, consiste em distribuir as $n_{partículas}$ entre os processadores disponíveis. Isso é atingido dividindo as $n_{partículas}$ entre os N_p, alocando uma quantia de partículas igual para cada processador. Este esquema permite que a estrutura original do algoritmo sequencial assíncrono seja mantida, salvo que, a ordem de execução das tarefas (ou ordem de avaliação das partículas) é alterada a cada *pseudo-revoada*, ou seja, assim que os N_p concluírem o primeiro lote de partículas, é estabelecida uma nova sequência de distribuição das partículas entre os N_p.

Assim que os N_p estejam em posse de suas partículas, cada qual, realiza a avaliação da sua respectiva partícula com a função de aptidão *fitness* bem como a atualização dos ótimos locais e globais.

Ao término desta etapa, o *SPMD* exige que seja implementado ao código a função *labBarrier*, que fará uma barreira durante a execução do algoritmo até que os N_p terminem seus trabalhos para que a etapa posterior (comunicação) se realize. As barreiras de sincronização são geralmente implementadas quando o valor de uma dada variável calculada por um determinado processador lógico é necessária para o processamento dos dados de um ou de todos os outros processadores envolvidos [Masuero, 2009]. A função *labBarrier* evita que ocorram problemas de *deadlock* (tratados na subseção 2.8.1), entretanto, sua utilização representa uma desvantagem associada ao desempenho do método, a qual será verificada na subseção 6.2.2.

No processo seguinte a função *labSendReceived* do *SPMD* permite a comunicação entre os N_p. Durante este processo, cada processador seleciona sua melhor partícula e a envia para cada um dos demais processadores, sendo assim, a cada *pseudo-revoada* todos os processadores estarão abrigando a melhor partícula do enxame. Supondo aplicar o algoritmo em um processo paralelo com a utilização de uma plataforma multiprocessada composta por 3 processadores, a troca de partículas entre os processadores é realizada em cada processador do primeiro ao terceiro. Na Figura 3.6 está demonstrado o exemplo supracitado com a utilização

da comunicação segundo a topologia em forma de estrela, semelhante ao encontrado em Masuero, 2009. Os processadores são representados por círculos que realizam a troca de partículas entre eles (representada pelas setas) em duas etapas. A etapa final consiste em todos estarem em posse da melhor partícula do enxame até aquele momento. O índice sobrescrito (n^p) na Figura 3.6, corresponde ao número do processador de origem da partícula e o índice subscrito (n_i), corresponde ao número da respectiva partícula.

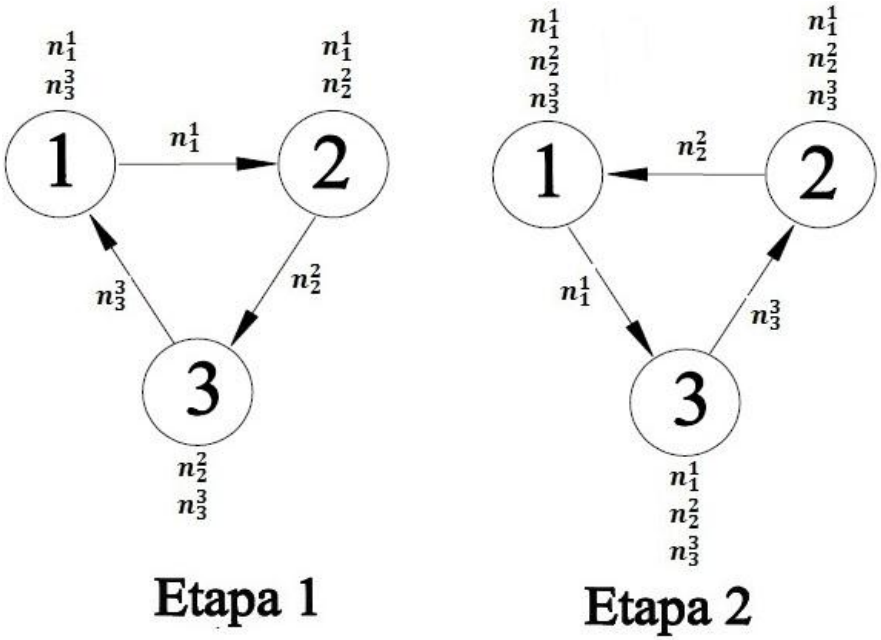


Figura 3. 6 – Comunicação entre 3 processadores e 2 etapas

O pseudocódigo_1 apresenta a etapa do algoritmo responsável por enviar uma nova ordem de retorno das partículas aos N_p bem como possibilitar que todos os processadores estejam a par do ótimo global. Em outras palavras, esse laço representa a implementação de uma *pseudo-revoada* no código proposto.

Pseudocódigo_1: Laço do *SPMD* Para a Pseudo-Revoada

Para cada partícula *i*, de forma paralela, faça

As partículas da *pseudo-revoada* são distribuídas entre os N_p

Para cada variável de projeto j, faça

Atualiza velocidade $vnew_{i,j}$ e posição $xnew_{i,j}$

Verifica *Se* a posição $xnew_{i,j}$ respeita as restrições $xmax_{i,j}$ e $xmin_{i,j}$

Fim_*Se*

Fim_*Para* (variáveis de projeto)

Cada N_p avalia sua $f(x_i)$

Se $(f(x_i) < lbest_i)$

Então $lbest_i = f(x_i)$ e $xlbest_i = xnew_i$ respectivamente

Fim_*Se*

Se$(f(x_i) < gbest_k)$

Então $gbest_k = f(x_i)$ e $xgbest = xnew_i$ respectivamente

Fim_*Se*

Barreira

Para os N_p, encontre a solução ótima $gbest_k$

Realiza-se a comunicação (troca de soluções das partículas) entre os N_p

Se $(gbest_{processador\ atual} < gbest_{processador\ vizinho})$

Então $gbest_k$=$gbest_{processador\ atual}$

Fim_*Se*

Cada processador envia sua melhor partícula para o restante do grupo

Fim *Para* (Os N_p conhecem a solução ótima)

Fim_*Para*

Para a etapa final do código, onde o laço de otimização do algoritmo é interrompido, utiliza-se a condição de parada citada na seção anterior e apresentada no pseudocódigo_6. Essa parte, como no algoritmo do PSPSO, ocorre no processador mestre de forma sequencial, forra do laço *SPMD*.

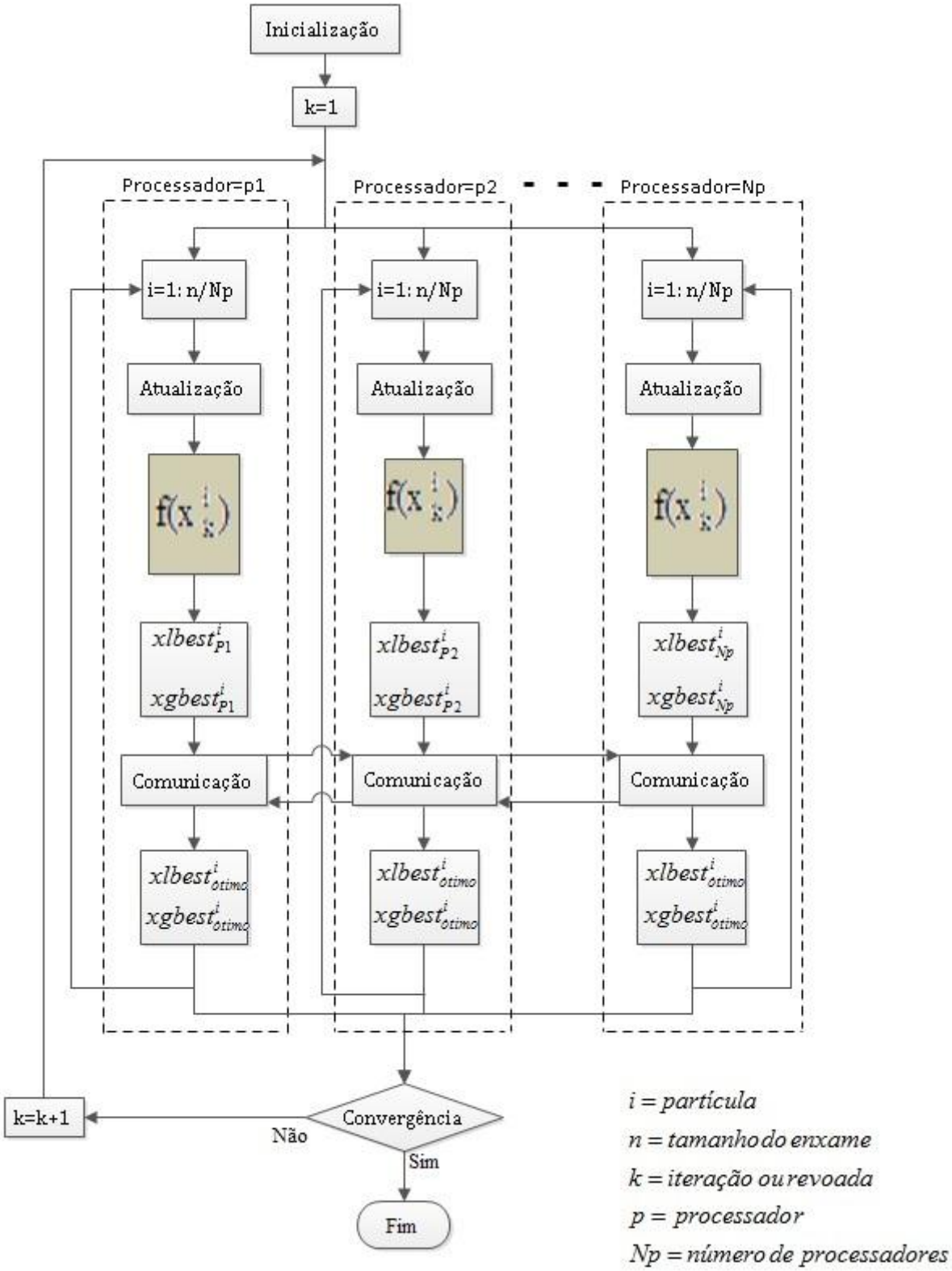


Figura 3. 7 – Fluxograma da implementação do PAPSO

Considerando a ordem de atualização das partículas verificada nos algoritmos paralelos desenvolvidos neste trabalho (PSPSO e PAPSO), é possível perceber que a implementação paralela aplicada às versões seriais deve afetar o resultado final obtido (verificar as soluções que serão apresentadas na seção 6.2), mesmo quando uma mesma semente de números aleatórios for utilizada. A justificativa para a não coerência entre as soluções obtidas com o algoritmo serial e paralelo pode ser tratada da seguinte forma: 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, sendo assim, elas podem tomar caminhos diferentes, a depender da ordem de retorno das partículas pelos escravos.

3.3 FUNCIONAMENTO DO PROGRAMA PARA A OTIMIZAÇÃO DE ESTRUTURAS METÁLICAS

Para a atual seção, procurou-se trazer o conceito de fenômeno (enxame de partículas) do PSO, dado aos programas desenvolvidos, ao contexto da otimização de estruturas treliçadas de aço, a fim de conduzir o entendimento do algoritmo PSO segundo uma abordagem prática na mecânica estrutural. Para facilitar o entendimento, o fluxograma da Figura 3.8 apresenta, de forma simplificada, a rotina desenvolvida. Algumas considerações são feitas no fluxograma:

- Função objetivo: minimização da massa total da estrutura (treliça);
- Método de otimização tratado: otimização paramétrica (alteração na dimensão da área da seção transversal das barras (A));
- Variáveis de projeto: grupos de barras da treliça;

Cabe ressaltar, que a rotina detalhada aqui diz respeito ao SSPSO, entretanto, a analogia adotada (por exemplo: partícula = treliça), pode ser estendida aos demais métodos propostos no presente trabalho.

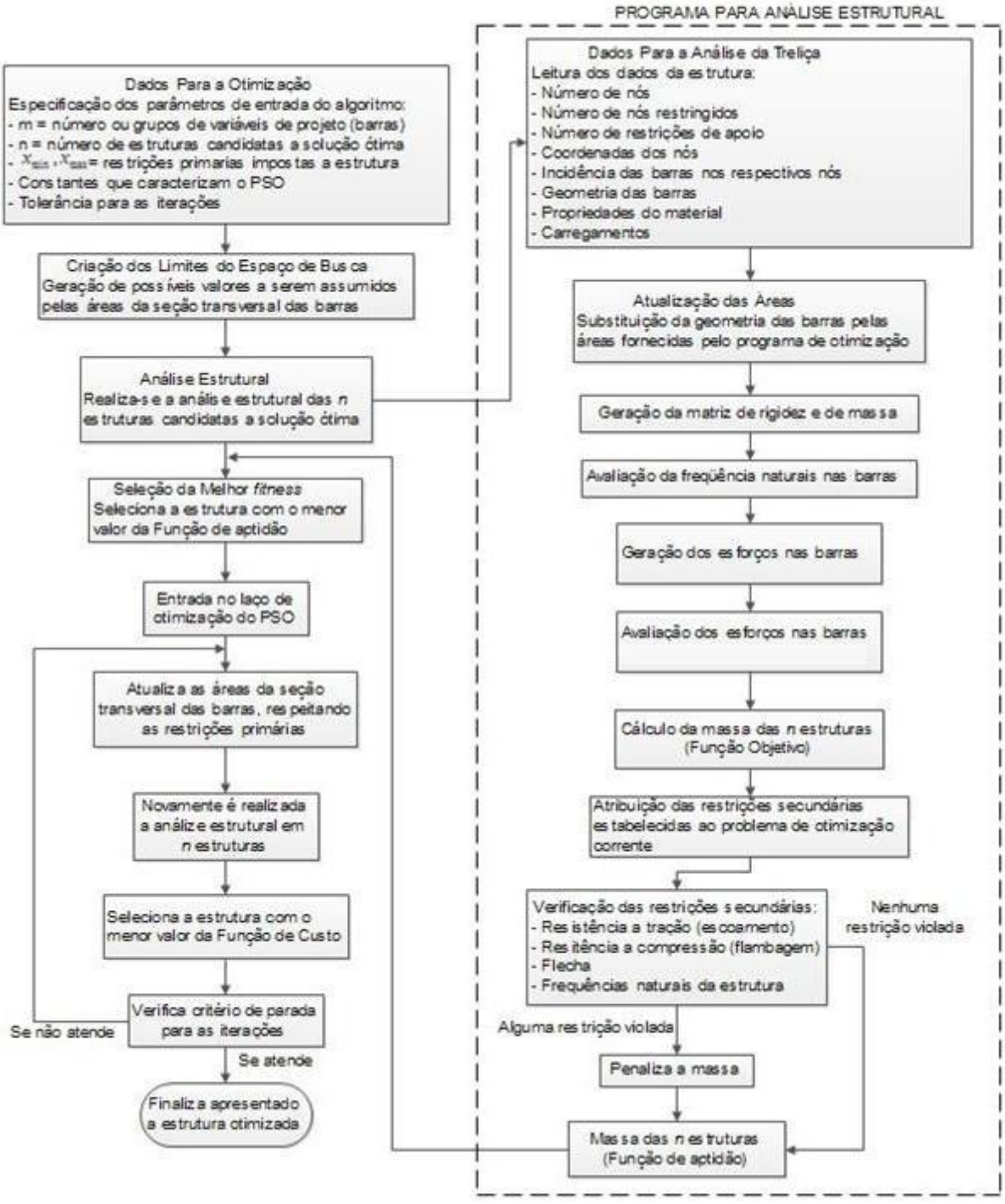


Figura 3. 8 – Fluxograma da rotina desenvolvida nos algoritmos

3.3.1 Dados Relativos à Otimização

A partir das disponibilidades de perfis e da configuração geométrica da treliça, inicialmente, são adotados os seguintes dados ao programa:

a. Número de grupos de variáveis: Caso não houver agrupamento de variáveis de projeto, ou seja, caso não se imponha que determinadas barras apresentem os mesmos valores de área para a seção transversal, o número de grupos será igual ao número de barras;
b. Número de estruturas candidatas à solução ótima: Esse número é escolhido de acordo com o problema, verificando-se qual o valor apropriado para se ter uma convergência satisfatória;
c. Restrições primárias: Também chamadas de restrições laterais, são os limites para o espaço de busca. Relacionam se diretamente com as variáveis de projeto limitando seus valores a um intervalo fixo, ou seja, uma restrição lateral é aplicada sobre a solução atual com o propósito de impedir que a otimização forneça valores numéricos “absurdos” (por exemplo, valores negativos para áreas da seção transversal das barras) ou inadequados.
d. Constantes do PSO: As constantes que caracterizam o PSO são as constantes apresentadas na subseção 2.4.1;
e. Tolerância para as iterações: A tolerância para as iterações trata-se de uma condição de parada do programa, discutida na subseção 3.2.1.

3.3.2 Análise Estrutural

Na etapa inicial da análise estrutural, realiza-se a leitura dos dados geométricos, carregamentos e propriedades do material. No programa desenvolvido, essas informações são fornecidas através de um arquivo de entrada de dados, como mostra a Figura 3.9.

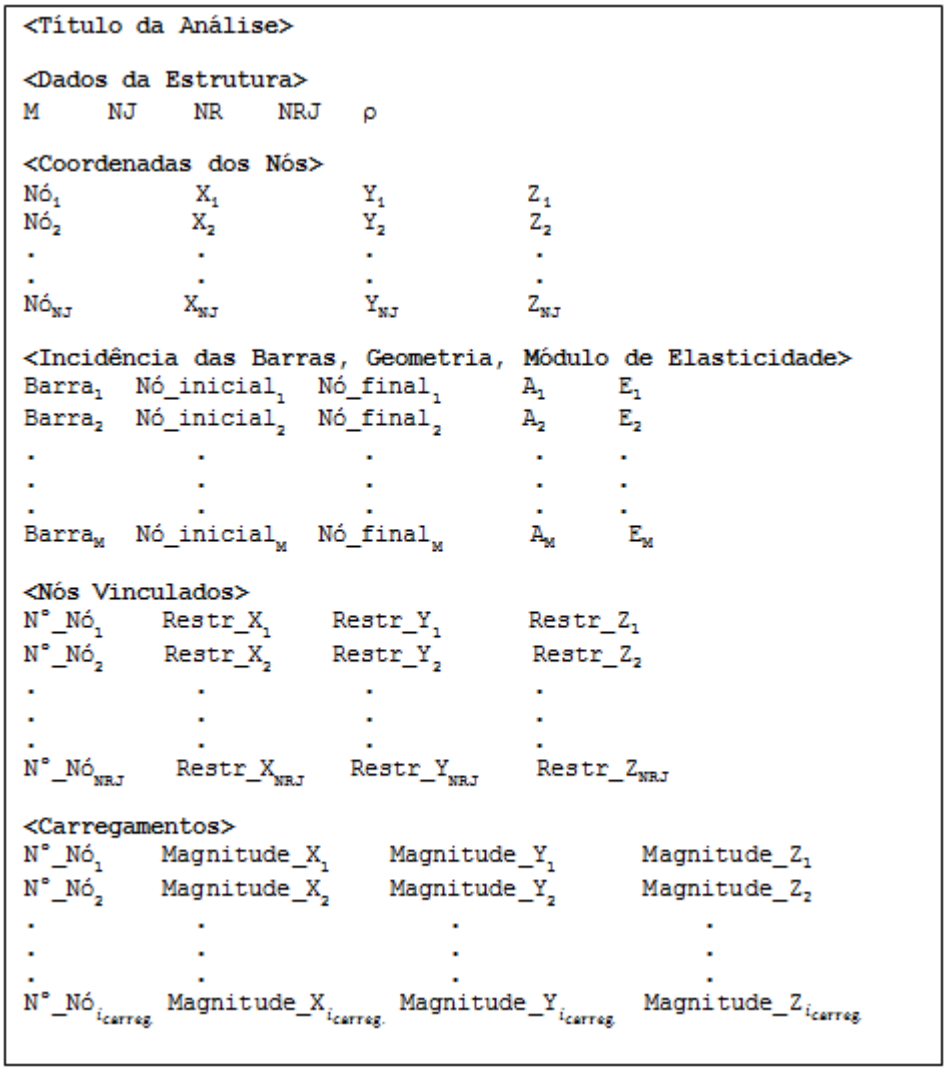


Figura 3. 9 – Entrada de dados para a análise estrutural

Fonte: Adaptado de Gomes, 2003.

Onde:

- M = Número de barras da estrutura;
- NJ = Número de nós da estrutura;
- NR = Número de deslocamentos restringidos: Corresponde ao número de graus de liberdade restringidos;
- NRJ = Número de nós com algum tipo de vínculo;
- ρ = Densidade do material;
- E = Módulo de Elasticidade do Material;
- A = Área da seção transversal da barra.

A execução da análise estrutural, após a leitura dos dados da treliça (e substituição das áreas, cadastradas no banco de dados da geometria da treliça, pelas fornecidas pelo programa de otimização), é realizada pelo Método de Elementos Finitos (MEF).

Uma visão completa do MEF pode ser encontrada em Hughes, 1987 e Zienkiewicz e Taylor, 2000, não sendo de interesse deste trabalho uma descrição detalhada do método. Entretanto, para o entendimento da metodologia aplicada na programação desenvolvida no presente trabalho utilizando o MEF, uma breve explicação será dada ao método no parágrafo seguinte.

O MEF é responsável por resolver o sistema linear composto pela matriz de rigidez e pelos vetores de força e deslocamento da estrutura. Uma vez determinado os vetores de deslocamentos nodais, é possível determinar as deformações e esforços axiais correspondentes nas barras, bem como as reações nodais. Para os casos onde a frequência natural da estrutura deva ser avaliada, um problema de autovalores e autovetores é formulado com as matrizes globais de massa e rigidez, obtendo-se assim, as frequências naturais não amortecidas da estrutura.

Neste trabalho, as matrizes de rigidez de barras tridimensionais com elementos finitos com função de interpolação linear são utilizadas. Da mesma maneira, em problemas modais, a matriz de massa consistente usual de treliças tridimensionais correspondentes é empregada nas análises efetuadas.

A execução da análise matricial, detalhada no parágrafo anterior, fornece uma série de dados necessários para determinar se as variáveis de projeto atendem as restrições secundárias (restrições de igualdade, desigualdade e de comportamento), a fim de garantir a integridade da estrutura, evitando que a mesma entre em colapso.

Para o modelo em estudo (barras), a possibilidade de falha da estrutura pode ocorrer devido aos esforços de tração, compressão (flambagem) ou frequência natural. A combinação destas ações deve respeitar as restrições do problema para que os critérios de falha sejam satisfeitos, tais como: condições específicas para força axial de tração e compressão, deslocamentos máximos, limites de esbeltez, frequência e etc..

Resolvendo e verificando a estrutura, caso algum dos critérios não seja atendido, uma vez encontrado o valor da função objetivo, este será tratado pelo método da penalização, onde um fator de penalização (FP) é adicionado à solução corrente.

Essa penalização é atribuída à função objetivo, de forma a aumentá-la quando alguma restrição for violada, ou seja, a função objetivo deverá ser tanto mais penalizada (aumentada), quanto maior for a violação de alguma restrição. A Equação 3.1, apresenta a formulação dada à função penalizada.

$$f_p(x,c) = f(x) + c * FP \tag{3.1}$$
$$FP_{pr+1} = FP_{pr} + pr_j$$

Onde:

$f_p(x,c)$ = função penalizada ou função de aptidão;

$f(x)$ = função objetivo;

c = coeficiente de penalização;

FP_{pr} = fator de penalização atual;

pr_j = valor associado à violação da restrição da barra *j*;

FP_{pr+1} = fator de penalização acumulado, refere-se ao somatório das restrições violadas.

Tratando a Equação 3.1 ao contexto da minimização da massa de uma estrutura em aço, a função de custo pode ser representada pela Equação 3.2.

$$Massa_p{}^* = \rho \sum_{j=1}^{M} A_j L_j + c * FP \tag{3.2}$$

Onde:

$Massa_p{}^*$ = função penalizada;

ρ = densidade do material;

A_j = área da seção transversal da barra *j*;

L_j = comprimento da barra *j*;

Pelo equacionamento realizado na determinação da função penalizada, fica evidente a seguinte condição: quando $FP \neq 0$, alguma restrição foi violada, do contrário, quando $FP = 0$, nenhuma restrição foi violada, e a função penalizada assume a forma indicada na Equação 3.3.

$$Massa_p{}^* = \rho \sum_{j=1}^{M} A_j L_j = Massa \tag{3.3}$$

Uma vez realizada a análise das *n* estruturas, é passada para a seleção da estrutura que apresentar a menor função de custo, ou seja, a treliça que apresentar a menor massa penalizada. Esse valor é definido como o ótimo global da função objetivo do problema. Assim, definidos esses valores iniciais para o melhor resultado da função de custo, a estrutura entra no laço de otimização.

3.3.3 Laço Para Otimização da Estrutura

Inicialmente à entrada no laço para a otimização da estrutura, é realizada a atualização dos valores que foram atribuídos as variáveis de projeto (área da seção transversal das barras) das *n* estruturas antes da análise estrutural. Essa atualização leva em consideração as Equações 2.5 e 2.2 respectivamente, as quais serão novamente apresentadas aqui, para melhor entendê-las no contexto da mecânica estrutural.

$$v_{i,j}(t+1) = \chi\left(wv_{i,j}(t) + c_1 r_1\left(xlbest_{i,j}(t) - x_{i,j}(t)\right) + c_2 r_2\left(xgbest_j(t) - x_{i,j}(t)\right)\right) \quad (2.5)$$

$$x_{i,j}(t+1) = x_{i,j}(t) + v_{i,j}(t+1) \quad (2.2)$$

Onde:

- $x_{i,j}(t)$: é a atual área da treliça *i* correspondente a variável de projeto (barra) *j*;
- $x_{i,j}(t+1)$: é a área atualizada da treliça *i* correspondente a variável de projeto (barra) *j*;
- $xlbest_{i,j}(t)$: é a melhor área já encontrada pela treliça *i* correspondente a variável de projeto (barra) *j*;
- $xgbest_j(t)$: é a melhor área já encontrada dentre as *n* treliças, correspondente a variável de projeto (barra) *j*, esse parâmetro foi fornecido pela análise estrutural anterior;

Os demais parâmetros existentes na equação são melhor compreendidos ao contexto do enxame de partículas, detalhados na seção 2.4.

Em seguida, as novas áreas das *M* barras nas *n* estruturas são verificadas quanto às restrições primárias (restrições laterais). Em caso da violação destas, suas áreas são corrigidas para os respectivos valores mínimos ou máximos das restrições laterais.

A etapa subsequente consiste novamente na realização da análise estrutural das *n* estruturas, conforme apresentada na subseção 3.3.2, e a posterior seleção da treliça que possuir o menor valor da função custo.

Por fim, verifica-se por meio de um critério de parada, se o processo será finalizado ou não. Caso não satisfeito, um novo processo de otimização é realizado, do contrário, o algoritmo se da por encerrado e o valor da função de custo (massa otimizada da treliça) e suas respectivas variáveis de projeto (áreas da seção transversal das *M* barras) são apresentados.

4 METODOLOGIA DE AVALIAÇÃO DOS ALGORITMOS

Neste capítulo, são apresentadas as definições necessárias ao entendimento de alguns dos métodos de avaliação existentes, os quais são desenvolvidos e adotados no presente trabalho.

4.1 ACURÁCIA, ROBUSTEZ E DESEMPENHO DOS ALGORITMOS

A *acurácia* (μ) de um determinado algoritmo em termos da posição, pode ser definida como a média das distâncias obtida entre as posições encontradas pelo algoritmo (após satisfeita a condição de parada e para diversas reinicializações com sementes aleatórias diferentes) e a posição ótima conhecida do problema. Essa métrica é implementada no algoritmo de acordo com a Equação 4.1.

$$\mu = \frac{1}{R}\sum_{i=1}^{R}\left\|\overrightarrow{xgbest_i} - \vec{x}^*\right\|$$

$$= \frac{1}{R}\sum_{i=1}^{R}\sqrt{\left(xgbest_{i,1} - {x_1}^*\right)^2 + \left(xgbest_{i,2} - {x_2}^*\right)^2 + \cdots + \left(xgbest_{i,m} - {x_m}^*\right)^2} = \tag{4.1}$$

$$\frac{1}{R}\sum_{i=1}^{R}\sqrt{\sum_{j=1}^{m}\left(xgbest_{i,j} - {x_j}^*\right)^2}$$

Onde x^* é a solução ótima conhecida para o problema, m representa o número de variáveis de projeto e *R* o número de reinicializações.

Para quantificar a *acurácia* do algoritmo PSO, se faz necessário o conhecimento do ponto ótimo global. Em funções simples, que permitam serem resolvidas analiticamente, como é o caso das funções de testes padrão utilizadas a seguir, a posição do ótimo global é de fato conhecida no intervalo de busca utilizado. No entanto, este método apenas pode ser utilizado em funções que representam problemas simples, já no caso de problemas mais complexos, um método que possibilita encontrar o ponto ótimo global, pode ser descrito da seguinte forma: inicializar o enxame várias vezes e definir que o melhor resultado encontrado entre todas as vezes que o enxame foi inicializado seja o ponto ótimo global. Quanto menor o

valor encontrado para a *acurácia*, melhor será o algoritmo, pois mais próximo do valor verdadeiro ótimo estará ele.

A *robustez* ($\boldsymbol{\sigma}$) relaciona quantas vezes o algoritmo reinicializou a procura e quantas destas vezes ele atingiu o mesmo local, que não necessariamente é o ótimo global, sob condições diferentes de reinicialização. Ele representa o quão robusto é o algoritmo ao iniciar em posições diferentes do espaço de procura e alcançar o mesmo resultado. Vale ressaltar que para cada reinicialização (*R*) do algoritmo, uma semente diferente de números aleatórios é utilizada para iniciar o enxame. Esse parâmetro pode ser tratado como sendo o desvio padrão da média dos ótimos globais obtido em todas as vezes que o algoritmo foi reinicializado. A Equação 4.2 apresenta a formulação adotada para a obtenção da *robustez*.

$$\left\|\overrightarrow{xgbest_i} - \vec{\mu}_{xgbest}\right\| = \sqrt{\sum_{i=1}^{R} (xgbest_i - \mu_{xgbest})^2} \tag{4.1}$$

$$\boldsymbol{\sigma} = \sqrt{\frac{\sum_{i=1}^{R}\left(\left\|\overrightarrow{xgbest_i} - \vec{\mu}_{xgbest}\right\|\right)^2}{R-1}} = \sqrt{\frac{\sum_{i=1}^{R}\left(\sqrt{\sum_{j=1}^{m}\left(xgbest_{i,j} - \mu_{xgbest_j}\right)^2}\right)^2}{R-1}} = \sqrt{\frac{\sum_{i=1}^{R}(\sum_{j=1}^{m}\left(xgbest_{i,j} - \mu_{xgbest_j}\right)^2)}{R-1}} \tag{4.2}$$

Onde μ_{xgbest} representa a média das melhores posições do enxame (*xgbest*) para todas reinicialização (*R*). O cálculo de μ_{xgbest} é feito como indicado:

$$\mu_{xgbest_j} = \frac{1}{R}\sum_{i=1}^{R} xgbest_{i,j}$$

O *desempenho do algoritmo* pode ser definido como o número médio de avaliações da função objetivo necessário para resolver o problema. Sendo assim, quanto menor for o número de avaliações da função objetivo, melhor será o desempenho do algoritmo. Outra definição dada ao desempenho do algoritmo pode se considerada como sendo a velocidade de convergência que o algoritmo apresenta para chegar à solução do problema

4.2 MÉTRICAS PARA AVALIAÇÃO DO DESEMPENHO COMPUTACIONAL

Conforme citado na seção 2.7, um dos objetivos fundamentais da computação paralela consiste em possibilitar resolver problemas de larga escala, com maior desempenho do que o

processamento sequencial. Nesse contexto, o desempenho de um programa pode ser mensurado de diversas formas. Em Masuero, 2009, é possível encontrar algumas das métricas para avaliar o desempenho de implementações paralelas: *Speedup*, Tempo de Processamento, *Eficiência* e Fração Serial α definida pela métrica de Karp-Flatt, 1990. As métricas utilizadas para mensurar o desempenho das extensões paralelas desenvolvidas na presente pesquisa serão: Tempo de Execução (ou tempo de processamento), *Speedup* e *Eficiência*.

a) Tempo de execução: o tempo de execução de um programa na forma serial ou paralela é o tempo decorrido desde o primeiro processador iniciar a execução do programa até o último terminar. A Equação 4.3 determina o tempo de execução.

$$T_{exec} = T_{final} - T_{inicial} \tag{4.3}$$

b) *Speedup* ou Ganho: é a demonstração do ganho efetivo em termos de tempo de processamento que o algoritmo paralelo fornece sobre o algoritmo serial, ou seja, é a razão entre o tempo gasto pelo computador executando o algoritmo de forma serial (T_{ser}) e o tempo gasto pelo mesmo computador executando o algoritmo em paralelo usando *p* processadores (T_{Par}). Esta definição é representada pela equação 4.4.

$$Speedup = \frac{T_{ser}}{T_{par}} \tag{4.4}$$

c) *Eficiência*: é a razão entre o *Speedup* e o número de processadores (N_p) utilizados na execução do algoritmo. A eficiência demonstra a fração de tempo em que os processadores estão ativos. Pode ser expressa pela Equação 4.5. Em geral o comportamento da eficiência é crescer com o aumento do número de processadores, mas depois de certo número começar a decair em virtude do aumento das trocas de informações entre processadores.

$$Eficiência = \frac{Speedup}{N_p} \tag{4.5}$$

5 VALIDAÇÃO DA METODOLOGIA PROPOSTA

Foram realizadas algumas validações com os algoritmos desenvolvidos no presente trabalho em funções de teste padrão, também conhecidas como *benchmarks*, desafiando-os a encontrarem os extremos destas funções.

Devido ao baixo tempo de processamento gasto com a avaliação das funções, os algoritmos serão verificados apenas quanto à solução obtida, as demais métricas apresentadas no capítulo 4, serão tratadas na otimização das treliças de 18, 72 e 120 barras.

5.1 FUNÇÕES DE TESTE

Os testes foram executados utilizando-se funções multimodais (contém muitos ótimos locais e somente um ótimo global) e unimodais (contém apenas um único ótimo local como sendo o ótimo global) encontradas comumente na literatura, as quais estão apresentadas na Tabela 5.1.

Tabela 5. 1 – Funções de teste unimodais e multimodais

Nome	Função	Limites
Rosenbrock	$f_1(x_i) = 100(x_2 - x_1)^2 + (1 - x_1)^2$	$-2.048 < x_i < 2.048$
Rastringin	$f_2(x_i) = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10)$	$-5.12 < x_i < 5.12$
Schwefel	$f_3(x_i) = 418{,}9829 + \sum_{i=1}^{n} x_i sen(\sqrt{\lvert x_i \rvert})$	$-500 < x_i < 500$

Fonte: Peer *et al.*, 2003

A função teste *Rosenbrock* possui um único ponto de mínimo localizado numa região que se apresenta na forma de um grande vale. Encontrar o vale é fácil, no entanto, a convergência para um ótimo global não é tão trivial, pois nesta região, a variação no valor da função é muito pequena. A Figura 5.1 ilustra o gráfico da $f_1(x_i)$ onde pode ser visto o seu vale. O único ponto mínimo esta localizado na posição $x^* = (1,1)$, onde a função apresenta um valor nulo $f_1(x^*) = 0$.

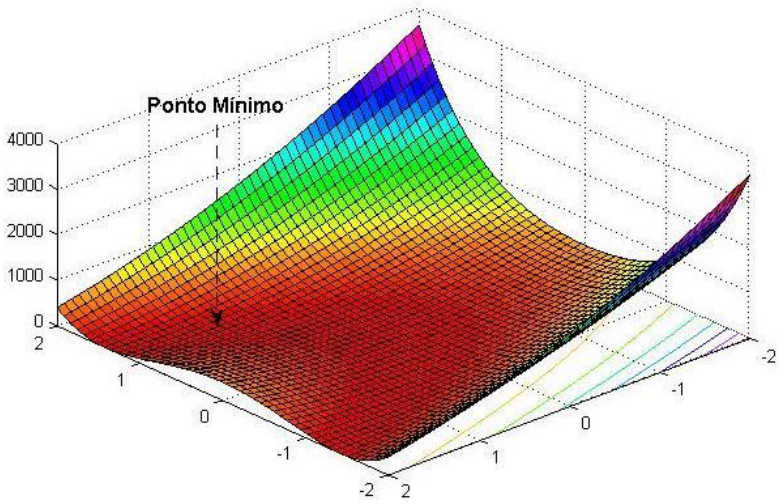


Figura 5. 1 – Gráfico da função *Rosenbrock*

A função *Rastrigin*, em duas dimensões apresenta-se com a forma de um paraboloide com inúmeros pontos de máximos e mínimos locais de menor amplitude ao longo desta superfície parabólica. A posição do mínimo global esta localizado na origem, $x^* = (0,0)$, onde o valor da função é nulo, $f_2(x^*) = 0$. A Figura 5.2 apresenta o gráfico da função.

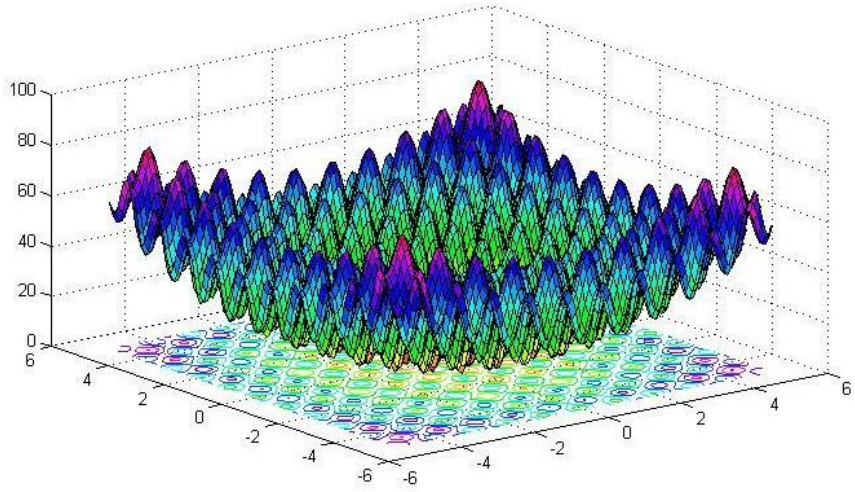


Figura 5. 2 – Gráfico da função *Rastrigin*

A função *Schwefel* apresenta diversos mínimos locais distribuídos ao longo do espaço de busca, sendo que o valor da função objetivo do mínimo global é muito próximo do valor da função objetivo de um segundo mínimo que se encontra bastante distante do global. A posição do mínimo global é $x^* = [-420{,}9687, -420{,}9687]^m$, cujo valor da função $f_3(x^*) = 0$. Essa função em um espaço de busca com dimensão *m* = 2 (2D), está representado na Figura 5.3.

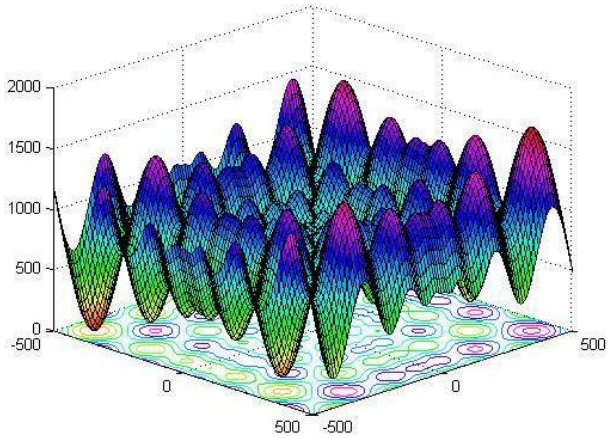


Figura 5. 3 – Gráfico da função *Schwefel*

5.2 EXPERIMENTOS E RESULTADOS

Para os testes preliminares realizados com os algoritmos propostos na otimização das funções apresentadas acima, foram utilizadas 60 partículas (n), dimensão igual a 30 (m), (exceto função *Rosenbrock*, m = 2), constantes de cognição (c_1 e c_2) iguais a 2.0. Para os demais parâmetros (inércia (w) e aleatoriedade (r_1 e r_2)), cada versão do algoritmo recebeu o valor correspondente a seu melhor desempenho.

Para fins de comparação, são realizadas 20 reinicializações com cada método, assim, os resultados são mensurados estatisticamente em valores médios. Essa técnica é adotada para evitar que a aleatoriedade favoreça um determinado método em particular, ou seja, considerando que uma mesma semente de números aleatórios seja atribuída aos diferentes métodos, um deles em específico será mais favorecido, gerando melhores resultados, entretanto, se essa semente for alterada, possivelmente outro método venha a ser mais favorecido que do caso anterior. Para cada reinicialização, é adotado como critério de parada dos algoritmos o total de 100 iterações (k_{max}=100).

A plataforma computacional é composta por um processador *quad core* (quatro núcleos), 298.09GB de HD, memória RAM de 8.0 GB, sistema operacional de 64 Bits e Windows Vista.

Com as tabelas 5.2 a 5.4, é possível comparar os valores encontrados pelos algoritmos desenvolvidos com aqueles fornecidos pela literatura (Peer *et al*., 2003, com o algoritmo PSO segundo uma topologia de vizinhança global - P_g), e verificar a proximidade entre eles.

Tabela 5. 2 – Resultados da otimização da função *Rosenbrock*

Método	$f_{média}(x^*)$	$f_{mín}(x^*)$	$f_{máx}(x^*)$
SSPSO	2.0173e-4	1.5012e-10	0.0022
SAPSO	3.8934e-5	2.769e-10	2.9515e-4
PSPSO	8.1722e-5	3.4555e-8	0.0010
PAPSO	1.3189e-4	6.6829e-9	0.0011
P_g(Peer *et al*., 2003)	5.70e-3	4.0e-4	4.3011

Tabela 5. 3 – Resultados da otimização da função *Rastrigin*

Método	$f_{média}(x^*)$	$f_{mín}(x^*)$	$f_{máx}(x^*)$
SSPSO	162.0642	86.1719	238.7533
SAPSO	112.150	64.348	189.862
PSPSO	175.2938	106.1098	269.4252
PAPSO	130.0373	66.1077	199.8236
P_g(Peer *et al*., 2003)	68.652	29.85	136.31

Tabela 5. 4 – Resultados da otimização da função *Schwefel*

Método	$f_{média}(x^*)$	$f_{mín}(x^*)$	$f_{máx}(x^*)$
SSPSO	4556.5	3349.6	6050.2
SAPSO	4463.6	3063.4	5610.7
PSPSO	4549.3	3440.8	5805.6
PAPSO	4284.3	3092.8	5843.8
P_g(Peer *et al*., 2003)	4510.8	2803.1	6179.5

* As soluções encontradas para SSPSO e PSPSO diferem entre si pelo fato dos algoritmos utilizarem sementes randômicas diferentes.

No geral, o assincronismo possibilitou aos algoritmos chegarem a soluções melhores que as obtidas com a versão síncrona, já os ganhos com o paralelismo nos algoritmos, não ficaram tão evidentes, variando de acordo com o problema. Quanto a solução das funções *Rosenbrock* e *Schwefel*, estas ficaram próximas as apresentadas por Peer *et al*., 2003, entretanto, para a função *Rastrigin*, os algoritmos não apresentaram soluções tão atraentes, mesmo assim, a versão assíncrona foi a que obteve os melhores resultados, dando indícios de que o assincronismo possa ser uma boa opção para a solução de problemas de otimização.

6 APLICAÇÃO DOS ALGORITMOS NA OTIMIZAÇÃO DE ESTRUTURAS METÁLICAS

Procurando aumentar a complexidade dos testes e aproximá-los aos casos de interesse da presente pesquisa, os métodos desenvolvidos são submetidos a problemas clássicos de otimização de estruturas treliçadas, permitindo assim, demonstrar a validade e desempenho da metodologia proposta na otimização estrutural. Esses exemplos têm sido estudados por diversos pesquisadores [Haftka e Gürdal, 1991; Rajeev e Krishnamoorthy, 1992; Kripka, 2004]. Nas treliças, a função objetivo consiste na minimização da massa respeitando as restrições estabelecidas para cada problema. As variáveis de projeto são tratadas como variáveis contínuas. Sendo assim, a formulação do problema para encontrar a massa mínima das treliças dos exemplos pode ser representada matematicamente pela Equação 6.1.

Minimizar: Função Objetivo: $$Massa(A) = \rho \sum_{j=1}^{M} A_j L_j \quad (6.1)$$

Variáveis de Projeto: A_j

Sujeito a: Restrições:

$$g_j(A) = \omega_j \leq \omega_j^{máx} \qquad j=1,2,..,M$$

$$g_j(A) = \sigma_j \leq \sigma_j^{T} \qquad j=1,2,...,M$$

$$g_j(A) = \sigma_j \leq \sigma_j^{F} \qquad j=1,2,..,M$$

$$A_j^{inf} \leq A_j \leq A_j^{Sup} \qquad j=1,2,...,M$$

$$x_q^{inf} \leq x_q \leq x_q^{Sup} \qquad q=1,2,..., n_{coord}$$

Onde g_j são as restrições secundárias (restrições de desigualdade), representadas nos problemas testes por: frequência natural da estrutura (ω_j) e tensão de escoamento atuando no material da barra (σ_j) quanto a tração (σ_j^T) ou flambagem (σ_j^F) por compressão. O $\omega_j^{máx}$ é o valor limite para os diferentes modos de frequência natural existentes na treliça, o qual deve ser respeitado pelo valor de frequência obtido com o programa de otimização (ω_j). As restrições laterais são representadas pelas limitações mínimas (A_j^{inf} x_q^{inf}) e máximas (A_j^{Sup}, x_q^{Sup}) impostas às variáveis de projeto, que correspondem área da seção transversal da barra (A_j) e as coordenadas de alguns nós (x_q). Resolvendo o problema estrutural, caso alguma restrição secundária não seja respeitada, uma vez encontrado o valor da função objetivo, este

será penalizado pelo fator de penalização (FP) na solução corrente. A equação (6.2) apresenta a formulação dada à função penalizada e as equações (6.3) e (6.4) representam, respectivamente, a magnitude da penalização quanto à restrição de frequência (pr_j^{ω}) e tensão (pr_j^{σ}) que forem violadas.

$$Massa(A)_p{}^{*} = \rho \sum_{j=1}^{mbarras} A_j L_j + c * FP, \qquad \text{onde: } FP_{pr+1} = FP_{pr} + pr_j \tag{6.2}$$

$$pr_j^{\omega} = \frac{\omega_j}{\omega_j^{máx}} - 1 \tag{6.3}$$

$$pr_j^{\sigma} = \frac{\sigma_j}{\sigma_j^{T}} - 1 \; ou \; \frac{|\sigma_j|}{\sigma_j^{F}} - 1 \tag{6.4}$$

Onde:

- Se $pr_j^{\sigma} \geq 0$: a restrição foi violada e considera-se o valor de pr_j^{σ} na penalização;
- Se $pr_j^{\sigma} < 0$: a restrição não foi violada, o FP_{pr+1} não é incrementado por pr_j^{σ}.

No caso da restrição de frequência pr_j^{ω}, as seguintes condições devem ser atendidas:

1. Quando se deseja que $\omega_j \leq \omega_j^{máx}$, então:
    - Se $pr_j^{\omega} \geq 0$: restrição violada, considerar pr_j^{ω} na penalização;
    - Se $pr_j^{\omega} < 0$: restrição não violada, FP_{pr+1} não é incrementado por pr_j^{σ}.
2. Quando se deseja que $\omega_j \geq \omega_j^{máx}$, então:
    - Se $pr_j^{\omega} < 0$: restrição violada, considerar pr_j^{ω} na penalização;
    - Se $pr_j^{\omega} \geq 0$: restrição não violada, FP_{pr+1} não é incrementado por pr_j^{σ}.

Para a otimização das treliças, foram utilizadas 60 partículas (n) e nas Tabelas 6.2, 6.3 e 6.5 estão dispostos os demais parâmetros do PSO, definidos de acordo com o melhor desempenho registrado pelo algoritmo perante o problema proposto. Quanto à plataforma computacional, esta é composta pelo processador AMD FX-8110 com 8 núcleos, 2.81GHz de frequência, memória RAM de 8.0 GB, sistema operacional de 64 Bits e Windows 7. Foram realizadas 40 reinicializações por algoritmo (otimizações para cada treliça), sob condições diferentes de inicialização, para calcular as métricas de avaliação dos algoritmos, as quais são mensuradas assim que o método tenha convergido. Cabe aqui ressaltar que as mesmas sementes (as quais são diferentes para cada reinicialização) são utilizadas para os quatro algoritmos desenvolvidos.

6.1 ESTRUTURAS METÁLICAS ANALISADAS

Nesta seção são apresentadas as treliças utilizadas para avaliação dos algoritmos desenvolvidos, são elas: treliça espacial de 72 barras, domo de 120 barras e treliça plana de 18 barras.

6.1.1 Problema da Treliça Espacial de 72 Barras

A estrutura treliçada espacial, apresentada na Figura 6.1, possui 72 barras e 20 nós, sendo que os nós localizados na extremidade superior (1, 2, 3, 4), possuem uma massa concentrada de 2270 kg. Neste problema há 16 variáveis a serem otimizadas para obter a massa ótima da estrutura, as quais correspondem à área da seção transversal das barras. Quanto às restrições, essas são relativas às frequências naturais da estrutura, onde $\omega_1 = 4\text{Hz}$ e $\omega_3 \geq 6\text{Hz}$. As áreas da seção transversal das barras são classificadas em grupos da seguinte forma: G1 – A_1 a A_4, G2 – A_5 a A_{12}, G3 – A_{13} a A_{16}, G4 – A_{17} a A_{18}, G5 – A_{19} a A_{22}, G6 – A_{23} a A_{30}, G7 – A_{31} a A_{34}, G8 – A_{35} a A_{36}, G9 – A_{37} a A_{40}, G10 – A_{41} a A_{48}, G11 – A_{49} a A_{52}, G12 – A_{53} a A_{54}, Q13 – A_{55} a A_{58}, G14 – A_{59} a A_{66}, G15 – A_{67} a A_{70}, G16 – A_{71} a A_{72}. As propriedades do material e da estrutura são listadas na Tabela 6.1. Um dos primeiros autores a analisar este problema foi Konzelman, 1986, com o método *Dual Method* (*DM*). Gomes, 2011, realizou a otimização da treliça de 72 barras com o algoritmo PSO, o qual apresenta uma estrutura algorítmica similar à utilizada na presente pesquisa, obtendo resultados muito próximos aos obtidos por Konzelman, 1986. Miguel e Fadel Miguel, 2012, utilizando a meta-heurística *Firefly Algorithm* (*FA*) também realizaram a otimização deste problema, obtendo resultados bastante competitivos. Para a presente pesquisa, realizou-se uma busca na literatura pela melhor solução encontrada até o momento para este problema, revelando que Sedaghati, 2005, utilizando *Force Method* (*FM*), obteve uma massa mínima de 327,605 kg, a qual é utilizada como referência para o trabalho. Seus respectivos valores de áreas das barras para a massa mínima são apresentados nas Tabelas 6.7 e 6.8.

Tabela 6. 1 – Propriedades do material e restrições da estrutura de 72 barras.

Propriedades	Valores	Unidades
E (Módulo de Elasticidade)	6,98 x 10^{10}	N/m^2
ρ (Densidade do material)	2770.0	kg/m^3
Massa concentrada	2270.0	Kg
Limite inferior das variáveis de projeto	0.6452 x 10^{-4}	m^2
Restrições de frequência	$\omega_1 = 4.0, \omega_2 \geq 6.0$	Hz

Tabela 6. 2 – Parâmetros utilizados para execução do PSO na estrutura de 72 barras.

$N°_{partículas}$	ω	C_1	C_1	tol_{COV}	tol_{FP}
60	0.62	2.03	2.03	10^{-3}	10^{-5}

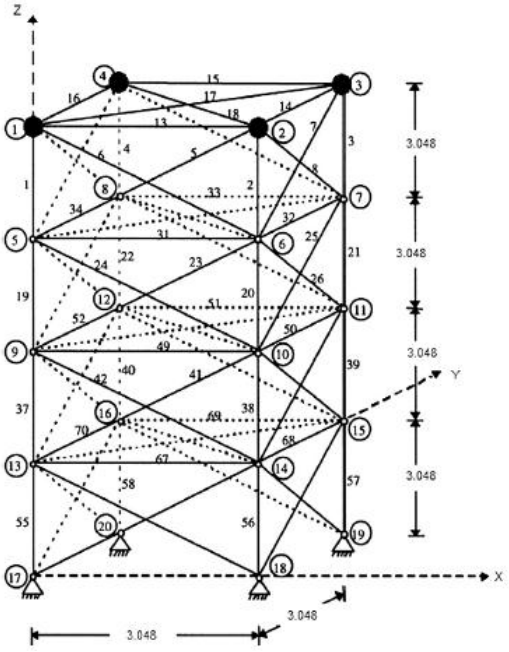


Figura 6. 1 – Estrutura espacial de 72 barras com massa concentrada (dimensão em m)

6.1.2 Problema da Treliça Plana de 18 Barras

A estrutura plana de 18 barras corresponde a um dos casos clássicos de otimização estrutural já analisado por diversos autores com diferentes métodos [Imai e Schmit, 1981, Felix, 1981, Yang, 1996, Soh e Yang, 1996, entre outros]. Como referência, será utilizado o artigo publicado por Lee e Geem, 2005, os quais utilizaram o algoritmo *Harmony Search* (HS) para chegar a uma massa mínima otimizada da estrutura de 2048,20 kg após 25 mil avaliações da função de custo. Os valores das variáveis de projeto para essa massa mínima

são apresentados na tabela 6.9. O presente trabalho resolve este problema usando o algoritmo por enxame de partículas na forma síncrona (SSPSO) e assíncrona (SAPSO), bem como suas respectivas versões seriais e paralelas (PSPSO e PAPSO), detalhados anteriormente. O problema corresponde a uma treliça composta por 18 barras e 11 nós, que esta sujeita à aplicação da carga concentrada P nos nós 1, 2, 4, 6 e 8, conforme ilustração da Figura 6.2, com a magnitude de 20 kip (88964 N). A densidade do material bem como módulo de elasticidade são, respectivamente, 0,1 lb/in³ (2768 kg/m³) e 10000 kip (68,95 GPa). A seção transversal inicial das barras adotada para este trabalho é de 10,25 in² (0,00661 m²), a qual pelo, processo de otimização, pode variar de forma contínua entre 3,5 in² (0,00226 m²) e 18 in² (0,01161 m²). Também, visando à minimização da massa, as coordenadas dos nós 3, 5, 7 e 9, podem ser alteradas. As áreas da seção transversal são designadas a grupos de barras, atendendo ao projeto da estrutura, da seguinte forma: G1 – $A_1 = A_4 = A_8 = A_{12} = A_{16}$, G2 – $A_2 = A_6 = A_{10} = A_{14} = A_{18}$, G3 – $A_3 = A_7 = A_{11} = A_{15}$ e G4 – $A_5 = A_9 = A_{13} = A_{17}$. Portanto, as variáveis a serem otimizadas correspondem às coordenadas dos quatro nós (cada nó pode variar as coordenadas em x e y) e os quatro grupos de barras, totalizando doze variáveis de projeto. São consideradas apenas restrições de tensão. Se a tensão normal atuante na barra (σ_j) for de tração, então ela não pode exceder a tensão de tração (σ_j^T), considerada para o presente problema como a tensão de escoamento do material (*Fy*) igual a 20 kpsi (137,90 MPa). Quanto à compressão, pela condição de flambagem, a tensão normal (σ_j) não pode exceder a tensão crítica de flambagem da barra, dada por Euler pela Equação 6.5.

$$\sigma_j^F = \frac{-kEA_j}{L_j^2}, \quad j = 1, \ldots, 18 \tag{6.5}$$

Onde *k* representa a constante determinada pela geometria da seção transversal da barra, como sendo, para o presente estudo, igual a 4, *E* corresponde ao módulo de elasticidade do material, A_j é a área da seção transversal da barra *j* e L_jé o comprimento da barra.

Tabela 6. 3 – Parâmetros utilizados para execução do PSO na estrutura de 18 barras.

$N°_{partículas}$	ω	C_1	C_1	tol_{COV}	tol_{FP}
60	0.5	1.5	1.5	10^{-3}	10^{-5}

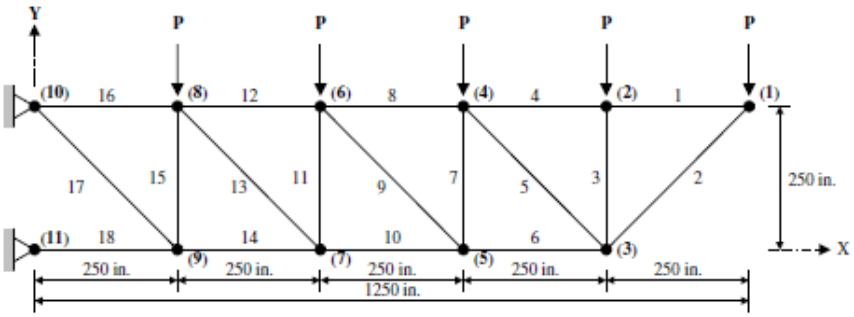


Figura 6. 2 – Estrutura plana de 18 barras com massa concentrada

Fonte: Lee e Geem, 2005.

6.1.3 Problema do Domo de 120 Barras

A estrutura espacial de 120 barras corresponde a uma estrutura composta por 49 nós em formato de domo conforme ilustração da Figura 6.3, sujeita a aplicação da carga concentrada P na direção vertical para baixo segundo posição e magnitude apresentados na Tabela 6.4, bem como as propriedades do material das barras e algumas das demais características do projeto original. As variáveis de projeto correspondem as seções transversais das barras, limitadas ao intervalo entre 0,0005 e 0,00323 m². Para o processo de otimização, agrupa-se a seção transversal das barras da seguinte forma: G1 – A_1 *a* A_{12}, G2 – A_{13} *a* A_{24}, G3 – A_{25} *a* A_{36}, G4 – A_{37} *a* A_{60}, G5 – A_{61} *a* A_{84}, G6 – A_{85} *a* A_{96}, G7 – A_{97} *a* A_{120}. Neste problema procura-se minimizar a massa da estrutura por meio da otimização paramétrica, respeitando as restrições quanto às variáveis de projeto bem como as restrições referentes às tensões nas barras, as quais podem atuar sob tração ou compressão. Similar ao problema da treliça de 18 barras, para tração tem-se que a tensão normal admissível na barra (σ_j) não pode exceder a tensão normal de tração (σ_j^T), que no atual problema equivale a 60% da tensão de escoamento do aço considerado (*Fy*=400 MPa). Sob compressão, a tensão admissível é tratada considerando flambagem segundo norma Americana AISC dada pela Equação 6.6.

$$\sigma_j^F = \frac{\left(1 - \frac{\lambda_j^2}{2C_c^2}\right) F_y}{\frac{5}{3} + \frac{3\lambda_j}{8C_c} - \frac{\lambda_j^3}{8C_c^3}} \quad Se\ \lambda_j < C_c \tag{6.6}$$

$$\sigma_j^F = \frac{12\pi^2 E}{23\lambda_j^2} \qquad Se\ \lambda_j \geq\ C_c$$

$$\lambda_j = \frac{KL_j}{r_j} \tag{6.7}$$

$$C_c = \sqrt{\frac{2\pi^2 E}{Fy}} \tag{6.8}$$

Onde *K* representa a constante determinada pela geometria da seção transversal da barra e suas constantes de fixação, que para o presente caso é assumido que *K*=12.0/23.0, *E* representa o módulo de elasticidade do material. O índice de esbeltez λ_j é obtido com a Equação 6.7, onde o raio de giração r_j depende do tipo de seção transversal das barras, no caso em estudo assume-se que as barras tenham o formato de um tubo, portanto $r_j = 0.4993A_j^{0.677}$, A_j é a área da seção transversal da barra j e L_jé o comprimento da barra. O índice de esbeltez que divide a flambagem em elástica ou inelástica C_c é dado pela Equação 6.8. Como referência para as variações do PSO propostas na presente pesquisa utilizaram-se os resultados encontrados por Lee e Geem, 2004, os quais utilizaram o algoritmo *Harmony Search* (*HS*) para chegar a uma massa mínima otimizada da estrutura de 8939,3 kg após 35 mil chamadas à função de custo.

Tabela 6. 4 – Propriedades do material e restrições para o domo de 120 barras.

Propriedades	Valores	Unidades
E (Módulo de Elasticidade)	21 x 10^{10}	N/m^2
ρ (Densidade do material)	7971,810	kg/m^3
Massa concentrada	2270.0	Kg
Área inicial da seção transversal das barras	2 x 10^{-4}	m^2
Limite inferior e superior das variáveis	5 x 10^{-4} e 32.3 x 10^{-4}	m^2
Carga – Nó	Magnitude	
P_1 – 1	60	KN
P_{2-13} – 2 a 13	30	KN
P_{14-37} – 14 a 37	10	KN

Tabela 6. 5 – Parâmetros utilizados para execução do PSO no domo de 120 barras.

$N°_{partículas}$	ω	C_1	C_1	tol_{COV}	tol_{FP}
60	0.5	1.5	1.5	10^{-3}	10^{-5}

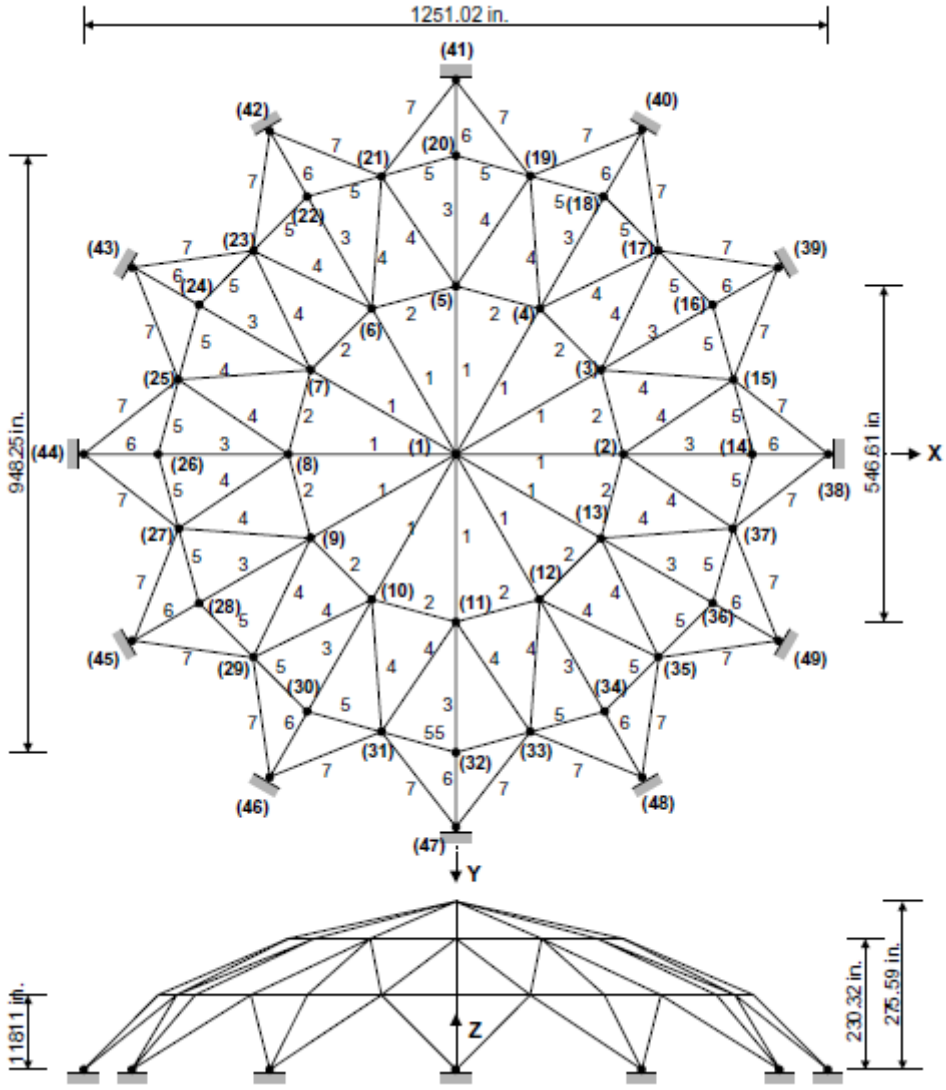


Figura 6. 3 – Estrutura em formato de domo com 120 barras

Fonte: Lee e Geem, 2004.

6.2 RESULTADOS E DISCUSSÕES

6.2.1 Quanto à Acurácia, Robustez e Desempenho do Algoritmo

A massa mínima otimizada da treliça de 72 barras obtida por Sedaghati, 2005, corresponde ao ponto ótimo pelo qual os algoritmos propostos tendem a encontrar. Sendo assim, na Tabela 6.6, estão dispostos os valores de massa média encontrados pelos algoritmos propostos bem como as respectivas métricas para sua avaliação.

Tabela 6. 6 – Massa otimizada e respectivas métricas para a treliça de 72 barras

	Método	$Massa_{média}$ [kg]	*Acurácia* [kg]	*Robustez* [kg]	$desemp._{médio}$
Presente Trabalho	SSPSO	332,041	4,436	9,112	6828
	SAPSO	332,635	5,03	9,450	4887
	PSPSO	332,041	4,436	9,112	6828
	PAPSO	331,945	4,34	9,310	2940
	Método	$Massa_{min}$ [kg]			
Sedaghati (2005)	*FM*	327,605			

Pela Tabela 6.6, pode-se verificar que aos valores apresentados para as diferentes métricas pelo algoritmo síncrono surgem em igual magnitude. Este comportamento aparece pelo fato da estrutura do algoritmo serial não possibilitar que com a implementação de sua extensão paralela possa haver alguma mudança, seja em termos de solução ótima ou quanto ao número de chamadas da função objetivo. Maiores detalhes que justificam tal comportamento foram apresentados na subseção 3.2.1.

Na estrutura otimizada pelos algoritmos assíncronos, apesar da similaridade com os algoritmos síncronos, quanto à massa, acurácia e robustez, os ganhos em termos de desempenho começam a ser significativos e se tratando do algoritmo assíncrono paralelo (PAPSO), é possível verificar ganhos ainda maiores em termos de solução ótima bem como desempenho. A Figura 6.4 complementa as informações supracitadas, na qual é possível perceber a robustez através da dispersão de valores para a massa de cada método ao longo das reinicializações.

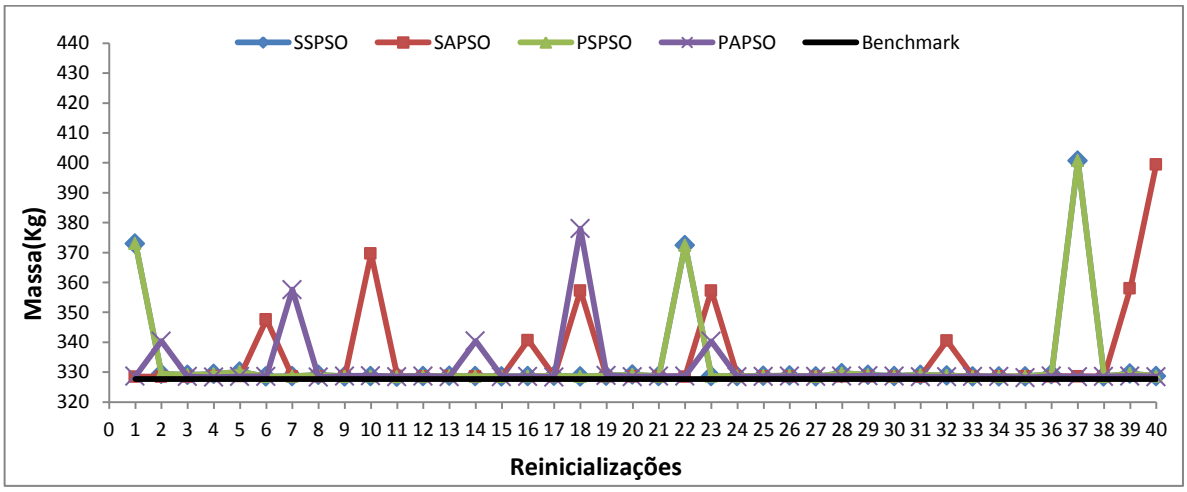


Figura 6. 4 – Gráfico da robustez e acurácia para treliça 72 barras

A acurácia esta representada no gráfico pelos pontos que representam a solução encontrada por cada método ao longo das reinicializações em relação à solução conhecida do problema (*benchmark* – Sedaghati, 2005). É importante ressaltar que garantidamente nenhuma restrição foi violada, em virtude da condição de parada do algoritmo a qual é atendida somente quando o algoritmo chega a uma solução pré-estabelecida sem violar nenhuma restrição (segundo tol_{FP}, Pseudocódigo_6 da seção 3.6.1). Quanto ao desempenho dos algoritmos, a Figura 6.5 contempla o comportamento de cada método na busca pela solução ótima (número de iterações) durante as 40 reinicializações.

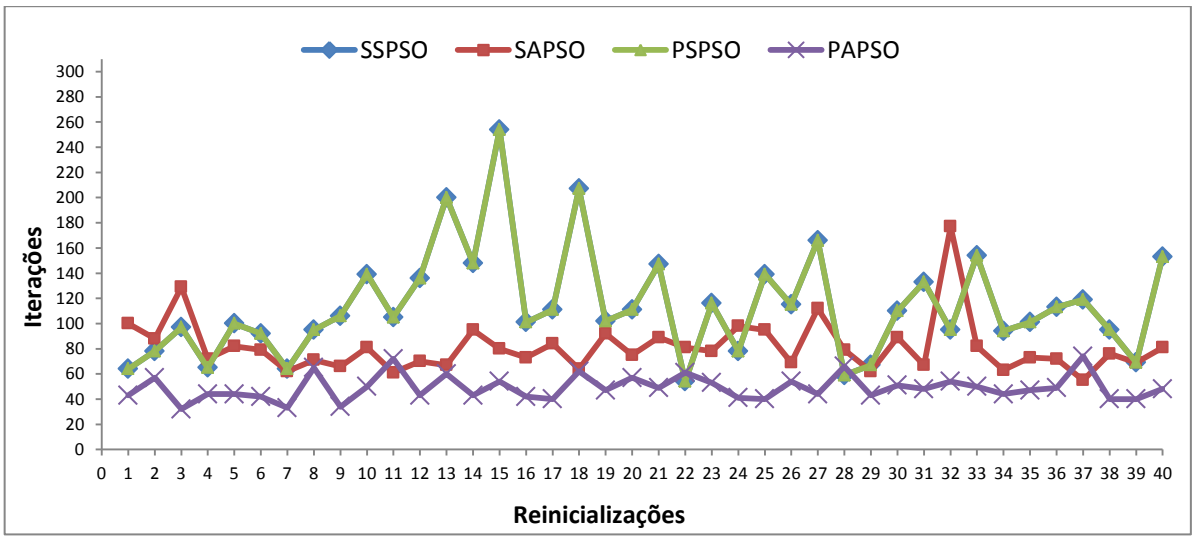


Figura 6. 5 – Desempenho dos algoritmos durante a realização dos 40 experimentos

Pela posição dos pontos e suavidade da curva, fica evidente o bom e uniforme desempenho obtido com o algoritmo paralelo assíncrono, o qual supera os demais métodos. Comportamento similar é verificado com a versão sequencial do algoritmo assíncrono, o qual, apesar de apresentar o menor desempenho que o PAPSO, demonstrou-se mais regular, exceto pela terceira e trigésima segunda reinicialização, se comparados com as versões sequenciais e síncronas do PSO. Com a Tabela 6.7, pode-se comparar a média das 40 rodadas para cada variável de projeto, bem com a melhor solução encontrada por Sedaghati, 2005.

Tabela 6. 7 – Média das variáveis de projeto obtidas com os algoritmos.

Grupos de Barras	Sedaghati (2005)	Presente Trabalho			
	FM	SSPSO	SAPSO	PSPSO	PAPSO
	Var.Proj. (cm²)	Var. Proj.$_{\text{média}}$	Var. Proj.$_{\text{média}}$	Var. Proj.$_{\text{média}}$	Var. Proj.$_{\text{média}}$
G1	3,499	3,440	3,364	3,440	3,505
G2	7,932	7,908	7,945	7,908	7,927
G3	0,645	0,771	0,645	0,771	0,645
G4	0,645	0,645	0,645	0,645	0,655
G5	8,056	10,209	10,056	10,209	8,159
G6	8,011	8,086	7,991	8,086	8,025
G7	0,645	0,645	0,645	0,645	0,645
G8	0,645	0,645	0,645	0,657	0,645
G9	12,812	12,608	12,337	14,437	12,766
G10	8,061	8,061	8,112	8,160	8,106
G11	0,645	0,645	0,646	0,646	0,645
G12	0,645	0,645	0,645	0,645	0,654
G13	17,279	17,273	18,408	16,987	17,370
G14	8,088	8,155	8,155	8,160	8,141
G15	0,645	0,645	0,645	0,645	0,646
G16	0,645	0,645	0,645	0,645	0,645

Em média, os métodos propostos apresentam soluções próximas às obtidas por Sedaghati, 2005.

De maneira análoga, realizaram-se os testes com a treliça plana de 18 barras, revelando um comportamento similar ao encontrado no problema estrutural da treliça espacial de 72. Entretanto, observou-se que os algoritmos assíncronos apresentaram um ganho em termos de acurácia e robustez no problema da treliça de 18 barras, em relação aos algoritmos

síncronos. Os respectivos valores das métricas discutidas são ilustrados na Tabela 6.8.

Tabela 6. 8 – Massa otimizada e respectivas métricas para a treliça de 18 barras

	Método	$Massa_{média}$ [kg]	$Acur.$ [kg]	*Robust.* [kg]	$Desemp._{médio}$
Presente Trabalho	SSPSO	2.338,341	290,141	251,158	10080
	SAPSO	2.160,359	100,090	87,139	4641
	PSPSO	2.338,341	290,141	251,158	10080
	PAPSO	2.169,990	121,790	75,581	2982
	Método	$Massa_{min}$ [kg]			
Lee e Geem (2005)	HS	2.048,200			

A Tabela 6.9 apresentam a média das 40 rodadas para cada variável de projeto, bem como seu respectivo desvio padrão, os valores médios podem ser comparados com a melhor solução encontrada por Lee e Geem, 2005, para confirmar a proximidade entre as soluções.

Tabela 6. 9 – Média das variáveis de projeto obtidas com os algoritmos seriais.

		Lee e Geem (2005)	Presente Trabalho			
		HS	SSPSO	SAPSO	PSPSO	PAPSO
		Var.Proj.	Var. $Proj._{média}$	Var. $Proj._{média}$	Var. $Proj._{média}$	Var. $Proj._{média}$
Coordenadas Nodais (m)	X3	22,939	23,676	23,225	23,676	23,789
	Y3	4,427	4,498	4,571	4,498	4,533
	X5	16,010	16,741	16,275	16,741	16,781
	Y5	3,462	3,145	3,195	3,145	3,157
	X7	10,213	10,544	10,187	10,544	10,471
	Y7	2,299	1,874	1,973	1,874	1,870
	X9	4,961	5,136	5,119	5,136	5,162
	Y9	0,777	0,222	0,421	0,222	0,274
Seção Trans. Grupos de Barras (cm^2)	G1	81,613	75,018	76,139	75,018	74,198
	G2	111,097	115,017	115,519	115,017	115,389
	G3	39,806	73,839	46,594	73,839	48,146
	G4	22,903	39,847	31,920	39,847	35,366

As soluções apresentadas na Tabela 6.10, as quais atendem as restrições de tensão (segundo tol_{FP}, Pseudocódigo_6 da seção 3.6.1), demonstram uma significante

suscetibilidade dos métodos de ficarem presos aos ótimos locais, no entanto, esses resultados tendem a melhorar perante uma calibração mais eficiente dos parâmetros utilizados pelo PSO. A prova de que a utilização de um conjunto de parâmetros corretos do PSO reproduz excelentes soluções, é apresentado na otimização do domo de 120 barras a seguir.

A otimização do domo de 120 barras mostrou-se de fácil solução, respeitando as restrições impostas pelo problema, como podem ser verificadas pelas Tabelas 6.11 e 6.12, de modo que os algoritmos propostos foram capazes de encontrar melhores valores que os apresentados por Lee e Geem, 2004. Este comportamento impossibilita a avaliação da acurácia (rever seção 4.1) dos algoritmos, já as demais métricas bem como a massa média otimizada, podem ser verificadas na Tabela 6.12.

Tabela 6. 10 – Massa otimizada e respectivas métricas para o domo de 120 barras.

	Método	$Massa_{média}$ [kg]	*Acur.* [kg]	*Robust.* [kg]	$Desemp._{médio}$
Presente Trabalho	SSPSO	7.633,82	-	0,2	3900
	SAPSO	7.633,70	-	0,35	3900
	PSPSO	7.633,82	-	0,2	3900
	PAPSO	7.633,05	-	2	3600
	Método	$Massa_{min}$ [kg]			
Lee e Geem (2004)	HS	8.939,3			

Pelo fato dos algoritmos desenvolvidos encontrarem uma estrutura mais otimizada que a obtida por Lee e Geem (8.939,3 kg) com o *HS*, nas Tabelas 6.11 e 6.12, são apresentadas as tensões atuantes nas barras ($\sigma_{atuante}$), as quais respeitam (ou seja, são inferiores) as restrições quanto à tensão admissível de tração (σ_{adm}^{T}) e compressão, considerando flambagem (σ_{adm}^{F}). Nas respectivas tabelas, por conversão, valores negativos correspondem à tensão de compressão e valores positivos, tensão de tração.

Tabela 6. 11 – Tensões atuantes no domo de 120 barras [Pa].

Grupos de Barras	SSPSO			SAPSO		
	$\sigma_{atuante}$	σ_{adm}^{T}	σ_{adm}^{F}	$\sigma_{atuante}$	σ_{adm}^{T}	σ_{adm}^{F}
G1	-1.5643E7	+2.4000E8	-1.5792E7	-1.5706E7	+2.4000E8	-1.5707E7
G2	-4.4429E7	+2.4000E8	-4.4754E7	-4.3928E7	+2.4000E8	-4.5446E7
G3	-2.1829E7	+2.4000E8	-2.2000E7	-2.1871E7	+2.4000E8	-2.1943E7

G4	-1.1513E7	+2.4000E8	-1.1528E7	-1.1458E7	+2.4000E8	-1.1604E7
G5	+4.4613E7	+2.4000E8	-1.1528E7	+4.4613E7	+2.4000E8	-1.1604E7
G6	-2.9089E7	+2.4000E8	-2.9096E7	-2.9063E7	+2.4000E8	-2.9131E7
G7	-1.5712E7	+2.4000E8	-1.5939E7	-1.5750E7	+2.4000E8	-1.5887E7

Tabela 6. 12 – Tensões atuantes no domo de 120 barras [Pa].

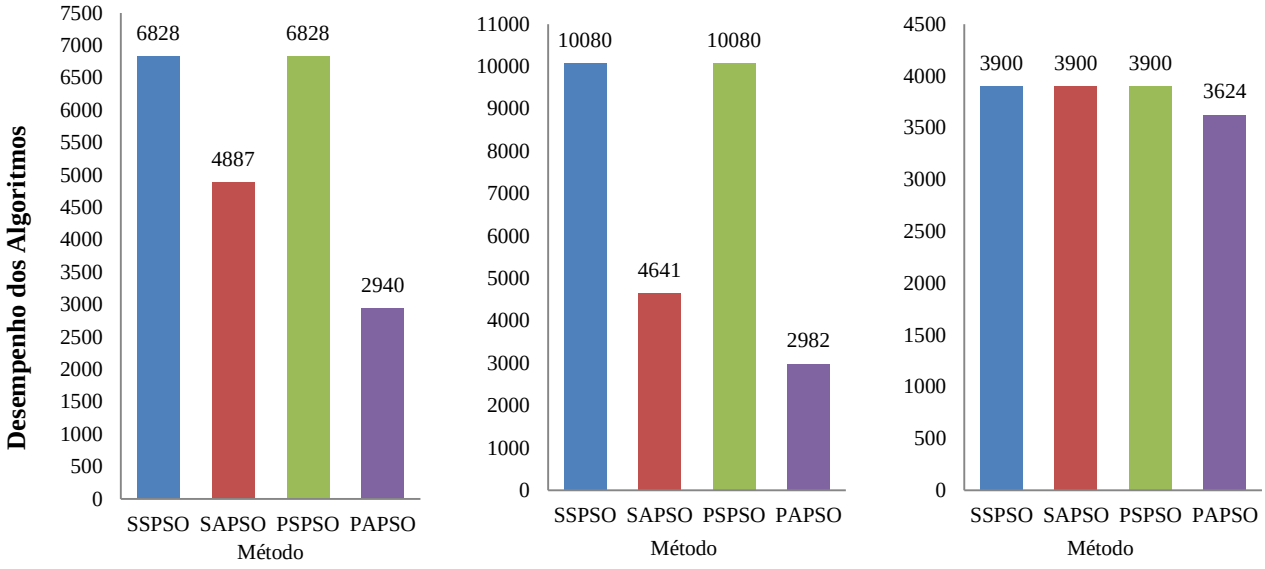
Grupo de Barras	PSPSO			PAPSO		
	$\sigma_{atuante}$	σ_{adm}^{T}	σ_{adm}^{F}	$\sigma_{atuante}$	σ_{adm}^{T}	σ_{adm}^{F}
G1	-1.5650E7	+2.4000E8	-1.5783E7	-1.5582E7	+2.4000E8	-1.5876E7
G2	-4.4099E7	+2.4000E8	-4.5208E7	-4.4494E7	+2.4000E8	-4.4665E7
G3	-2.1891E7	+2.4000E8	-2.1915E7	-2.1873E7	+2.4000E8	-2.1940E7
G4	-1.1482E7	+2.4000E8	-1.1570E7	-1.1483E7	+2.4000E8	-1.1568E7
G5	+4.4613E7	+2.4000E8	-1.1570E7	+4.4402E7	+2.4000E8	-1.1568E7
G6	-2.8912E7	+2.4000E8	-2.9337E7	-2.9053E7	+2.4000E8	-2.9145E7
G7	-1.5781E7	+2.4000E8	-1.5844E7	-1.5769E7	+2.4000E8	-1.5861E7

Quanto aos valores das variáveis de projeto para a respectiva massa encontrada pelos algoritmos, estes podem ser verificados e comparados aos obtidos por Lee e Geem, 2004, na Tabela 6.13.

Tabela 6. 13 – Média das variáveis de projeto obtidas com os algoritmos seriais.

Grupos de Barras	Lee e Geem (2004)	Presente Trabalho			
	HS	SSPSO	SAPSO	PSPSO	PAPSO
	Var.Proj. (m²)	$\text{Var. Proj.}_{\text{média}}$	$\text{Var. Proj.}_{\text{média}}$	$\text{Var. Proj.}_{\text{média}}$	$\text{Var. Proj.}_{\text{média}}$
G1	0,0021	0,0020	0,0020	0,0020	0,0020
G2	0,0015	0,0016	0,0016	0,0016	0,0016
G3	0,0025	0,0021	0,0021	0,0021	0,0021
G4	0,0017	0,0014	0,0014	0,0014	0,0014
G5	0,0007	0,0005	0,0005	0,0005	0,0005
G6	0,0021	0,0016	0,0016	0,0016	0,0016
G7	0,0018	0,0015	0,0015	0,0015	0,0015

Os resultados apresentados até agora são reunidos na Figura 6.6 para facilitar a comparação entre o desempenho obtido pelos algoritmos nos três casos analisados.



(a) (b) (c)

Figura 6. 6 – (a) Desempenho para a treliça 72 barras, (b) Desempenho para a treliça 18 barras, (c) Desempenho para o domo 120 barras

Pela Figura 6.6, pode-se notar uma semelhança entre a atuação registrada pelos modelos do PSO na otimização das treliças de 72 e 18 barras, demonstrando claramente um desempenho superior da forma assíncrona do PSO em relação ao PSO síncrono. Esse comportamento é dado em função da característica de assincronismo do algoritmo, permitindo que a cada pseudo-revoada, haja uma verificação dos valores ótimos apresentados pela partícula, os quais, se melhores que os da pseudo-revoada anterior, possibilitam que informações adicionais sejam fornecidas ao enxame. Mais expressivo e observado em todos os problemas analisados, é o desempenho registrado pela versão paralela do algoritmo assíncrono. Esse fato ocorre em função do PAPSO disponibilizar eventuais valores de partículas ótimas encontradas para as outras partículas que estão sendo analisadas paralelamente, fato este não possível numa versão serial do algoritmo. Este comportamento do algoritmo paralelo possibilita que a convergência para o ótimo se processe com uma significativa diminuição de avaliações da função objetivo.

A Figura 6.6 (c) não demonstra uma diferença muito expressiva de desempenho entre os métodos propostos, dando indícios de que um algoritmo pode apresentar um bom desempenho para um determinado problema, no entanto, esse comportamento não é garantindo sob todos os problemas de otimização existentes [Yang, 2010].

6.2.2 Quanto ao Desempenho Computacional

Para avaliar o desempenho computacional dos algoritmos na solução dos problemas estruturais da treliça de 72 barras, 18 barras e o domo de 120 barras, utilizou-se o desempenho dos algoritmos obtido anteriormente em cada problema. Sendo assim, adota-se como critério de parada o número de iterações correspondente ao desempenho do algoritmo apresentado nas Tabelas 6.6, 6.8 e 6.10.

Nas Figuras 6.7 e 6.8, são apresentados os gráficos que representam o tempo de execução computacional consumido na otimização da estrutura espacial de 72 barras bem como os ganhos obtidos com as variações do PSO em relação ao SSPSO (PSO padrão).

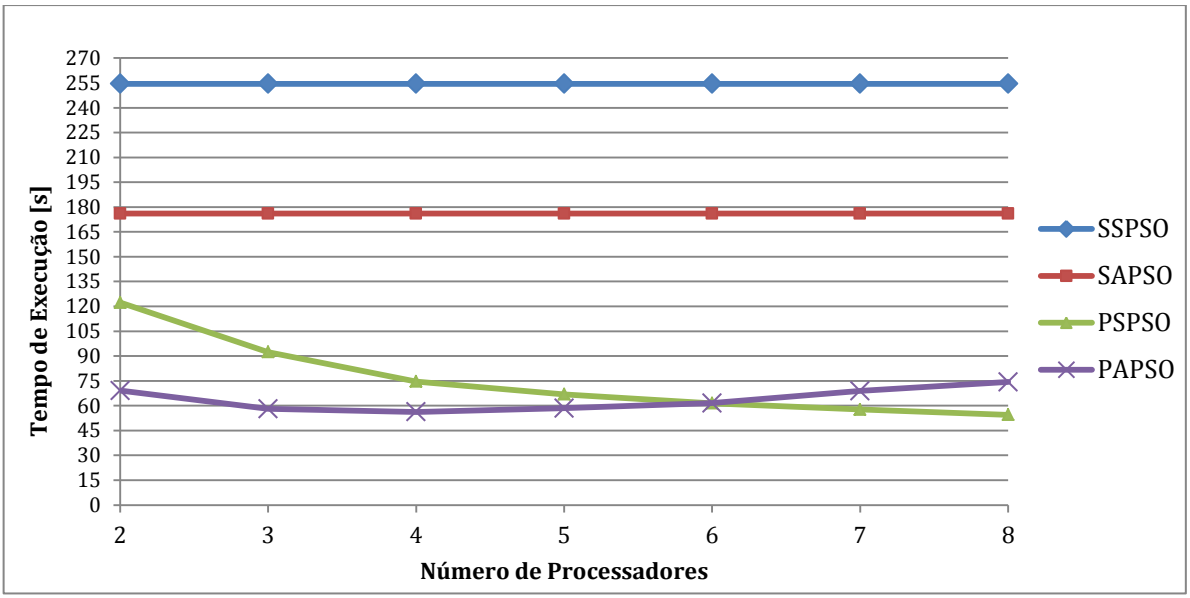


Figura 6. 7 – Tempo de execução para a otimização da treliça de 72 barras de acordo com o número de processadores

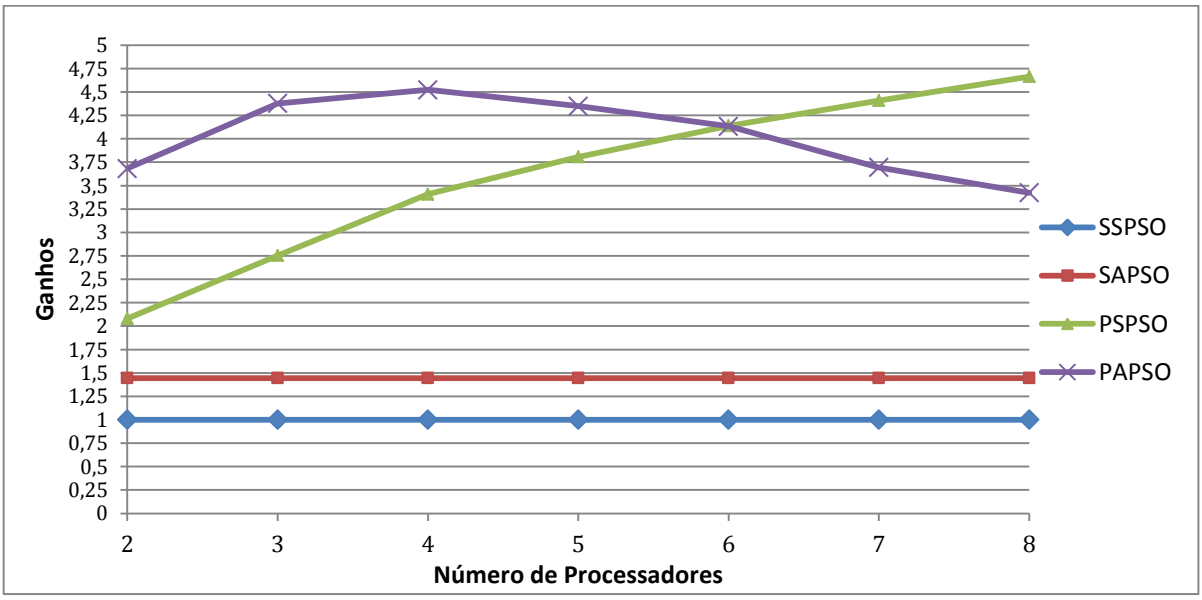
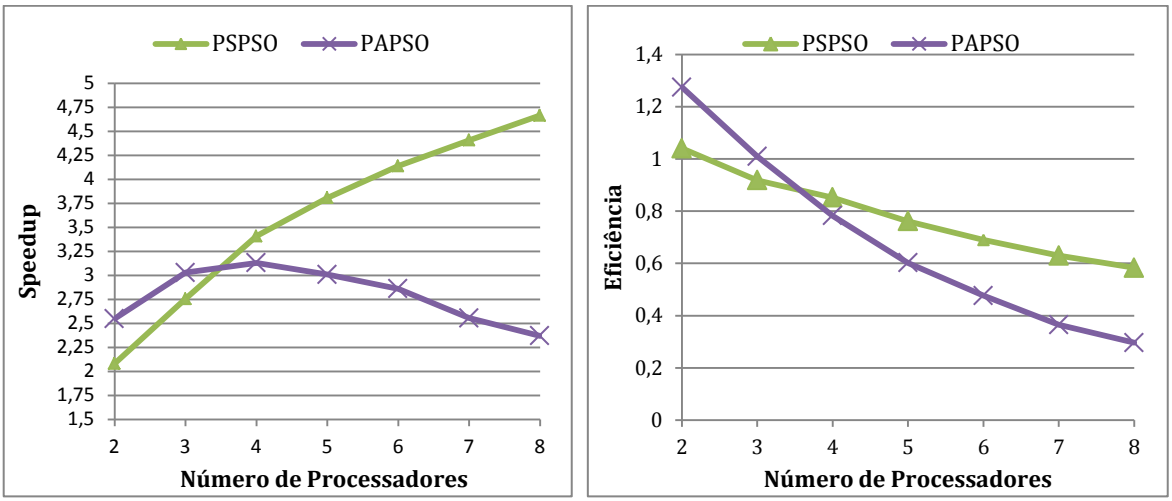


Figura 6. 8 – Ganhos em tempo de processamento com as variações do PSO padrão

O desempenho dos algoritmos paralelos, em termos de *Speedup* e *Eficiência*, podem ser analisados nos gráficos da Figura 6.9.



(a) (b)

Figura 6. 9 – (a) *Speedup* da treliça de 72 barras, (b) *Eficiência* da treliça de 72 barras

Conforme esperado, os algoritmos paralelos apresentam maiores ganhos os quais aumentam à medida que mais processadores são envolvidos no processamento, esse comportamento se mantém ao longo de todo o experimento para o PSPSO, já no PAPSO, a partir do quarto processador, os ganhos diminuem à medida que mais processadores são adicionados. Para justificar tal comportamento, os algoritmos são submetidos à otimização do

domo de 120 barras, o qual é explorado em função da particularidade descrita no parágrafo abaixo.

Sabe-se que a etapa do código que mais influencia no desempenho do algoritmo de otimização corresponde ao tempo necessário para cada avaliação dos candidatos a solução, que para os casos de otimização considerados consiste na analise estrutural pelo Método dos Elementos Finitos (MEF). Para a treliça de 72 barras, a etapa supracitada corresponde a um tempo aproximado de 0,0363 segundos, acredita-se que este tempo seja relativamente pequeno (pequeno o suficiente para caber no cache) a ponto de expor maiores particularidades dos algoritmos desenvolvidos. Sendo assim, ao monitorar o tempo de análise estrutural do domo de 120 barras, verificou-se um aumento de 10 vezes neste tempo em relação à treliça de 72 barras, com essa diferença (dentre outras), espera-se uma representação mais significante do comportamento dos algoritmos propostos.

Os gráficos apresentados abaixo (Figuras 6.10 e 6.11) mostram o desempenho dos algoritmos ao otimizar a estrutura de 120 barras quanto ao tempo de execução bem como quanto aos ganhos obtidos com as variações do PSO em relação ao SSPSO (PSO padrão).

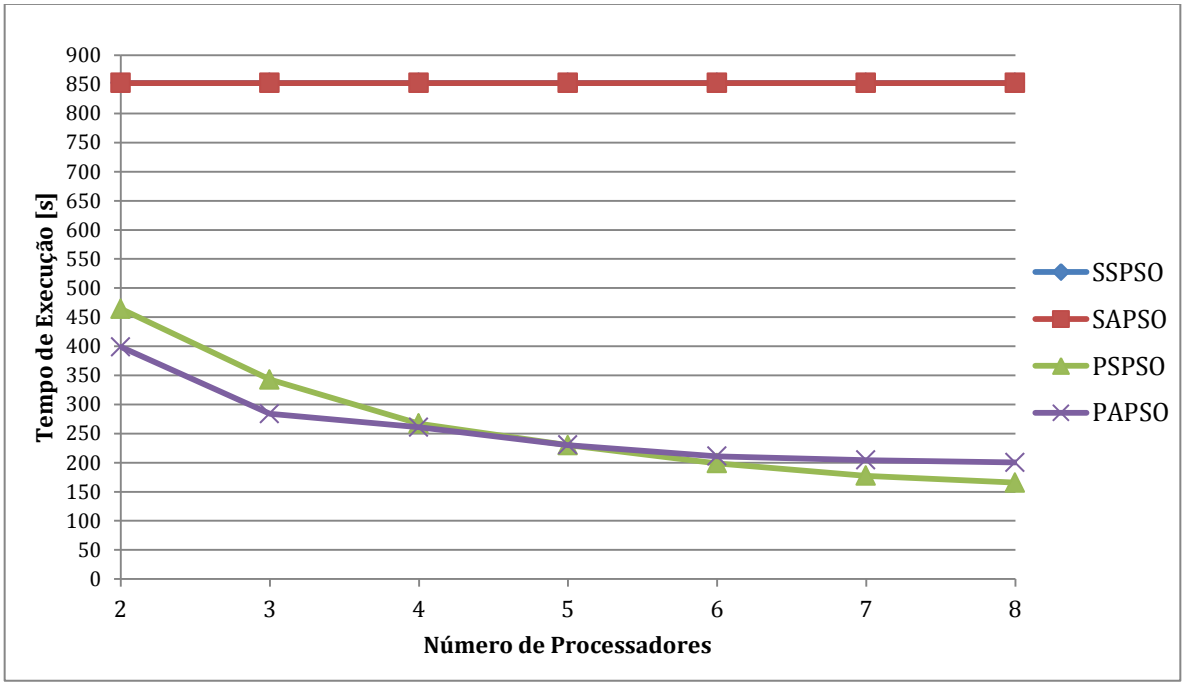


Figura 6. 10 – Tempo de execução para a otimização do domo de 120 barras de acordo com o número de processadores

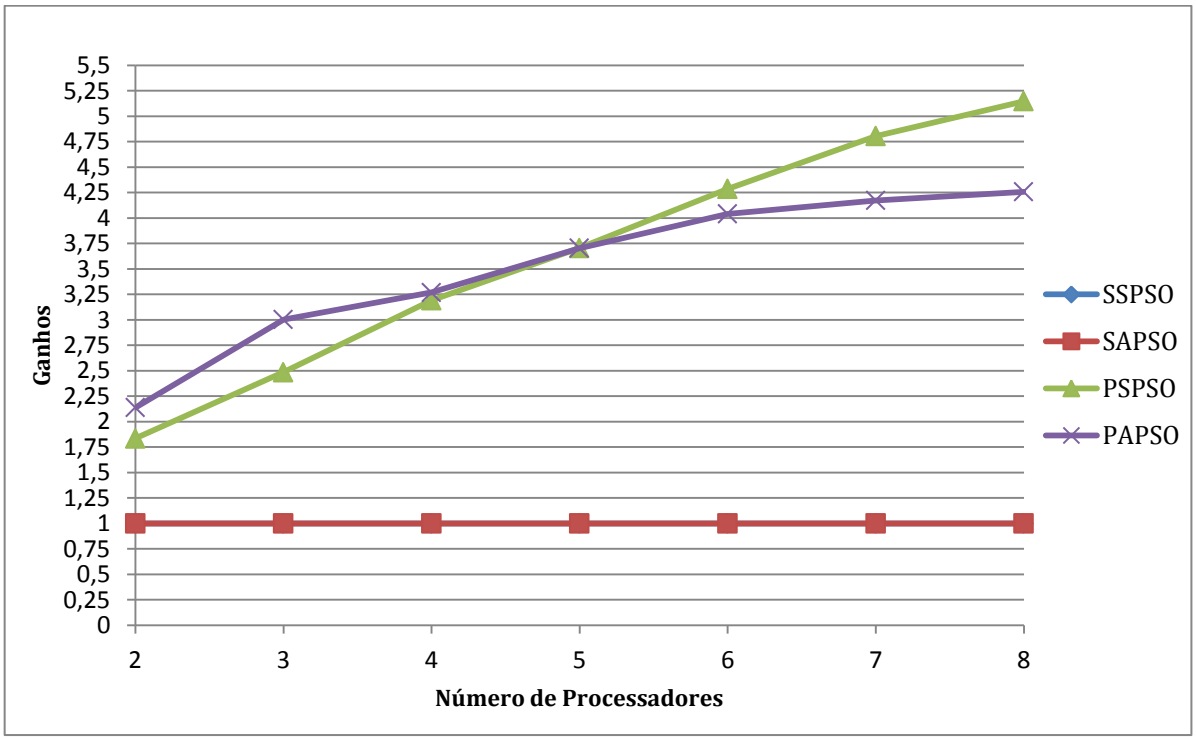


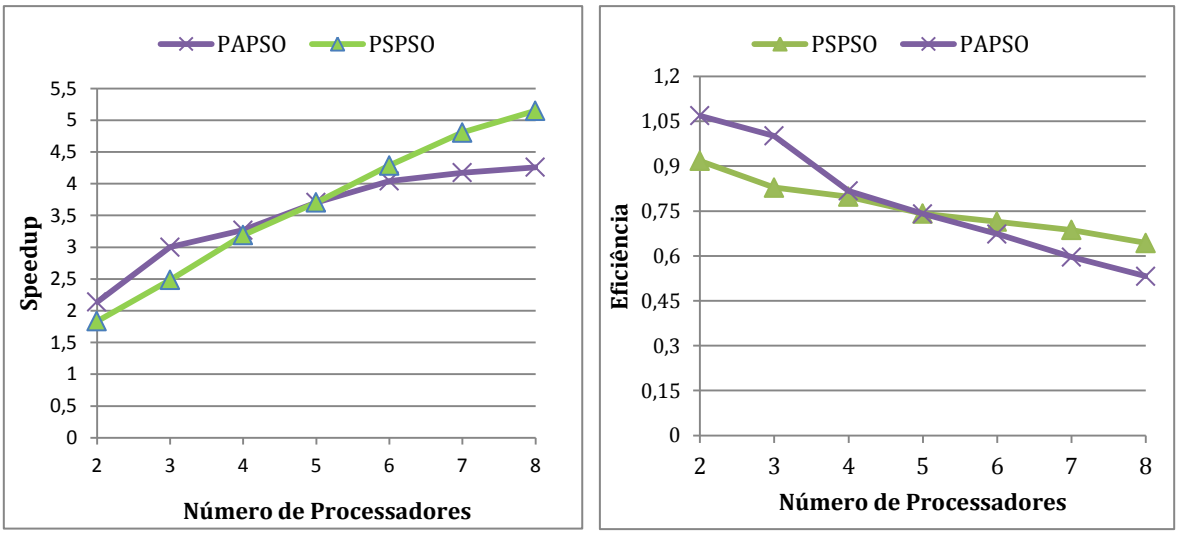
Figura 6. 11 – Ganhos em tempo de processamento com as variações do PSO padrão para otimização do domo de 120 barras

Os respectivos gráficos confirmam a superioridade (em termos de desempenho computacional), dos algoritmos paralelos sob os algoritmos sequenciais, também registrada na otimização da treliça de 72 barras. Quanto à sobreposição de valores apresentada pelas versões seriais, esta é consequência do igual desempenho (em termos de número de chamadas da função objetivo) obtido pelos métodos na solução do problema, o qual pode ser verificada na Figura 6.6 (c).

Novamente verifica-se que a versão paralela assíncrona apresenta inicialmente um maior desempenho que a versão paralela síncrona, no entanto, esse comportamento se mantém perante a execução de até cinco processadores em paralelo e à medida que mais processadores são solicitados, o desempenho do PAPSO torna-se inferior ao do PSPSO.

No gráfico da Figura 6.11, os ganhos registrados pelo PSPSO crescem de forma quase que linear à medida que mais processadores são adicionados ao processamento (na treliça de 72 barras, uma curva nitidamente logarítmica representava o comportamento do PSPSO). Quanto ao PAPSO, diferentemente do comportamento apresentado na treliça de 72 barras, os ganhos registrados demonstram um crescente desempenho do método durante todo o experimento, contudo os maiores ganhos continuam sendo obtidos com o PSPSO.

A Figura 6.12 contêm os gráficos que reforçam as características discutidas no parágrafo anterior em termos de *Speedup* (Figura 6.12(a)) e *Eficiência* (Figura 6.12(b)).



(a) (b)

Figura 6. 12 – (a) *Speedup* da treliça de120 barras, (b) Eficiência da treliça de 120 barras

Tanto o *Speedup* quanto a *Eficiência* demonstram o crescente desempenho do PSPSO, bem como a tendência à desaceleração apresentada pelo PAPSO com o aumento da capacidade de processamento. Conforme informado, a natureza da paralelização utilizada está associada aos resultados supracitados nos gráficos. Para demonstrar melhor este efeito, com auxilio do gerenciador de tarefas do Windows (Figuras 6.13 e 6.14), realiza-se uma investigação detalhada quanto ao aproveitamento dos processadores perante a execução dos algoritmos na otimização do domo de 120 barras.

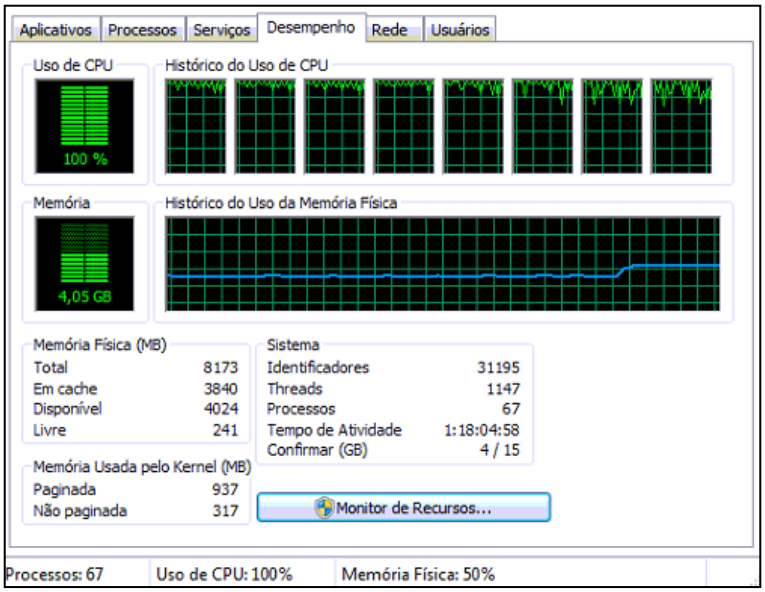


Figura 6. 13 – Desempenho dos processadores com o PSPSO

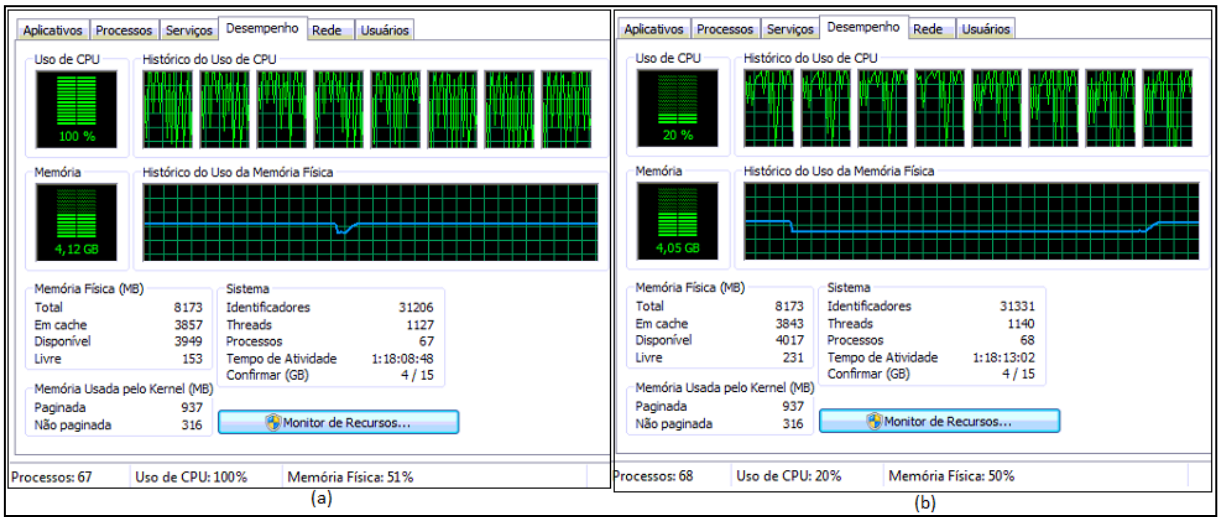


Figura 6. 14 – (a) Utilização de 100% do CPU, (b) Utilização de 20% do CPU.

Percebe-se que a execução do algoritmo PSPSO (Figura 6.13) demanda a capacidade máxima dos oito processadores disponíveis durante todo o processo de otimização. Já na execução do PAPSO (Figura 6.14), é verificado que o uso dos processadores oscila entre 100% (Figura 6.14(a)) e aproximados 20% (Figura 6.14(b)) ao longo da otimização.

Para ratificar o exposto na atual seção até o momento, nos dois parágrafos subsequentes dar-se-á uma explicação do desempenho computacional quanto à topologia de paralelização utilizada nos algoritmos, a qual justifica a variação de comportamento registrada

pelos métodos à medida que se aumenta o número de processadores envolvidos no processamento da solução.

O algoritmo PSPSO, por apresentar a estrutura de comunicação mestre-escravo, utiliza os processadores escravos somente para avaliação dos candidatos à solução (etapa em que é realizada a análise da estrutura pelo MEF). Esta característica evita a perda de tempo computacional com a atualização das partículas ou verificação do critério de convergência, pois estes são executados no processador mestre apenas após todas as partículas serem calculadas. Essa característica torna o método mais eficiente quanto maior for o tempo necessário para a avaliação de cada candidato, pois nessa condição, o tempo gasto com a comunicação (ou troca de dados) entre os processadores fica pouco expressivo, diferentemente do que ocorre com o PAPSO.

O moderado desempenho apresentado pelo PAPSO, contradiz a expectativa levantada ao analisar o método desenvolvido por Koh *et al*, 2006 (o qual é apresentado como sendo uma alternativa mais eficiente para a computação paralela com o PSO). Essa abordagem pode ser justificada com o auxilio da Figura 6.14, onde fica claro o ponto fraco do algoritmo desenvolvido na presente pesquisa, o qual não é encontrado no algoritmo proposto pelo autor. A característica destacada consiste na necessidade que o algoritmo paralelo assíncrono desenvolvido apresenta de estabelecer um ponto de sincronização ao término da avaliação dos candidatos (rever subseção 3.2.2). Segundo Masuero, 2009, quando uma divisão de tarefas ou carga de trabalho computacional ótima não é obtida, alguns processadores irão processar as tarefas a eles alocadas mais rapidamente que outros, aguardando pelos demais em uma barreira de sincronização. Como a execução do código não prossegue sem que todos os processadores tenham alcançado a barreira, o tempo total de processamento do conjunto é o tempo de processamento do processador mais lento. Ao contexto do problema abordado, no final do cálculo responsável pela análise estrutural da treliça (Figura 6.14(a)), se faz necessária a interrupção do processo para que todos os processadores apresentem seus resultados e só então possam prosseguir com a comunicação entre eles (Figura 6.14(b)) para que todos conheçam o melhor candidato. Essa propriedade possibilita que a solução seja encontrada mais rapidamente conforme o desempenho dos algoritmos apresentados na Figura 6.6 (c), entretanto, como pode ser verificado na Figura 6.14(b), perde-se um precioso tempo computacional com a parada ou sincronização a cada lote de partículas, para troca de informações.

Com o aumento na quantidade de processadores, aumenta-se o tempo de aguardo durante a sincronização e a intercomunicação entre eles (causada pela topologia em forma de estrela) surge como mais um agravante para a queda de desempenho computacional do método.

A discussão realizada no parágrafo anterior justifica o bom desempenho computacional do PAPSO no início dos testes, onde poucos processadores são solicitados e o tempo gasto com a sincronização e comunicação é pequeno, logo, à medida que mais processadores são solicitados, ocorre o efeito inverso do desempenho.

Tanto na otimização da treliça de 72 barras quanto do domo de 120, a implementação paralela nos algoritmos por enxame de partículas promoveram grandes ganhos de desempenho computacional, salientando que em problemas que necessitam executar cálculos mais complexos, como no caso do domo de 120 barras, este desempenho demonstra-se de uma forma mais expressiva. Entretanto, com base nas análises realizadas, acredita-se que a otimização de problemas de baixo custo computacional não viabiliza o uso da computação paralela proposta na presente pesquisa, sendo assim, é realizado um estudo do desempenho dos algoritmos propostos na otimização da treliça de 18 barras, um caso trivial de *benchmark*, de rápida implementação.

O tempo de execução computacional necessário para otimização da treliça plana de 18 barras bem como os ganhos obtidos com as variações do PSO em relação ao SSPSO (PSO padrão), são monitorados nos gráficos das Figuras 6.15 e 6.16.

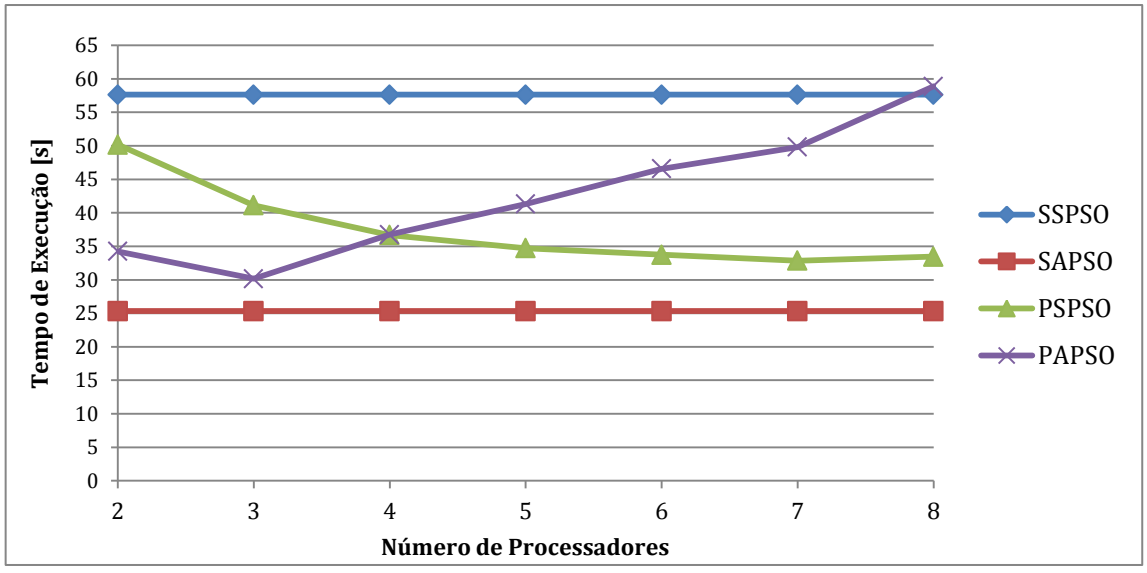


Figura 6. 15 – Tempo de execução para a otimização da treliça de 18 barras de acordo com o número de processadores

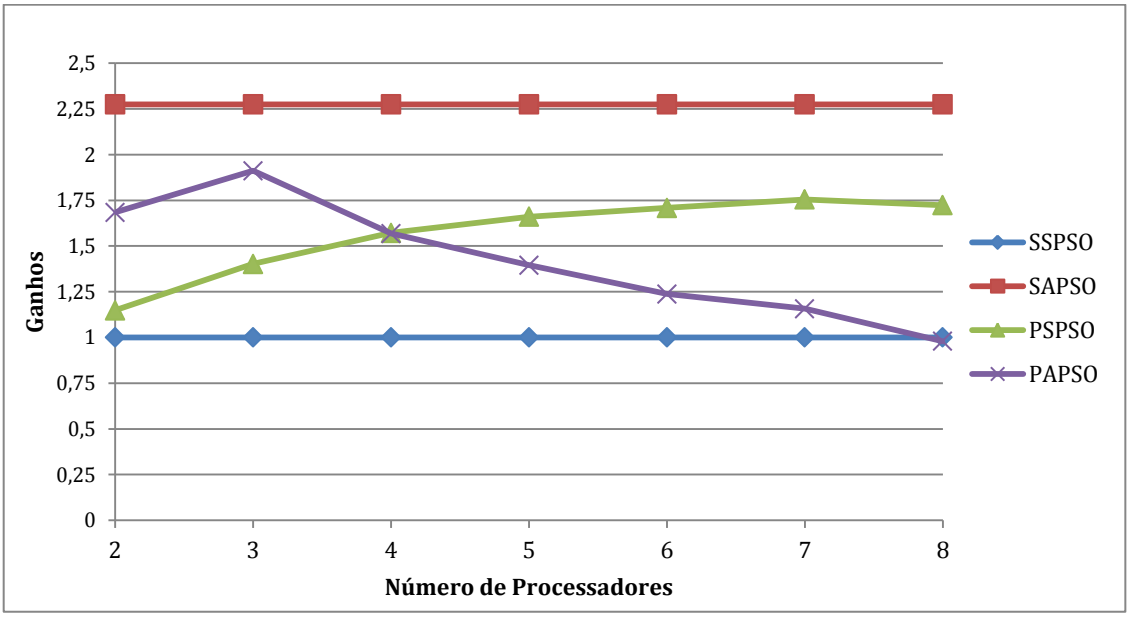
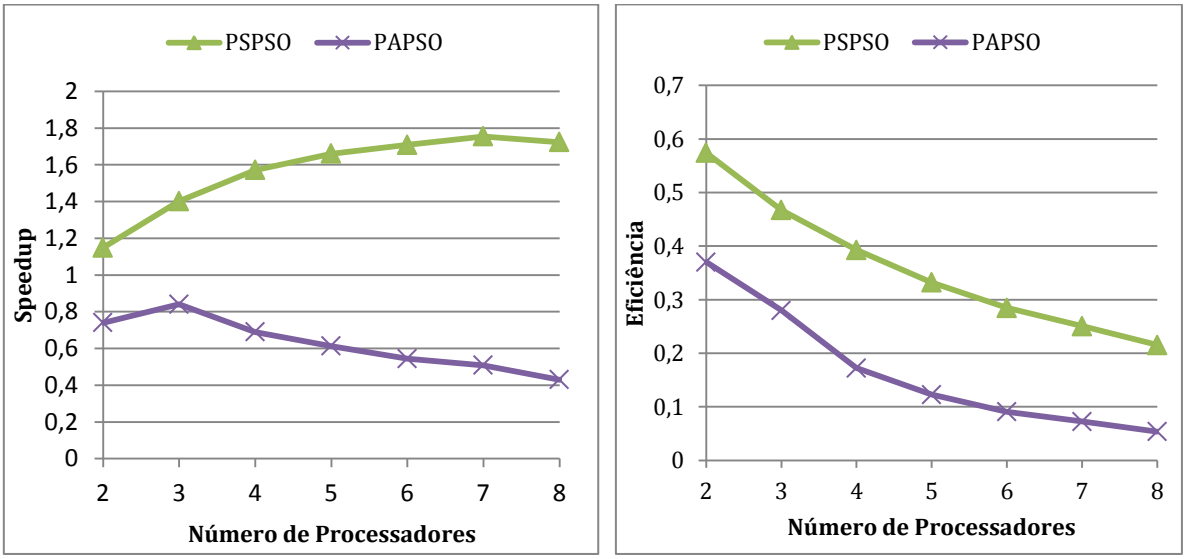


Figura 6. 16 – Ganhos em tempo de processamento com as variações do PSO padrão

No presente problema, percebe-se que as extensões paralelas dos algoritmos são computacionalmente mais custosas do que a forma serial do algoritmo assíncrono SAPSO, além disso, a perda de desempenho apresenta-se de forma mais acentuada à medida que mais processadores são utilizados. Esse comportamento repercute em desfavoráveis valores em termos de *Speedup* e *Eficiência*, novamente, notam-se (Figura 6.17) expressivas perdas à medida que mais processadores são solicitados.



(a) (b)

Figura 6. 17 – (a) *Speedup* da treliça de18 barras, (b) Eficiência da treliça de 18 barras

Conforme esperado, os ganhos obtidos com a paralelização para a solução da treliça plana de 18 barras são pouco atraentes. Na Figura 6.17 podem-se confirmar as afirmações feitas anteriormente, consolidando o algoritmo sequencial assíncrono como o método que registrou os maiores ganhos. Já sua extensão paralela surge como a mais sensível perante o aumento na capacidade de processamento.

A provável causa que inviabiliza a aplicação das extensões paralelas desenvolvidas no presente trabalho em problemas como o da treliça de 18 barras, consiste no tempo processamento necessário para cada avaliação dos candidatos a solução ser relativamente pequeno (em torno de 0,0054 segundos) em comparação com o tempo gasto com a troca de informações entre os processadores. Sendo assim, em problemas com baixo custo computacional (em termos de avaliação da função objetivo), a aplicação da paralelização proposta não possibilita melhores velocidades de processamento, se comparadas às versões seriais desenvolvidas.

7 CONCLUSÕES E TRABALHOS FUTUROS

Alguns resultados iniciais foram apresentados quanto à avaliação dos algoritmos desenvolvidos na otimização de funções padrão da literatura como também na otimização de problemas estruturais, sendo eles: treliça de 18 barras, 72 barras e domo de 120 barras, os quais revelaram algumas particularidades associadas à acurácia, robustez, desempenho dos algoritmos e desempenho computacional, confirmando o comportamento aleatório que os métodos de otimização apresentam de acordo com o problema ao qual são submetidos.

Os resultados numéricos indicam que a forma assíncrona do algoritmo por enxame de partícula (SAPSO) encontrou mais facilmente a solução ótima na maioria dos problemas estruturais analisados, superando significativamente os algoritmos síncronos em termos do número de chamadas à função objetivo. E se tratando da paralelização desenvolvida, verificou-se que sua aplicação ao algoritmo assíncrono (PAPSO) permitiu uma melhora ainda maior em sua velocidade de convergência.

Perante o apreciável desempenho dos algoritmos assíncronos, esperava-se que o desempenho computacional registrado pela versão assíncrona paralela (PAPSO) fosse superior ao PSPSO. De fato, esse comportamento é verificado a um determinado número de processadores envolvidos, o qual varia dependendo do problema tratado, limitando os ganhos de desempenho computacional do método ao número máximo de processadores. Como exemplo pode-se citar o tempo de execução gasto com os métodos na otimização do domo de 120 barras, onde é verificada a atuação superior do PAPSO em comparação com os demais métodos, quando submetido ao processamento de até 5 processadores em paralelo. Logo, ao acrescentar mais processadores, é verificada uma inversão de posições, e o PSPSO surge como uma melhor alternativa para se atingir maiores velocidades de processamento quando se dispõe de uma maior plataforma computacional. Como exposto no respectivo exemplo, o maior ganho obtido com o PAPSO em termos de tempo computacional utilizando 8 processadores foi de 4,26 vezes, já o PSPSO demonstrou ganhos superiores a 5,14 vezes.

Sob o ponto de vista dos resultados do processo de otimização, observa-se que a atualização imediata das partículas possibilitou aos algoritmos assíncronos chegarem a soluções melhores que as obtidas com a versão originalmente síncrona do PSO.

Cabe também salientar, que os algoritmos paralelos desenvolvidos não devem ser aplicados à otimização de problemas onde o custo computacional é pouco relevante. Isso

porque nestes casos, com uma boa manutenção dos parâmetros utilizados pelo PSO, é possível que o algoritmo atinja a convergência rapidamente, como ocorreu com o último caso analisado, onde os ganhos obtidos com o algoritmo sequencial assíncrono foram maiores do que as extensões paralelas durante todo o experimento.

Sendo assim, como o foco do trabalho é o processamento paralelo com multiprocessadores, os resultados apresentados demonstram que esta técnica mostrou-se bastante atrativa, já que para os problemas analisados com os algoritmos paralelos desenvolvidos foram conseguidos ganhos de 5,14 vezes. Além disso, em função do comportamento apresentado pelos algoritmos durante os testes, acredita-se que submeter os algoritmos paralelos desenvolvidos em problemas realísticos complexos e que tenham alto custo computacional promoverá ganhos ainda maiores.

Sugere-se também que com uma manutenção da pesquisa na área de otimização estrutural utilizando os métodos heurísticos, novas alternativas de implementação paralelas devem surgir possibilitando reduções ainda mais expressivas de tempos computacionais. Para tanto, mostra-se pertinente:

a) Desenvolver e implementar novas topologias de comunicação, de forma a minimizar a sobrecarga de comunicação verificada na topologia em estrela, utilizada pelo método PAPSO;
b) Modificar o código do PAPSO de modo que não seja mais necessária a existência da etapa de sincronização (barreira);
c) Desenvolver versões híbridas do PSO, de modo a obter códigos mais robustos e de melhor velocidade de convergência, para posteriormente aplicar as estratégias paralelas nos algoritmos mais “inteligentes”;
d) Implementar os códigos desenvolvidos no ambiente *MATLAB*$^{@}$ em linguagem de programação compilada, como por exemplo: Fortran, C e C++.
e) Implementar os algoritmos sequenciais síncrono e assíncrono do PSO em computação paralela com clusters;
f) Tornar paralela a etapa responsável pela análise estrutural;
g) Estudar a influência de parâmetros como o tamanho do enxame nos resultados obtidos através das estratégias empregadas.

REFERÊNCIAS

Arora, J. S. **Introduction to Optimum Design**. Elsevier Academic Press, 2nd edition, 2004.

Belal, M., El-Ghazawi, T. **Parallel Models for Particle Swarm Optimizers**. IJICIS, v. 4, n. 1, p. 100 – 111, 2004.

Cantú-Paz, E. **Efficient and Accurate Parallel Genetic Algorithms**. Kluwer Academic Publishers, Boston, 2000.

Berci, C. D. and Botura, C. P. **Modulação de Velocidade em Otimização Por Enxame de Partículas**. In: 7th Brazilian Conference on Dynamics, Control and Applications, Presidente Prudente, SP., p. 241-248, 2008.

Bergh, F. and Engelbrecht, A. P. **A Study of Particle Swarm Optimization Particle Trajectories**. Information Sciences, 176, 937–971, 2006.

Chan, J. & Lee, C.Y. **General Multiprocessor Task Scheduling**, Naval Research Logistics, v.46, 57-74, 1999.

Clerc, M., and Kennedy, J. **The Particle Swarm – Explosion, Stability, and Convergence in a Multidimensional Complex Space**. IEEE Transactions on Evolutionary Computation, v.6, p. 58-73, 2002.

Eberhart, R. and Kennedy, J. A New Optimizer Using Particle Swarm Theory, **Proccedings of the Sixth International Symposium on Micro Machine and Human Science**, p. 39-43, 1995.

Esmin, A. A. A. **Estudo de Aplicação do Algoritmo de Otimização por Enxame de Partícula na Resolução de Problemas de Otimização Legados ao SEP**. Tese de Doutorado, Universidade Federal de Itajubá, Departamento de Elétrica, 2005.

Flynn, M. J. **Very High Speed Computing Systems**. Proceedings of IEEE, v. 54, p. 1901-1909, 1966.

Foster, I. **Designing and Building Parallel Programs**. Concepts and Tools for Parallel Programs, Addison-Wesley, 1995.

Fritzsche, H. Programação **Não-Linear: Análise e Métodos.** São Paulo: Edgard Blücher: Ed. Da Universidade de São Paulo, 1978.

Gomes, H. M. A swarm Optimization Algorithm for Optimum Vehicle Suspension Design. In: **20th International Congress of Mechanical Engineering**, Novembro, Gramado, RS, Brazil, 2009.

Gomes, H. M. **Análise Matricial de Estruturas**. Notas de Aula, Departamento de Estruturas e Construção Civil, Universidade Federal de Santa Maria, p. 90, 2003.

Gomes, H. M. **Truss Optimization With Dynamic Constraints Using a Particle Swarm Algorithm**. Expert Systems with Applications, v. 38, p. 957-968, 2011.

Haftka, R. and Gürdal, Z. **Elements of Structural Optimization**. Kluver Academic Publishers, 3rd edition, 1991.

Hassan, R., Cohanim, B., Weck, O. **A Comparison of Particle Swarm Optimization and the Generic Algorithm**. Massachusetts Institute of Technology, Cambridge, 2005.

Hibbert, D. B. **A Hybrid Genetic Algorithm for the Estimation of Kinetic Parameters**. Chemometrics and Intelligent Laboratory Systems, v. 19, p. 319-329, 1993.

Hughes, T. J. R. **The Finite Element Method – Linear Static and Dynamic Finite Element Analysis**. Prentice-Hall, 1987.

Kaewkamnerdpong, B., Bentley, P. J. **Perceptive Particle Swarm Optimization: An Investigation.** Department of Computer Science, University College London, UK, 2005.

Kennedy, J., and Mendes, R. Population Structure and Particle Swarm Performance. In: **Congress on Evolutionary Computation**, v.2, p. 1671-1676, 2002.

Koh, B., George, A. D., Haftka, R., T. and Fregly, B., J. Parallel Asynchronous Particles Swarm Optimization, **International Journal for Numerical Methods in Engineering**, v.67(4), p.578-595, 2006.

Konzelman, C. J. **Dual Methods and Approximation Concepts for Structural Optimization.** M.A.Sc. thesis, Department of Mechanical Engineering, Universityof Toronto, 1986.

Kripka, M. Discrete Optimization of Trusses by Simulated Annealing, **Journal of the Brazilian Society of Mechanical Sciences and Engineering**, Rio de Janeiro, v. 26, n.2, pp. 170 – 173, ISSN 1678-5878, 2004.

Kvasnicka, V., and Pospíchal, J. **A Hybrid of Simlex Method and Simulated Annealing**. Chemometrics and Intelligent Laboratory Systems, v. 39, p. 161-173, 1997.

Lee, K. S., Geem, Z. W. **A New Meta-Heuristic Algorithm for Continuous Engineering Optimization: Harmony Search Theory and Practice**. Computer Methods in Applied Mechanics and Engineering, v. 194, p. 3902-3933, 2005.

Lee, K. S., Geem, Z. W. **A New Structural Optimization Method Based on the Harmony Search Algorithm.** Computers and Structures, v. 82, p. 781 – 798, 2004.

Masuero, J. R. **Computação Paralela na Análise de Problemas de Engenharia Utilizando o Método dos Elementos Finitos**. Tese de Doutorado, Universidade Federal do Rio Grande do Sul, 2009.

Miguel, L. F. F., Fadel Miguel, L. F. Shape and Size Optimization of Truss Structures Considering Dynamic Constraints Through Modern Metaheuristic Algorithms, **Expert Systems with Applications**, v. 39, p. 9458–9467, 2012.

Miranda, V. **Computação Evolucionária Fenotípica,** Notas de Aula, versão 2.0, Universidade do Porto, Porto, 2005.

Mostaghim, S., Branke, J. and Schmeck, H. **Multi-objective Particle Swarm Optimization on Computer Grids.** GECCO, p. 869-875, 2007.

Parsopoulos, K. E., and Vrahatis, M. N. **Initializing the Particle Swarm Optimizer Using the Nonlinear Simplex Method**. Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation, p. 216-221, 2002.

Patel, N. R., Smith, R. L., and Zabinsky, Z. B. **Pure Adaptive Search in Monte Carlo Optimization**. Mathematical Programming, v. 43, p. 317-328, 1988.

Peer, E. S., Van Den Bergh, F., Engelbrecht, A. P. **Neighborhoods with the Guaranteed Convergence PSO, Swarm Intelligence Symposium**, 235 - 242, 2003.

Price, W. L. A Controlled Random Search Procedure for Global Optimization, **Computer Journal**, v. 20, p. 367-370, 1976.

Rozvany, G., Bendsoe, M.P., Kirsch, U. **Layout Optimization of Structures**. Applied Mechanical Review, v.2, p. 41-119, 1995.

Schutte, J. F. and Groenwold, A. A. A Study of Global Optimization using Particle Swarm, **Journal of Global Optimization**, v. 31, p. 93-108, 2003.

Schutte, J. F., Reinbolt, J. A., Fregly, B. J., Haftka, R. T. and George, A. D., Parallel Global Optimization With The Particle Swarm Algorithm, **International Journal for Numerical Methods in Engineering**, v. 61, p. 2296-2315, 2004.

Schwaab, M. **Avaliação de Algoritmos Heurísticos de Otimização em Poblemas de Estimação de Parâmetros**. Dissertação de Mestrado, Universidade Federal do Rio de Janeiro, COPPE, 2005.

Sedaghati, R. Benchmark Case Studies in Structural Design Optimization Using the Force Method. **International Journal of Solids and Structures**, 42, 5848–5871, 2005.

Shi, Y. **Particle Swarm Optimization**. IEEE Neural Network Society, p. 8-13, 2004.

Shi, Y., and Eberhart, R. **A Modifiel Particle Swarm Optimizer**. IEEE International Conference of Evolutionary Computation, Anchorage, Alaska, 1998.

Souza, F.H., Gomes, H. M. Asynchronous Particle Swarm Algorithm for Size Truss Optimization With Strength and Natural Frequencies Constraints. In: **COBEM 21st Brazilian Congress of Mechanical Engineering,** Natal, RN, Brazil, October, p. 24 – 28, 2011.

Venter, G. and Sobieski, J. S. A Parallel Particle Swarm Optimization Algorithm Accelerated by Asynchronous Evaluations, **World Congresses of Structural and Multidisciplinary Optimization**, 2005.

Waintraub, M. **Algoritmos Paralelos de Otimização por Enxame de Partículas em Problemas Nucleares**. Tese de Doutorado, Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Nuclear, 2009.

Waintraub, M., Baptista, R. P., Schirru, R., Pereira, C. M. N. A. **Desenvolvimento de um Algoritmo Genético Paralelo Utilizando MPI e sua Aplicação na Otimização de um Projeto Neutrônico**. Proceedings of the International Nuclear Atlantic Conference (INAC), 2005.

Yang, J. P. **Development of Genetic Algorithm-Based Approach for Structural Optimization.** Ph.D. Thesis, Nanyang Technology University Singapore, 1996.

Yang, X. Y. **Engineering Optimization: An Introduction with Metaheuristic Applications**. John Wiley & Sons, 25-28, 2010.

Zabinsky, Z. B., and Smith, R. L. **Pure Adaptive Search in Global Optimization**. Mathematical Programming, p. 53:323 – 338, 1992.

Zienkiewicz, O. and Taylor, R. L. **The Finite Element Method – The Basis**. v. 1. Elsevier Academic Press, 5th edition, 2000.

APÊNDICE A – CONFIGURAÇÃO DO MATLAB®

A transformação de um algoritmo estruturado num algoritmo paralelizado exige funções presentes na *PARALLEL COMPUTING TOOLBOX* do *MATLAB*®. Entretanto, isso só será possível se algumas configurações tenham sido realizadas na parte de paralelização do *MATLAB*®. Sendo assim, a seguir estão descritos os passos para configurar o *MATLAB*® para processamento paralelo.

1. O primeiro passo para configuração é iniciar o *software MATLAB*® ;
2. Clicar em *Parallel >> Configurations Manager...;*
3. Na janela *Configurations Manager*, selecione: *File>>New>>local;*
4. Ao abrir a janela Local *Scheduler Configuration Properties*, informe: número de trabalhadores disponíveis na guia *Scheduler*; número de trabalhadores para executar tarefas em paralelo na guia *Configuration*;
5. Configuradas todas as abas, clique em “ok”;
6. Surgirá uma janela de avaliação para estas configurações, basta clicar em *Start Validation* e o processo de validação é iniciado;
7. No final da validação todos os itens do *Status* devem estar *Succeeded.*

Feche a janela *Configurations Manager*. Agora o *MATLAB*® está apto a desempenhar suas funções na parte de processamento paralelo.

Maiores detalhes sobre a configuração descrita podem ser obtidas pelo *Help* do *MATLAB*®.