Universidade de Aveiro Departamento de Engenharia Mecânica
2009

João Filipe Moreira Caseiro

Estratégias Evolucionárias de Optimização de Parâmetros Reais

Universidade de Aveiro Departamento de Engenharia Mecânica
2009

João Filipe Moreira Caseiro

Estratégias Evolucionárias de Optimização de Parâmetros Reais

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob a orientação científica do Doutor António Gil D'Orey de Andrade Campos, Professor Auxiliar Convidado do Departamento de Engenharia Mecânica da Universidade de Aveiro

o júri

presidente

Prof. Doutor Robertt Ângelo Fontes Valente
Professor Auxiliar do Departamento de Engenharia Mecânica da Universidade de Aveiro

Prof. Doutor João Pedro Antunes Ferreira da Cruz
Professor Auxiliar do Departamento de Matemática da Universidade de Aveiro

Prof. Doutor António Gil d'Orey de Andrade Campos
Professor Auxiliar Convidado do Departamento de Engenharia Mecânica da Universidade de Aveiro

agradecimentos

Ao Professor Doutor António Gil D'Orey Andrade Campos, com quem tive o prazer de trabalhar, pela orientação, disponibilidade e amizade ao longo de todo este trabalho.

Ao Eng. João Alexandre pela disponibilização do programa FRAN.

A todos os meus amigos e colegas de trabalho, em especial ao Christian, Daniel e Sérgio, com quem partilhei este percurso académico e que me proporcionaram inúmeros momentos de boa-disposição e alegria. À Ana, pela amizade, apoio e paciência, sobretudo nos momentos mais difíceis.

Aos membros do GRIDS com os quais passei bons e revitalizantes momentos.

À minha família, por todo o apoio que me deram desde o início deste meu percurso.

palavras-chave

evolução diferencial, identificação de parâmetros, método dos elementos finitos, optimização estrutural, optimização por bandos de partículas.

resumo

Actualmente, existem diversos problemas de engenharia cujas propriedades podem ser expressas através de uma função, denominada função objectivo. Existem diversos métodos que possuem como principal objectivo minimizar a referida função. Os métodos baseados no gradiente são métodos nos quais a direcção e tamanho do passo são calculados a partir do declive da função objectivo. Apesar destes métodos necessitarem de reduzidos tempos de computação, estes podem convergir prematuramente ou ficar estagnados em mínimos locais. Os métodos de optimização baseados na teoria evolucionária são aproximações que possuem como principal desvantagem elevados tempos de computação. No entanto, estes apresentam uma grande flexibilidade na modelação de problemas de engenharia. Neste grupo, os algoritmos mais conhecidos e aplicados em problemas de optimização são os Algoritmos Evolucionários (EA's), Algoritmos Genéticos (GA's) e Evolução Diferencial (DE). Existem ainda algoritmos baseados em processos naturais tais como o algoritmo de Optimização por Bandos de Partículas (PSO), que reproduz o comportamento de bandos de animais.

Neste trabalho é desenvolvido um algoritmo de optimização de procura directa baseado em métodos diferenciais, evolucionários e no comportamento de animais. O algoritmo desenvolvido é aplicado a problemas de engenharia inversa.

Numa primeira fase, o algoritmo desenvolvido é validado e comparado com algoritmos existentes recorrendo a um conjunto de funções compostas especialmente criadas para este fim. Numa segunda fase, o algoritmo desenvolvido é aplicado a problemas de Engenharia Mecânica e Mecânica Computacional. Nesta secção, os problemas das três barras e da cúpula de 120 barras são analisados recorrendo ao Método dos Elementos Finitos (MEF). Seguidamente, os problemas de compressão de um provete cilíndrico e da placa com furo central são analisados. Nestes problemas a função a minimizar é dada por um programa do MEF comercial. Finalmente, o algoritmo é aplicado a um problema de identificação de parâmetros de um modelo constitutivo. O algoritmo desenvolvido apresenta bons resultados e uma boa taxa de convergência.

keywords

differential evolution, parameter identification, finite element method, structural optimization, particle swarm optimization.
.

abstract

Nowadays, there are many inverse engineering problems whose properties can be expressed by a function, called objective function. There are several methods whose main goal is to minimize the value of that function. The gradient-based methods are optimization methods in which the step direction and length are calculated in terms of the objective function's slope. Although these methods require little computation time, they may converge prematurely or get trapped in a local minima. The optimization methods based on the evolutionary theory are approaches that need, as a main disadvantage, high computation times. However, they have a great flexibility in modeling engineering problems. In this class of methods, the ones that are best known and more often applied in optimization problem are the Evolutionary Algorithms (EA's), Genetic Algorithms (GA's) and Differential Evolution (DE). There are also nature-inspired algorithms such as the Particle Swarm Optimization method (PSO) that mimics the behavior of animal swarms.

In this work a direct search optimization algorithm based on differential and evolutionary methods as well as in the behavior of animals is developed. This algorithm is applied to inverse engineering problems.

In a first stage, the developed algorithm is validated and compared with existing optimization algorithms using a set of composite functions specially design for that purpose. In a second phase, the algorithm is applied to Engineering and Computational Mechanics problems. In this section, the three-truss bar problem and the 120-bar dome truss problem that are solved using the Finite Element Method (FEM) are analyzed. Subsequently, the compression of a cylindrical billet and the plate with a central cut-out problems are analyzed. In these problems, the function to minimize is given by a commercial FEM code. Finally, the algorithm is applied to a constitutive model parameter identification problem. The develop algorithm obtains good results and a good convergence rate.

Conteúdo

Capítulo 1

Introdução

Descreve-se, de forma resumida, em que consiste a optimização estrutural em engenharia e quais os principais métodos de optimização. São expostos os objectivos do presente trabalho.

1.1 Optimização em Engenharia

Presentemente, é de maior importância reduzir o tempo e custo de produção de um qualquer produto. Seguindo esta linha de pensamento, é possível concluir que o desenvolvimento e aperfeiçoamento de ferramentas numéricas para simulação e optimização dos mais variados processos é de grande valor na sociedade actual.

Neste contexto, pode definir-se optimização em engenharia como uma tentativa de maximizar as propriedades desejáveis de um sistema enquanto, simultaneamente, se minimizam as suas características inconvenientes. No caso de uma aplicação estrutural, a optimização consiste, por exemplo, em modificar a geometria de uma dada estrutura, reduzindo o seu peso mas, ao mesmo tempo, garantindo que os valores limite de tensão e/ou deslocamento não são excedidos.

1.1.1 Optimização Estrutural

Em engenharia, os problemas de optimização estrutural podem ser divididos em três classes [Christensen e Klarbring 09]. Tomando $\mathbf{x}$ como as variáveis de projecto de optimização, estas são:

1. **Optimização de tamanho:** quando $\mathbf{x}$ representa um tipo de espessura estrutural, *i.e.*, áreas das secções transversais de elementos barra, distribuição de espessura numa placa, etc;

2. **Optimização de forma:** neste caso $\mathbf{x}$ define a forma ou contorno de uma porção da fronteira de um determinado componente estrutural. Na optimização de forma, a conectividade da estrutura não é modificada, pelo que não se formam novos limites;

3. **Optimização topológica:** Esta é uma forma mais geral de optimização estrutural que permite que a topologia da estrutura a optimizar seja alterada. Num problema discreto, como por exemplo, estruturas articuladas, este tipo de optimização é alcançado tomando as áreas das secções dos elementos barra como variáveis de optimização e permitindo que estas tomem o valor zero (as barras são retiradas da estrutura). Numa estrutura contínua, como uma placa bi-dimensional, é permitido que a espessura de um elemento da placa seja zero.

1.1.2 Identificação de Parâmetros de Modelos Constitutivos

Com o decorrer dos tempos, e devido à exigência da comunidade científica de realizar simulações e obter resultados mais precisos, têm-se desenvolvido técnicas cada vez mais complexas para simular o comportamento dos materiais. Por esta razão, surgem novos modelos de comportamento que visam caracterizar de uma forma mais precisa o material. No entanto, muitas destas leis de comportamento exigem que se determinem um número elevado de parâmetros ajustados ao material cujo comportamento se pretende simular [Andrade-Campos 05].

O desenvolvimento de modelos de materiais inelásticos pode ser dividido em quatro etapas. Inicialmente, um conjunto de testes experimentais deve ser correctamente realizado de modo a identificar o comportamento do material. No segundo é escrito um modelo matemático baseado nos dados experimentais e conhecimentos físicos. O terceiro passo consiste na identificação dos parâmetros do material representados no modelo escolhido. Finalmente, o modelo resultante deve ser testado e validado confrontando os resultados numéricos e/ou analíticos com os resultados experimentais [Kleinermann e Ponthot 03].

Recentemente têm sido desenvolvidas estratégias que recorrem ao Método dos Elementos Finitos (MEF). Nestas, a ideia principal consiste em simular o ensaio experimental realizado, tentando adaptar os parâmetros dos materiais de modo a obter, através da simulação, os mesmos resultados que se obtiveram através dos estudos experimentais.

1.1.3 Métodos de Optimização

A abordagem de optimização é, geralmente, baseada em métodos numéricos, pois estes permitem resolver problemas de grande complexidade e podem ser implementados com relativa facilidade. Nesta abordagem, geram-se inicialmente conjuntos de soluções que vão sendo iterativamente aperfeiçoados [Antoniu e Lu 07]. O processo é terminado quando um critério de paragem (*e.g.* ao ser alcançada uma solução satisfatória ou atingido o numero máximo de iterações) é satisfeito.

Actualmente, existem diversas classificações para os métodos de optimização de acordo com os mecanismos que estes utilizam para obter a solução e com a sua origem. Nos métodos de procura indirecta, os algoritmos movem-se de acordo com a informação fornecida pelo gradiente da função a optimizar. O menor valor de gradiente indica qual a direcção da descida mais abrupta, ou seja, a direcção do mínimo. No entanto esta não é uma propriedade global, o que pode conduzir o algoritmo a um mínimo local [Singiresu 96].

Os métodos de procura directa pertencem a uma classe de métodos de optimização nos quais não é necessário recorrer ao cálculo das derivadas. Dentro destes podemos encontrar os métodos de procura aleatória que se baseiam no uso de números gerados aleatoriamente para encontrar o mínimo de uma função.

Podem ainda dividir-se os métodos de optimização em determinísticos ou estocásticos. Os métodos determinísticos são aqueles que não possuem características aleatórias ou estocásticas. A maioria destes métodos não fornece uma garantia de que o mínimo global irá ser encontrado, a não ser que sejam feitas suposições acerca do problema. Estas suposições podem ser, por vezes, difíceis de provar na realidade.

Por outro lado, os métodos estocásticos diferem dos anteriores pois possuem características estocásticas, permitindo que o método forneça uma garantia probabilística de que o mínimo global será alcançado [Byrd *et al.* 90]. Dentro desta última categoria podemos encontrar, por exemplo, os métodos de Monte Carlo. Estes referem-se a qualquer método que utilize uma sequência de números aleatórios para realizar uma simulação estatística.

Os métodos de optimização podem também ser classificados de acordo a sua origem. Métodos clássicos, algoritmos evolucionários e algoritmos baseados em comportamentos de animais são três das classes abordadas neste trabalho.

Os métodos clássicos são estratégias de optimização nas quais a direcção e tamanho de cada incremento são dados de acordo com a informação fornecida pelo gradiente. Apesar de possuírem baixos tempos de computação, estes métodos podem convergir e ficar estagnados em mínimos locais quando o problema em estudo é complexo. Os métodos de Newton e de Levenbeg-Marquardt são exemplos de metodologias clássicas de optimização.

Os algoritmos evolucionários são métodos de optimização baseados em mecanismos de evolução biológica como reprodução, mutação, recombinação e selecção. Estes recorrem, geralmente, a uma população de partículas em que cada uma delas representa uma possível solução. Dentro desta categoria podemos encontrar os Algoritmos Genéticos e Evolução Diferencial.

Nos algoritmos baseados no comportamento de animais, uma população de partículas é criada no domínio do problema. Cada uma dessas partículas move-se no domínio através da partilha de informação com as restantes partículas da população. O algoritmo de Optimização por Bandos de Partículas e Optimização com Colónias de Formigas são exemplos de algoritmos baseados no comportamento de animais.

1.2 Objectivos

Neste trabalho pretende desenvolver-se um algoritmo de optimização de procura directa baseado em métodos diferenciais, evolucionários e em comportamentos de animais. Este algoritmo será, posteriormente, validado com recurso a um conjunto de funções de teste e, por fim, será aplicado a problemas específicos de Engenharia Mecânica. Durante esta última fase o algoritmo desenvolvido será, ainda, comparado com diversos algoritmos existentes.

1.3 Guia de Leitura

O presente trabalho encontra-se dividido em 6 capítulos. Seguidamente, é exposto de forma sucinta o conteúdo de cada capítulo.

Capítulo 1 - Descreve-se de forma resumida em que consiste a optimização estrutural em engenharia e quais os principais métodos de optimização. São expostos os objectivos do presente trabalho.

Capítulo 2 - Realiza-se uma exposição teórica dos conteúdos necessários. Descreve-se a formulação do Método dos Elementos Finitos e programas que implementam esta metodologia. Por fim, descreve-se o processo de definição de formas não regulares.

Capítulo 3 - Apresentam-se diferentes métodos de optimização. Descreve-se a formulação dos métodos e a influência dos diversos parâmetros operacionais. Incluí-se uma revisão bibliográfica dos métodos mais relevantes no âmbito do presente trabalho. É realizada uma breve introdução a alguns programas de optimização.

Capítulo 4 - Descreve-se de forma pormenorizada o algoritmo desenvolvido e quais as diferentes metodologias implementadas.

Capítulo 5 - Realiza-se o estudo para determinar quais os valores ideais dos diferentes parâmetros operacionais utilizados pelo algoritmo desenvolvido. Compara-se o algoritmo desenvolvido com algoritmos existentes recorrendo a funções de teste matemáticas. Aplica-se o algoritmo desenvolvido a diversos problemas de optimização em Engenharia Mecânica.

Capítulo 6 - Apresentam-se as conclusões gerais e discutem-se os resultados obtidos na realização deste trabalho.

Capítulo 2

Fundamentos

Realiza-se uma exposição teórica dos conteúdos necessários. Descreve-se a formulação do Método dos Elementos Finitos e programas que implementam esta metodologia. Por fim, descreve-se o processo de definição de formas não regulares.

2.1 Formulação Matemática do Problema de Optimização

Antes de ser realizada a operação de optimização, o problema necessita de ser correctamente formulado. Matematicamente, as propriedades de um sistema a ser optimizado podem ser definidas através de uma função, denominada por função objectivo. A referida função relaciona as propriedade do sistema através de D parâmetros que o definem de acordo com

$$F = f(x_1, x_2, ..., x_D). \tag{2.1}$$

O problema de optimização simples consiste em determinar as variáveis $x_1, x_2, ..., x_D$, contidas num intervalo $x_{i,\min} \leq x_i \leq x_{i,\max}$, com $i = 1, 2, ..., D$, de forma a minimizar o valor de F.

Em muitos problemas reais, as variáveis de optimização não podem ser escolhidas arbitrariamente pois estas necessitam de satisfazer certos requerimentos funcionais. As restrições que necessitam de ser satisfeitas para produzir um projecto aceitável são chamados de restrições de projecto. As restrições que representam limitações do comportamento do sistema são chamadas de restrições funcionais. Podem ainda existir restrições geométricas que estão ligadas às limitações físicas de fabrico e transporte .

De uma forma geral, o problema de optimização com restrições pode ser escrito como

$$\text{minimizar } F = f(\mathbf{x}) \tag{2.2}$$

sujeito a

$$x_{i,\min} \leq x_i \leq x_{i,\max} \tag{2.3}$$

$$h_j(\mathbf{x}) = 0, \text{para } j = 1, .., J \tag{2.4}$$

$$g_k(\mathbf{x}) \leq 0, \text{para } k = 1, ..., K. \tag{2.5}$$

Nestes problemas existem diversas formas de lidar com as restrições dadas pelas equações 2.4 e 2.5. Uma das metodologias existentes consiste em recorrer a uma função penalidade. Se uma solução gerada não obedece a alguma das restrições dadas, uma função, de valor positivo

e proporcional ao número de restrições infringidas, é adicionada à função objectivo. Deste modo garante-se que esta solução não seja solução do problema. Recorrendo a esta abordagem, o problema pode ser considerado como sendo não constrangido, sendo representado por

$$F = f(\mathbf{x}) + P(R, h(\mathbf{x}), g(\mathbf{x})), \tag{2.6}$$

em que R é um conjunto de parâmetros de penalidade e P um termo de penalidade.

2.2 O Método dos Elementos Finitos

Em engenharia, muitos fenómenos físicos, tais como transferência de calor e escoamentos de fluidos, podem ser descritos através de equações diferenciais parciais. Devido à sua elevada complexidade, a resolução destas através de métodos analíticos clássicos é uma tarefa quase impossível. Desenvolvido em 1950 pela industria aeronáutica, o Método dos Elementos Finitos (MEF) é um método numérico aproximado para resolução deste tipo de equações [Fish e Belytschko 07].

A ideia principal do MEF consiste em realizar uma discretização da geometria da estrutura em estudo em elementos com volume ou área finita, que se encontram ligados uns aos outros através de nós. Ao conjunto de todos os elementos dá-se o nome de malha. O MEF fornece a solução do problema através da resolução de sistemas de equações, recorrendo, normalmente, a meios computacionais. De forma geral, a exactidão da solução melhora com o aumento do número de elementos. No entanto, o tempo de computação e, consequentemente, o custo crescem.

2.2.1 Descrição do Comportamento de um Elemento Barra

Uma estrutura articulada consiste num conjunto de elementos esbeltos, usualmente chamados de barras. Assume-se que elementos barra são suficientemente delgados de modo a ser possível desprezar as forças de corte e torção. Deste modo, as únicas forças que actuam sobre estes elementos são forças axiais.

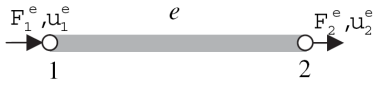


Figura 2.1: Elemento barra alinhado com o eixo x [Fish e Belytschko 07]

Considerando o elemento barra posicionado sobre o eixo x da figura 2.1, e assumindo um material linear elástico que obedece à Lei de Hooke e que apenas pode suportar cargas axiais, a deformação axial da barra é dada por

$$\varepsilon^e = \frac{\Delta l^e}{l^e} = \frac{u_2^e - u_1^e}{l^e}, \tag{2.7}$$

em que u_1^e e u_2^e são, respectivamente, os deslocamentos do nó 1 e 2 do elemento e l^e o comprimento da barra. É, então, possível obter a tensão no elemento através de

$$\sigma^e = E^e \varepsilon^e = E^e \frac{u_2^e - u_1^e}{l^e}, \tag{2.8}$$

em que E^e é o Módulo de Young do elemento e.

Considerando que a barra se encontra em equilíbrio estático, pode escrever-se

$$F_1^e + F_2^e = 0, \tag{2.9}$$

e consequentemente

$$F_1^e = -F_2^e = A^e E^e \frac{u_2^e - u_1^e}{l^e} \tag{2.10}$$

onde A^e é a área da secção transversal do elemento.

Reescrevendo as equações anteriores na forma matricial obtém-se

$$\begin{Bmatrix} F_1^e \\ F_2^e \end{Bmatrix} = \frac{E^e A^e}{l^e} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{Bmatrix} u_1^e \\ u_2^e \end{Bmatrix} \tag{2.11}$$

ou de forma mais compacta

$$\mathbf{F}^e = \mathbf{k}^e \mathbf{u}^e \tag{2.12}$$

em que $\mathbf{k}^e$ corresponde à matriz rigidez do elemento barra e. A equação 2.11 descreve a relação entre as forças e deslocamentos nodais de um elemento.

2.2.2 Descrição de Comportamento de um Elemento Barra no Plano

Neste tipo de problemas, cada grau de liberdade do elemento terá de ser decomposto em dois graus de liberdade dependentes, correspondentes às projecções do deslocamento axial da barra em duas direcções ortogonais. Por essa razão, torna-se necessário definir, para além do sistema de eixos local do elemento, um sistema de eixos global de forma a ser possível expressar o efeito aditivo das forças das extremidades de elementos que concorrem em determinado nó.

No caso do elemento barra plano da figura 2.2, o tamanho deste é dado por

$$L^e = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}, \tag{2.13}$$

e o ângulo de inclinação obtém-se através de

$$\beta = \tan^{-1}\left(\frac{y_2 - y_1}{x_2 - x_1}\right). \tag{2.14}$$

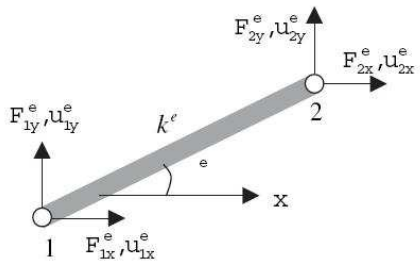


Figura 2.2: Elemento barra plano [Fish e Belytschko 07]

Na equação 2.12, $\mathbf{F}^e$ passa a ser escrito como

$$\mathbf{F}^e = \begin{Bmatrix} F_{1x}^e \\ F_{1y}^e \\ F_{2x}^e \\ F_{2y}^e \end{Bmatrix}, \tag{2.15}$$

e os deslocamentos nodais

$$\mathbf{u}^e = \begin{Bmatrix} u_{1x}^e \\ u_{1y}^e \\ u_{2x}^e \\ u_{2y}^e \end{Bmatrix}. \tag{2.16}$$

A matriz de rigidez $\mathbf{k}^e$ passa a ser escrita como

$$\mathbf{k}^e = \frac{E^e A^e}{l^e} \begin{bmatrix} \cos^2\beta & \sin\beta\cos\beta & -\cos^2\beta & -\sin\beta\cos\beta \\ \sin\beta\cos\beta & \sin^2\beta & -\sin\beta\cos\beta & -\sin^2\beta \\ -\cos^2\beta & -\sin\beta\cos\beta & \cos^2\beta & \sin\beta\cos\beta \\ -\sin\beta\cos\beta & -\sin^2\beta & \sin\beta\cos\beta & \sin^2\beta \end{bmatrix}. \tag{2.17}$$

2.2.3 Estruturas Articuladas Planas

As análises anteriores referem-se apenas ao cálculo de um único elemento. No entanto, em problemas reais, as estruturas a estudar podem ser mais complexas, possuindo uma grande quantidade de elementos. Torna-se então necessário criar uma matriz rigidez global que represente o sistema completo. O procedimento para construção do sistema de equações globais é o mesmo para qualquer tipo de problema e para qualquer número e tipo de elementos usados.

O procedimento de assemblagem das matrizes e vectores dos elementos baseia-se na condição de compatibilidade, isto é, nos nós nos quais os elementos estão ligados, o valor da variável em estudo (deslocamento, temperatura, etc.) é o mesmo para todos os elementos unidos nesse nó.

Considerando n como o número de elementos, a matriz rigidez global é dada por

$$\mathbf{K} = \sum_{e=1}^{n} \mathbf{k}^e, \tag{2.18}$$

e o vector das forças nodais obtém-se fazendo

$$\mathbf{F} = \sum_{e=1}^{n} \mathbf{F}^e. \tag{2.19}$$

O processo de assemblagem demonstrado é simples, pelo que pode ser implementado computacionalmente com facilidade.

2.2.4 Programas Baseados no Métodos dos Elementos Finitos

Existem diversos programas comerciais e de fonte aberta[1] que recorrem ao MEF para resolver problemas ou que auxiliam o pré e pós-processamento de modelos de elementos finitos. Os programas usados para resolução e/ou visualização dos problemas desenvolvidos neste trabalho são descritos em seguida.

O Programa ABAQUS

O programa ABAQUS é um software comercial, criado pela Hibbitt, Karlsson & Sorensen Inc. e actualmente em desenvolvimento contínuo pela SIMULIA, que utiliza o Método dos Elementos Finitos [Hibbit *et al.* 08]. Este consiste em dois módulos principais de análise : ABAQUS/Standard e ABAQUS/Explicit. O ABAQUS/Standard é um produto geral de análise quasi-estática que recorre a um esquema de integração implícito e que permite resolver uma variada gama de problemas. O ABAQUS/Explicit é um produto específico que utiliza uma formulação dinâmica explicita dos elementos finitos. Este é principalmente usado em simulações de impacto e em problemas em que a componente inercial não pode ser desprezada.

O software possui ainda dois módulos gráficos - ABAQUS/CAE e ABAQUS/Viewer. O módulo ABAQUS/CAE é um ambiente gráfico para o ABAQUS que permite a criação de modelos (geometria, materiais, propriedade mecânicas, malha, etc) e a partir destes cria um ficheiro de

[1] do inglês *open source*.

extensão *.inp* que é interpretado pelos módulos de análise (Standard ou Explicit). Estes ficheiros são do tipo ASCII e podem ser parametrizados. Estas funcionalidades permitem que os ficheiros sejam modificados directamente pelo utilizador e permite a integração com outras aplicações. O ABAQUS/Viewer é um sub-conjunto de visualização do ABAQUS/CAE que inclui unicamente a funcionalidade de pós-processamento.

O Programa FRAN

Baseado no algoritmo descrito por Smith e Griffiths [Smith e Griffiths 88], o programa FRAN[2] permite realizar a análise de estruturas compostas por elementos barra com diferentes propriedades. Neste programa fornece-se à entrada as coordenadas de cada nó, conectividades dos elementos, cargas impostas sobre a estrutura e deslocamentos prescritos. O FRAN realiza a assemblagem da matriz rigidez e resolve o sistema de equações. À saída fornece as forças axiais em cada elemento e os deslocamentos de cada nó. O programa exporta, simultaneamente, um ficheiro com formato reconhecido pelo programa GiD.

O Programa GiD

O programa GiD[3], desenvolvido pela CIMNE[4], é um interface gráfico interactivo usado para definição, preparação e visualização de todos os dados relacionados com uma simulação numérica. Estes dados incluem a definição da geometria, materiais, condições e informação acerca da solução. O programa permite ainda correr simulações numéricas e visualizar os resultados da análise [GiD Reference Manual].

2.2.5 Aplicações

O campo de aplicação do métodos dos elementos finitos em engenharia é muito vasto [Fish e Belytschko 07]. Exemplos de algumas aplicações são:

- Análise de tensões e análise térmica em componentes industriais como *chips* electrónicos, válvulas e motores;
- Análise de vibrações em componentes automóveis;
- Simulação de processos tecnológicos para previsão de fenómenos físicos, como o retorno elástico, e defeitos (*e.g.* rugas);
- Simulação de impacto de automóveis, comboios e aviões;
- Simulação de impactos balísticos.

2.3 Definição de Formas Não-Regulares

Muitos problemas possuem formas não regulares que dificultam a sua definição através de um número finito de parâmetros. Considerando que a forma a determinar é dada por uma função f, existem actualmente diversos métodos para interpolar valores de f num intervalo $[a, b]$ a partir de um número finito de pontos de dados.

A interpolação de Lagrange é um método clássico no qual é usado um polinómio de grau n para interpolar $n + 1$ pontos a partir das suas coordenadas. A qualidade desta aproximação pode

[2] O programa FRAN foi gentilmente cedido pelo Eng. J. A. Oliveira.
[3] http://gid.cimne.upc.es/
[4] *International Center for Numerical Methods in Engineering.*

ser melhorada se se dispuser de informação sobre as derivadas da função, podendo aumentar-se o grau do polinómio interpolador. Esta técnica é denominada por interpolação de Hermite.

Uma outra maneira simples e flexível de interpolar os valores de f consiste em dividir o intervalo $[a, b]$ em vários sub-intervalos e realizar, em cada um, uma aproximação através de polinómios de baixo grau. A estes polinómios aproximados dão-se o nome de splines, sendo os extremos de cada intervalo denominado por nós.

Pode afirmar-se que um spline de grau m, com $m > 1$, é uma função que é constituída por polinómios de grau m ou inferior em cada sub-intervalo. De modo a formarem uma função contínua (suave), deve garantir-se que as derivadas de ordem $m-1$ são contínuas nos nós interiores [Suli e Mayers 03].

2.3.1 Splines Quadráticos

De acordo com Chapra e Canale [Chapra e Canale 90], de forma a garantir que as derivadas de ordem m são contínuas nos nós, um spline de ordem $m+1$ ou maior deverá ser usado. Deste modo, um spline quadrático possui primeiras derivadas contínuas nos nós.

O objectivo dos splines quadráticos consiste em derivar um polinómio de segundo grau para cada intervalo entre pontos. Cada um desses polinómios pode ser representado por

$$f_i(x) = a_i x^2 + b_i x + c_i, \text{ com } i = 0, 1, 2, ..., n, \tag{2.20}$$

em que n é o número de intervalos, correspondendo a $n+1$ pontos de dados. De acordo com a equação 2.20, de modo a ser possível determinar todos os $3n$ coeficientes, é necessário o mesmo número de equações ou condições. Essas condições são:

1. O valor da função tem de ser igual nos nós interiores. Esta equação pode ser representada como

$$a_{i-1}x_{i-1}^2 + b_{i-1}x_{i-1} + c_{i-1} = f(x_{i-1}), \tag{2.21}$$

$$a_i x_{i-1}^2 + b_i x_{i-1} + c_i = f(x_{i-1}), \tag{2.22}$$

   para $i = 2, 3, ..., n$.

2. A primeira e última funções devem passar pelos pontos extremos. Estas duas equações podem ser representadas por

$$a_1 x_0^2 + b_1 x_0 + c_1 = f(x_0), \tag{2.23}$$

$$a_n x_n^2 + b_n x_0 + c_n = f(x_n). \tag{2.24}$$

3. A primeira derivada em cada nó interior deve ser igual. A primeira derivada da equação 2.20 é dada por

$$f'(x) = 2ax + b,$$

   pelo que esta condição pode ser, de uma forma geral, expressa da seguinte forma

$$2a_{i-1}x_{i-1} + b_{i-1} = 2a_i x_{i-1} + b_i. \tag{2.25}$$

4. Caso não se possua nenhuma informação acerca da função ou suas derivadas, é necessário realizar uma escolha arbitrária de modo a ser possível resolver o sistema de equações. Com este fim, assume-se que a segunda derivada no primeiro ponto é zero. Uma vez que a segunda derivada da equação 2.20 corresponde a $2a$, esta condição pode ser expressa matematicamente através de

$$a_1 = 0. \tag{2.26}$$

   Geometricamente, esta condição representa uma recta que une os dois primeiros pontos.

2.3.2 Interpolação de Lagrange

Interpolação de Lagrange, ou polinomial, é uma técnica clássica de interpolação que permite determinar um polinómio de grau n que passa por $n+1$ pontos [Suli e Mayers 03]. A fórmula de interpolação polinomial é dada por

$$f(x) = \sum_{i=1}^{n} f(x_i) P_i^L(x), \tag{2.27}$$

em que $f(x_i)$ são os valores conhecidos da função e $f(x)$ o valor a determinar. O polinómio de Lagrange P_i^L é um polinómio de grau $n-1$ que possui o valor 1 quando $x = x_i$ e 0 para todo $x_{i \neq j}$. Matematicamente pode ser expresso da seguinte forma

$$P_i^L(x) = \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)}. \tag{2.28}$$

Capítulo 3

Algoritmos de Optimização

Apresentam-se diferentes métodos de optimização. Descreve-se a formulação dos métodos e a influência dos diversos parâmetros operacionais. Incluí-se uma revisão bibliográfica dos métodos mais relevantes no âmbito do presente trabalho. É realizada uma breve introdução a alguns programas de optimização.

3.1 Evolução Diferencial

Inicialmente introduzido por Storn e Price [Storn e Price 95] em 1995, o método de Evolução Diferencial (DE[1]) é um algoritmo evolucionário simples e de fácil utilização que cria novas soluções através da combinação do progenitor com vários outros membros da mesma população. O DE alia operadores aritméticos simples com operadores de recombinação, mutação e cruzamento para evoluir de uma população inicial gerada aleatoriamente até uma solução final através da adaptação do seu passo de procura ao longo do processo evolucionário. Este processo permite a obtenção de um compromisso entre exploração global e procura local.

As vantagens do algoritmo DE residem na sua estrutura simples, facilidade de utilização, propriedades de convergência, qualidade da solução e robustez.

3.1.1 Formulação do Algoritmo

Neste método de optimização paralelo de procura directa, uma população de NP vectores de dimensão D é inicialmente criada dentro do universo de procura do problema representados por

$$x_{i,G}, \text{ com } i = 1, 2, ..., NP, \tag{3.1}$$

onde G corresponde ao número da geração da população e NP não varia durante o processo. A primeira geração é criada aleatoriamente e deverá cobrir de forma uniforme o espaço de procura.

Em cada geração, os vectores candidatos são obtidos através da adição de uma diferença ponderada entre dois membros da população (vectores diferença) a um terceiro (vector base). Se o candidato criado possuir um valor da função objectivo inferior ao de um outro membro predeterminado da população (vector alvo), o vector candidato irá substituir o membro com o qual foi comparado na geração seguinte. Caso contrário mantém-se o membro antigo. O vector

[1] do inglês *Differential Evolution*.

com o qual o membro foi comparado poderá, ou não, pertencer ao processo de criação de vectores mencionado anteriormente. Adicionalmente, o melhor vector $x_{\text{best},G}$ é avaliado a cada geração.

Na figura 3.1 apresenta-se a sequência do algoritmo básico. Na secções seguintes são descritas mais detalhadamente cada uma das diferentes etapas do algoritmo DE.

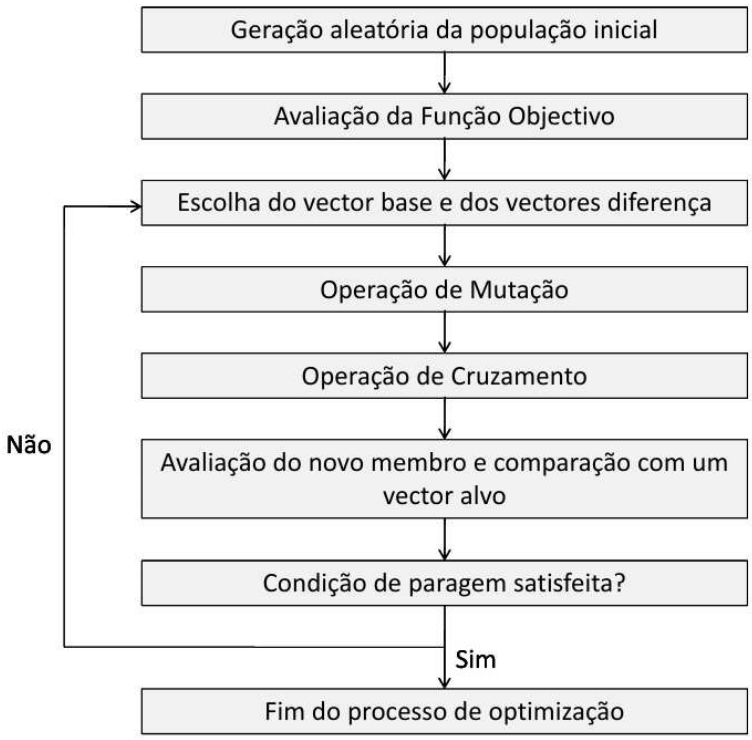


Figura 3.1: Esquema do algoritmo básico de Evolução Diferencial

Inicialização da População

Para inicializar a população é necessário, primeiramente, definir os limites do problema a estudar. Cada uma das D dimensões do problema necessita de estar contida dentro de um intervalo cujos limites inferior e superior são definidos, respectivamente, por $x_{j,\min}$ e $x_{j,\max}$ com $j = 1, 2, ..., D$. Após a definição dos limites a geração inicial é criada de acordo com

$$x_{j,\min} < \; x_{j,i,0} = \text{rand}_j(0,1)(x_{j,\max} - x_{j,\min}) + x_{j,\min} \; < x_{j,\max}, \tag{3.2}$$

em que $\text{rand}_j(0,1)$ é um número gerado aleatoriamente pertencente ao intervalo $[0,1]$. O índice j indica que um novo número aleatório deve ser gerado para cada uma das dimensões do vector.

Mutação

O objectivo da mutação é possibilitar uma procura mais diversificada no espaço de procura, bem como direccionar os vectores existentes, com uma variação adequada dos parâmetros, numa direcção que irá conduzir a melhores resultados. Este processo mantém ainda a procura robusta e explora novas áreas nos domínios do espaço de possíveis soluções.

Uma vez inicializado, o algoritmo realiza operações de mutação e, deste modo, para cada vector $x_{i,G}$ é criado um vector perturbado $v_{i+1,G}$ dado por

$$v_{i,G+1} = x_{r_1,G} + F(x_{r_2,G} - x_{r_3,G}). \tag{3.3}$$

O factor de escala da solução F é um valor real positivo que controla a amplificação da variação diferencial, isto é, controla a razão à qual a população evolui. r_2 e r_3 são números inteiros escolhidos aleatoriamente do intervalo $\{1, 2, ..., NP\}$ diferentes do índice corrente i. A escolha do valor de r_1 depende da estratégia de mutação adoptada, podendo ser, também ele, um número inteiro retirado do intervalo dado, ou poderá, por exemplo, corresponder ao melhor membro da geração anterior ($x_{best,G}$).

Numa fase inicial do processo de optimização, a perturbação criada pelo processo de mutação é elevada pois os membros da população encontram-se afastados uns dos outros. À medida que o processo evolucionário avança em direcção ao estágio final, a população converge para uma zona pequena e, consequentemente, a perturbação diminui. Deste modo, o passo de procura beneficia o algoritmo ao realizar uma procura global com um passo elevado no início do processo evolucionário e um refinamento deste num estágio mais avançado.

Cruzamento

Na operação de cruzamento são criados vectores candidatos $u_{i,G+1}$ a partir de valores de parâmetros copiados de dois vectores diferentes. O tipo de cruzamento depende do modelo escolhido, podendo ser binomial ou exponencial.

Num cruzamento do tipo binomial, o vector candidato é obtido da seguinte forma

$$u_{j,i,G+1} = \begin{cases} v_{j,i,G+1}, & \text{se } \text{rand}_j(0,1) \leq CR \\ x_{j,i,G}, & \text{se } \text{rand}_j(0,1) > CR \end{cases} . \tag{3.4}$$

O coeficiente probabilidade de cruzamento $CR \in [0, 1]$ é o valor que define qual a fracção de parâmetros que serão copiados do vector mutante e representa a probabilidade do vector candidato herdar os valores dos parâmetros do vector perturbado. Novamente, $\text{rand}_j(0, 1)$ é um número aleatório pertencente ao intervalo $[0, 1]$ e $j = 1, 2, ..., D$ representa qual o parâmetro afectado, ou não, pela operação de cruzamento (ver figura 3.2).

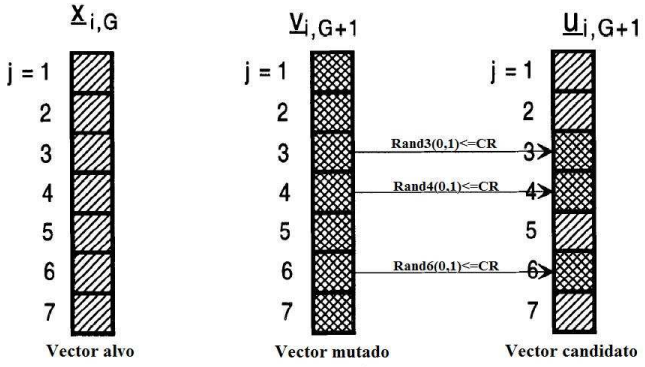


Figura 3.2: Ilustração do processo de cruzamento binomial para $D = 7$

No cruzamento do tipo exponencial, é escolhida uma posição aleatória n no intervalo $\{1, 2, ..., D\}$. Seguidamente, realiza-se o cruzamento nas L posições seguintes até que $\text{rand}_L(0, 1)$ seja inferior a CR, onde $\text{rand}_L(0, 1)$ é um número aleatório no intervalo [0,1] gerado para cada elemento do intervalo $\{n, n + 1, ...D\}$.

As principais diferenças entre os tipos de cruzamento acima mencionados são:

- L elementos adjacentes são trocados na variante exponencial, enquanto no cruzamento binomial as trocas estão aleatoriamente distribuídas no intervalo $\{1, 2, ..., D\}$

- No cruzamento binomial a relação entre a probabilidade de cruzamento e CR é linear. No cruzamento do tipo exponencial esta relação é não linear e esta não-linearidade aumenta com o crescimento das dimensões do problema.

Segundo Price [Price e Storn 09],o tipo de cruzamento não é de tão grande importância. No entanto afirma que o cruzamento binomial nunca é pior que o exponencial.

Selecção da nova população

Para decidir se o vector candidato $u_{i,G+1}$ fará parte da geração $G+1$, este é comparado com o vector progenitor $x_{i,G}$. Se $u_{i,G+1}$ possuir um valor de função objectivo inferior ao do $x_{i,G}$, então o valor de $x_{i,G+1}$ é substituído por $u_{i,G+1}$, caso contrário o valor $x_{i,G}$ é mantido. Este critério de selecção é chamado de critério elitista, uma vez que um vector apenas é aceite se este diminuir o valor da função objectivo.

Pseudo-código

Na tabela 3.1 encontra-se representado o algoritmo em pseudo-código no qual se demonstra o funcionamento do algoritmo básico de Evolução Diferencial.

Tabela 3.1: Pseudo-código do método de Evolução Diferencial

```
!Criação aleatoria da população inicial-----------------
initialize(x)
!Minimo global-----------------------------------------
initialize(best_value)

do k=1,itermax
   do i=1, NP
   !Escolha de 3 números aleatorios----------------------
      r1=rand(1,NP);r2=rand(1,NP);r3=rand(1,NP)
      r1/=r2/=r3/=i

      !Passo de mutação-------------------------------
      v(i,:)=x(r1,:)+F*(x(r2,:)-x(r3,:))

      !Passo de cruzamento binomial--------------------
      do j=1,D
         if (rand(0,1)<=CR) u(i,j)=v(i,j)
      end do

      !Avaliação do novo membro------------------------
      value_temp=evaluate(u)
      value_old=evaluate(x)

      !Comparação do novo membro-----------------------
      if (value_temp < value_old) x=u
      if (value_temp < value_best) value_best=value_temp
   end do
end do
```

3.1.2 Parâmetros Operacionais

O DE possui três parâmetros chave: tamanho da população (NP), factor de escala da solução (F) e probabilidade de cruzamento (CR). Uma boa configuração destes parâmetros permite obter um

bom compromisso entre exploração global e procura local de modo a aumentar a velocidade de convergência e robustez do método de procura [Tang *et al.* 08].

Tamanho da População

De acordo com estudos realizados por Storn e Price [Storn e Price 97], uma escolha razoável para NP encontra-se entre $5 \times D$ e $10 \times D$, mas NP deverá ser no mínimo 4 de modo a assegurar que o algoritmo contenha vectores diferentes para poder ser efectivo.

Factor de Escala da Solução

De uma forma geral, o processo de selecção tende a reduzir a diversidade da população, enquanto o processo de mutação a aumenta [Price *et al.* 05]. De modo a evitar uma convergência prematura é importante que o factor de escala da solução F tenha magnitude suficiente, encontrando-se, normalmente, no intervalo $[0, 1]$. Embora possa ser usado $F > 1$, as soluções obtidas tendem a ser inferiores e computacionalmente menos eficientes quando comparado com valores de $F < 1$. Para valores de $F = 1$, as combinações de vectores tornam-se indistinguíveis, isto é, não se consegue definir qual o vector a ser perturbado e quais os vectores usados na subtracção, podendo conduzir a uma convergência imprevisível.

Segundo Zahari [Zahari 02], o valor crítico de F pode ser dado por

$$F_{crit} = \sqrt{\frac{1 - CR/2}{NP}}, \tag{3.5}$$

no qual F_{crit} estabelece um limite inferior de F no sentido em que valores mais baixos induzem convergência mesmo numa área plana da função objectivo.

Prova-se que o uso de um valor de F constante durante o processo de optimização é efectivo [Price *et al.* 05], no entanto, gerar este parâmetro aleatoriamente oferece alguns benefícios. Transformar F num parâmetro aleatório permite alargar o espectro dos vectores diferenciais além das possibilidades permitidas pela combinação de vectores. Uma tal melhoria pode ser útil quando a população é pequena e/ou simetricamente distribuída, pois, sem acesso a uma distribuição por mutação suficientemente diversificada, o algoritmo pode estagnar. Quando estagnado, o DE não consegue encontrar melhores soluções pois nenhuma combinação de vectores e de diferenças entre vectores conduz a soluções aperfeiçoadas.

Probabilidade de Cruzamento

Apesar de ser um mediador do processo de cruzamento, a probabilidade de cruzamento CR também pode ser interpretada como um grau de mutação, isto é, a probabilidade aproximada de um parâmetro ser herdado de um mutante. No algoritmo DE, o número médio de parâmetros mutados para um dado valor de CR depende do modelo escolhido, mas, em ambos, um valor de CR baixo corresponde a um baixo grau de mutação.

Em resumo, a principal função de CR é providenciar diversidade adicional ao grupo de vectores candidatos, especialmente para valores de CR próximos de 1 em que as perdas de performance associados a baixos níveis de mutação são minimizados.

De acordo com Storn e Price [Storn e Price 97], uma boa primeira escolha para CR é 0.1. Contudo, como um valor grande de CR usualmente melhora a velocidade de convergência, o uso inicial de $CR = 0.9$ ou $CR = 1.0$ é apropriado de modo a verificar se é possível alcançar a solução rapidamente.

3.1.3 Estratégias e Modificações Aplicadas em Evolução Diferencial

Como referido anteriormente, existem diferentes estratégias que podem ser usadas no algoritmo. Storn e Price [Storn e Price 95] propõem duas estratégias, sendo, posteriormente, propostas outras oito adicionais [Onwubolu e Davendra 09]. A escolha da estratégia a usar depende principalmente do problema que se pretende optimizar. Uma estratégia que apresente os melhores resultados num dado problema poderá não funcionar correctamente quando aplicado noutro problema [Babu e Jehan 03]. As estratégias variam nas soluções a ser perturbadas, número de pares de vectores escolhidos para perturbação e no tipo de cruzamento a usar. As estratégias propostas por Price e Storn definem-se por:

1. DE/best/1/exp: $v_{i,G+1} = x_{\text{best},G} + F.(x_{r_1,G} - x_{r_2,G})$
2. DE/rand/1/exp: $v_{i,G+1} = x_{r_1,G} + F.(x_{r_2,G} - x_{r_3,G})$
3. DE/rand-best/1/exp: $v_{i,G+1} = x_{i,G} + \lambda.(x_{best,G} - x_{r_1,G}) + F.(x_{r_1,G} - x_{r_2,G})$
4. DE/best/2/exp: $v_{i,G+1} = x_{\text{best},G} + F.(x_{r_1,G} - x_{r_2,G} + x_{r_3,G} - x_{r_4,G})$
5. DE/rand/2/exp: $v_{i,G+1} = x_{r_5,G} + F.(x_{r_1,G} - x_{r_2,G} + x_{r_3,G} - x_{r_4,G})$
6. DE/best/1/bin: $v_{i,G+1} = x_{\text{best},G} + F.(x_{r_1,G} - x_{r_2,G})$
7. DE/rand/1/bin: $v_{i,G+1} = x_{r_1,G} + F.(x_{r_2,G} - x_{r_3,G})$
8. DE/rand-best/1/bin: $v_{i,G+1} = x_{i,G} + \lambda.(x_{\text{best},G} - x_{r_1,G}) + F.(x_{r_1,G} - x_{r_2,G})$
9. DE/best/2/bin: $v_{i,G+1} = x_{\text{best},G} + F.(x_{r_1,G} - x_{r_2,G} + x_{r_3,G} - x_{r_4,G})$
10. DE/rand/2/bin: $v_{i,G+1} = x_{r_5,G} + F.(x_{r_1,G} - x_{r_2,G} + x_{r_3,G} - x_{r_4,G})$

A convenção usada é DE/x/y/z. DE significa Evolução Diferencial, x representa qual o vector a ser modificado, y o número de vectores diferença considerados para perturbação de x, e z representa o modelo de cruzamento usado (exp: exponencial, bin: binomial).

Para além de todas as estratégias apresentadas, existem ainda inúmeros estudos e modificações aplicados ao algoritmo de Evolução Diferencial básico. Estes visam melhorar aspectos como a capacidade de exploração do universo de procura, velocidade de convergência, entre outros.

Noman e Iba [Noman e Iba 03] propuseram um novo método de cruzamento baseado numa procura local denominado por Refinamento do Melhor Indivíduo (FIR[2]). Este esquema acelera o DE ao melhorar a sua capacidade de procura através da exploração da vizinhança do melhor membro da população em sucessivas gerações.

Zhenyu *et al.* [Zhenyu *et al.* 06] estudaram o efeito da introdução de mutação caótica aplicada ao pior membro da população. Neste estudo considerou-se ainda um factor de agregação e um factor de evolução de velocidade para controlar os parâmetros F e CR, respectivamente.

Após o estudo do comportamento das partículas durante o processo de mutação e da influência de F, Ali [Ali 07] propõe uma regra de cruzamento, denominada regra de cruzamento preferencial. Esta aproximação permite diminuir o número de avaliações, e o tempo de CPU necessário, bem como aumentar o número de ensaios bem sucedidos.

Yang *et al.* ([Yang *et al.* 08-1] e [Yang *et al.* 08-2]) estudaram vários mecanismos auto-adaptativos para melhorar o algoritmo DE. Nestes estudos a estratégia de mutação os parâmetros CR e F são adaptados de forma dinâmica com o decorrer do processo de optimização.

[2]do inglês *Fittest Individual Refinement*.

Gong *et al.* [Gong *et al.* 08] incorporaram o método de concepção ortogonal no DE de modo a aumentar a razão de convergência. Este método foi aplicado não só durante a geração da população inicial, mas também durante o processo de cruzamento, tornando o algoritmo mais robusto.

Todos os estudos anteriormente referidos tornam o algoritmo mais robusto, conduzindo a novas aplicações do mesmo. Actualmente, o algoritmo de Evolução Diferencial pode ser usado em diversos problemas de optimização. Estes são, por exemplo, problemas de calendarização, rotas de veículos, problema da mochila e planeamento de sequências de trabalho para robôs [Onwubolu e Davendra 09]. O DE também é aplicado na aprendizagem de redes neuronais, na determinação de parâmetros de transferência de calor, optimização de processos químicos não lineares, reacções alcalinas e sistemas de bombeamento de água, caracterização cristalográfica, entre outros [Price e Storn 09]. Este algoritmo foi ainda aplicado com sucesso a problemas de optimização topológica [Wu *et al.* 07].

3.2 Optimização por Bandos de Partículas

Introduzido pela primeira vez por Kennedy e Eberhart em 1995 [Kennedy e Eberhart 09], o método de Optimização por Bandos de Partículas (PSO[3]) é um método de fácil implementação que se baseia na comunicação e interacção social de bandos de pássaros e cardumes de peixes. Neste tipo de bandos os membros tendem a seguir o líder, sendo este definido como o membro que possui melhor desempenho. No PSO, é usado um determinado número de agentes (partículas) que constituem o bando que se desloca dentro do espaço de procura até encontrar a melhor solução. Cada partícula representa uma possível solução do problema de optimização e é tratada como sendo um ponto num espaço D-dimensional que ajusta a sua velocidade de *voo* de acordo com a sua própria experiência, bem como a experiência de outras partículas.

Cada partícula guarda as coordenadas da sua melhor solução encontrada até ao momento, ou seja, o melhor valor pessoal ($p$best). Além dos valores pessoais, o algoritmo mantém ainda o melhor valor obtido por todas as partículas na sua vizinhança. Este é chamado de melhor valor global ($g$best).

O conceito básico do PSO consiste em acelerar cada partícula na direcção do seu $p$best e do $g$best usando um factor de aceleração aleatório diferente para cada um dos termos. Neste algoritmo a população é mantida, isto é, durante o processo de optimização as partículas não são substituídas por outras com melhores resultados.

3.2.1 Formulação do Algoritmo

Eberhart e Kennedy [Eberhart e Kennedy 95] propõem dois paradigmas para implementação do conceito do PSO: um orientado globalmente (Gbest) e outro orientado localmente (Lbest). Ambos os modelos serão descritos nas secções seguintes.

Modelo Gbest

O modelo Gbest, correspondente à formulação original do algoritmo PSO, possui uma velocidade de convergência superior mas uma robustez inferior ao Lbest. Considerando um bando de NP partículas num espaço D-dimensional, a posição de cada partícula i numa determinada iteração $k+1$ é dada por

$$x_{k+1}^i = x_k^i + v_{k+1}^i, \tag{3.6}$$

[3] do inglês *Particle Swarm Optimization*.

em que v_{k+1}^i é a velocidade da partícula. Esta velocidade é actualizada de acordo com a informação partilhada pelo bando através da equação

$$v_{k+1}^i = v_k^i + c_1 \text{rand1}_i(0,1)(pbest_i - x_k^i) + c_2 \text{rand2}_i(0,1)(gbest - x_k^i), \tag{3.7}$$

onde $\text{rand1}_i(0,1)$ e $\text{rand2}_i(0,1)$ são números gerados aleatoriamente no intervalo $[0,1]$ e c_1 e c_2 são coeficientes de aceleração. $pbest_i$ representa o melhor valor obtido pela partícula i durante o processo de optimização. Por essa razão o termo $c_1 \text{rand1}_i(0,1)(pbest_i - x_k^i)$ expressa a influência da experiência da própria partícula no seu movimento, sendo, por esse facto, designado como factor cognitivo. De forma análoga, $gbest$ representa o melhor valor obtido em toda a população e o termo $c_2 \text{rand2}_i(0,1)(gbest - x_k^i)$ resulta num desvio do movimento da partícula i na direcção do melhor resultado obtido até ao momento. Por esta razão, é denominado de factor social.

A cada iteração a velocidade e posição de cada partícula são actualizadas, sendo esta, posteriormente, avaliada. Este ciclo repete-se até que um determinado critério de paragem seja alcançado.

Na figura 3.3 encontra-se representado o esquema do algoritmo básico de Optimização por Bandos de Partículas. O pseudo-código para o modelo $g$best algoritmo PSO é apresentado na tabela 3.2.

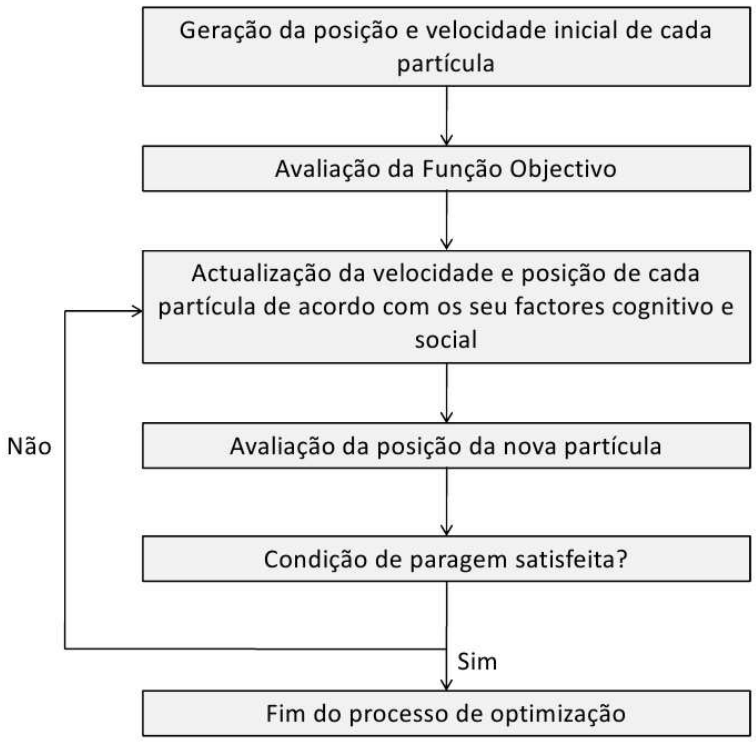


Figura 3.3: Esquema do algoritmo básico de Optimização por Bandos de Partículas

Modelo *L*best

O modelo *L*best tenta prevenir a convergência prematura mantendo múltiplas partículas atractivas, isto é, fazendo com que as partículas não sejam apenas atraídas pelo melhor membro da população e pelos seus melhores valores pessoais. Neste modelo as partículas possuem informação acerca do seu melhor valor e informação acerca do melhor valor obtido dentro de um subconjunto (vizinhança),

Tabela 3.2: Pseudo-código do modelo *gbest* do algoritmo PSO

```
!Criar a população aleatoriamente dentro do espaço de procura--------
initialize(x)
!Determinar a velocidade inicial-------------------------------------
initialize(v)
!Avaliar o melhor valor do bando-------------------------------------
initialize(gbest)
Fgbest=evaluate(gbest)
!Atribuir o melhor valor pessoal a população corrente----------------
pbest=x
!Melhor valor pessoal da função objectivo----------------------------
Fpbest=evaluate(pbest)

do k=1,itermax
   do i=1,NP
      do j=1,D
         r1=rand(0,1);r2=rand(0,1)
         v(i,j)=v(i,j)+r1*c1*(pbest(i,j)-x(i,j))+r2*c2*(gbest-x(i,j))
         x(i,j)=x(i,j)+v(i,j)
      end do
   end do

   !Avaliar a nova posição-----------------------------------------
   !do i=1,NP
      value(i)=evaluate(x(i,:))
      !Actualização do pbest e gbest
      if(value(i) < Fpbest) then
         pbest=x(i,:)
         Fpbest=value(i)
         if (value(i) < Fgbest) then
            gbest=x(i,:)
            Fgbest=value(i)
         end if
      end if
    end do
end do
```

mas não do conjunto global. Deste modo, a partícula move-se de acordo com o seu *p*best e o melhor valor da vizinhança lbest

A diferença entre este modelo e o descrito anteriormente centra-se na equação 3.7 que passa a ser escrita da seguinte forma

$$v_{k+1}^{i} = v_k^i + c_1 \text{rand1}_i(0,1)(p\text{best}_i - x_k^i) + c_2 \text{rand2}_i(0,1)(l\text{best}_i - x_k^i), \tag{3.8}$$

em que $lbest_i$ representa o melhor valor obtido na vizinhança da partícula i.

O modelo *Lbest* é uma aproximação mais flexível em termos de processamento de informação que o método *G*best. O modelo permite que grupos de partículas se separem espontâneamente e explorem regiões diferentes, fazendo com que o algoritmo fique preso em mínimos locais com menos frequência. No entanto, requer mais iterações para alcançar os objectivos pretendidos [Eberhart e Kennedy 95].

Existem diferentes tipologias de vizinhanças e estas afectam de forma significativa o desempenho do algoritmo, no entanto, o tipo de vizinhança ideal depende do problema em estudo [van der Berg 01]. As tipologias denominam-se pela definição dos grupos de vizinhanças. Como exemplo, destacam-se as tipologias de:

- **Anel** - neste tipo de vizinhança cada partícula considera que as suas duas partículas vizinhas imediatas constituem toda a sua vizinhança, isto é, uma dada partícula i apenas mantém contacto com as partículas $i-1$ e $i+1$. Esta vizinhança permite que as partículas explorem diferentes regiões do espaço de procura mantendo a partilha de informação [van der Berg 01].

- **Roda** - neste tipo de vizinhança considera-se que todas as partículas se encontram ligadas a uma única partícula central mas não directamente ligadas entre elas. Este modelo apresenta bons resultados quando aplicado a funções com vários mínimos locais.

- **Estrela** - neste tipo de vizinhança todas as partículas encontram-se ligadas entre si, pelo que corresponde ao modelo *G*best.

É ainda importante referir que as tipologias acima mencionadas são estruturadas de acordo com o índice das partículas e não de acordo com as suas posições no espaço de procura.

3.2.2 Parâmetros Operacionais

População e Velocidade Iniciais

A população de partículas no PSO é criada aleatoriamente dentro do espaço de procura cujos limites mínimos e máximos são dados, respectivamente, por $x_{j,min}$ e $x_{j,\max}$, com $j = 1, 2, ..., D$, através da equação

$$x_{ij}^0 = x_{j,\min} + \text{rand1}_j(0,1)(x_{j,\max} - x_{j,\min}), \tag{3.9}$$

em quem $\text{rand1}_j(0,1)$ é um número gerado aleatoriamente no intervalo $[0,1]$. De forma semelhante, a velocidade inicial de cada partícula é dada por

$$v_{ij}^0 = v_{j,\min} + \text{rand2}_j(0,1)(v_{j,\max} - v_{j,\min}), \tag{3.10}$$

onde $v_{j,\max} = (x_{j,\max} - x_{j,\min})/2$, $v_{j,\min} = -v_{j,\max}$ e $\text{rand2}_j(0,1)$ é um número gerado aleatoriamente no intervalo $[0,1]$.

Trelea [Trelea 03] concluiu que, na maioria dos casos, aumentando o número de partículas o número de iterações necessárias diminui e a taxa de sucesso do algoritmo (fracção do número de vezes que este alcançou o objectivo) aumenta significativamente. Esta melhoria deve-se ao facto de, por existirem mais partículas no espaço de procura, este é percorrido mais minuciosamente. No entanto, este aumento da população conduz a um aumento do número de avaliações da função que, por sua vez, leva a um aumento do tempo de computação. Como em aplicações reais o custo de optimização é geralmente dominado pelo número de avaliações da função objectivo, este parâmetro foi dado como sendo o principal critério de avaliação do desempenho do algoritmo.

O uso de uma população pequena pode conduzir a baixas taxas de sucesso e ao aumento do número de iterações até à convergência, mas populações grandes necessitam de um número elevado de avaliações da função a cada iteração.

Peso Inercial

Introduzido pela primeira vez por Shi e Eberhart [Shi e Eberhart 98], o peso inercial w é um parâmetro cujo objectivo é controlar o impacto da velocidade da iteração anterior na velocidade actual. Deste modo, e para o caso do modelo *G*best, a equação 3.7 passa a ser escrita da seguinte forma

$$v_{k+1}^i = wv_k^i + c_1\text{rand1}_i(0,1)(pbest_i - x_k^i) + c_2\text{rand2}_i(0,1)(gbest - x_k^i). \tag{3.11}$$

Shi e Eberhart [Shi e Eberhart 98] estudaram os efeitos de valores de w no intervalo $[0, 1.4]$. Os resultados obtidos indicam que um valor de w no intervalo $[0.9, 1.2]$ resulta numa convergência mais rápida, mas valores elevados ($w > 1.2$) resulta em mais falhas de convergência.

Os mesmos autores realizaram novos estudos considerando w como um factor que decrescia linearmente de 0.9 a 0.4 durante o processo de optimização [Shi e Eberhart 98-2]. Deste modo era permitido ao algoritmo realizar uma exploração global no início da simulação e refinar a solução numa fase mais tardia. Os resultados obtidos demonstram que o PSO possui a habilidade de convergir rapidamente, mas poderá faltar a capacidade para realizar a exploração global numa fase mais avançada do problema, o que poderá conduzir a dificuldades em encontrar o mínimo desejado.

Zheng *et. al.* [Zheng *et al.* 03] estudaram a influência de w quando este crescia linearmente no intervalo $[0.4, 0.9]$. Os resultados experimentais mostram que a exactidão e velocidade de convergência da procura global são melhoradas, sem aumentar a carga computacional.

Coeficientes de Aceleração

Os coeficientes de aceleração $c_1, c_2 \epsilon [0, 2]$ afectam o avanço que cada partícula pode perfazer numa determinada iteração. Um valor recomendado para estes coeficientes é $c_1 = c_2 = 2$, pois, deste modo, faz com que o peso do factores cognitivo e social sejam, em média, 1 [van der Berg 01].

Suganthan [Sunganthan 99] avaliou a hipótese de estes coeficientes serem linearmente decrescentes, concluindo que, de modo a obter bons resultados, os valores de c_1 e c_2 necessitam de ser constantes, mas não necessariamente iguais a 2.

3.2.3 Modificações e Aplicações do Algoritmo Básico

O algoritmo PSO foi, com o passar do tempo, alvo de inúmeros estudos que visam melhorar vários dos seus aspectos, tais como a velocidade de convergência e capacidade de equilibrar uma exploração global no início de processo e uma procura local durante uma fase mais avançada.

Vcerumachancni *et al.* [Vcerumachancni *et al.* 03] desenvolveram uma variante do PSO cujo objectivo era combater o problema de convergência prematura do algoritmo. Nesta variante, cada partícula é atraída em direcção às melhores posições visitadas anteriormente pelas partículas vizinhas. Isto é conseguido recorrendo a uma razão entre o valor relativo da função objectivo e a distância de outras partículas, determinando, deste modo, a direcção na qual cada componente da posição da partícula necessita de ser alterada.

Stacey *et al.* [Stacey *et al.* 03] introduziram um operador de mutação no algoritmo PSO de modo a acelerar a convergência e escapar de mínimos locais. Como o melhor indivíduo da população atrai todos os outros membros, é possível conduzir o bando noutra direcção caso a partícula mutada seja colocada numa nova posição melhor que aquela até ao momento descoberta.

Liu e Abraham [Liu e Abraham 05] propõem um algoritmo que recorre a um limiar mínimo de velocidade para controlar a velocidade das partículas. Esta velocidade mínima é ajustada dinamicamente através de um controlador lógico difuso. A ideia base do algoritmo consiste em controlar a velocidade para permitir que as partículas escapem de mínimos locais e continuem a exploração do espaço de procura. O limiar de velocidade mantém as partículas em movimento até que o algoritmo entre em convergência.

Foi proposto por Yang *et al.* [Yang *et al.* 07] um algoritmo de optimização por bando de partículas com adaptação dinâmica. No algoritmo desenvolvido a velocidade de cada partícula é actualizada através de uma equação na qual a aleatoriedade vai diminuindo durante o processo de optimização. Neste estudo consideram-se ainda pesos inerciais diferentes para cada partícula. De forma a descrever de uma forma mais eficaz a evolução do algoritmo são introduzidos os parâmetros factor de agregação e factor de velocidade de evolução.

Xi *et al.* [Xi *et al.* 08] propõem um algoritmo de optimização por bando de partículas com comportamento fraccionário (QPSO[4]) com uma capacidade de exploração melhorada e menos parâmetros de controlo. É ainda introduzida uma média ponderada da melhor posição da população que permite guiar a procura dos bando de partículas.

Jie *et al.* [Jie *et al.* 08] desenvolveram uma variante do algoritmo PSO que simula os processos de auto-aprendizagem dos agentes evolucionários em ambientes especiais e que memoriza a informação da procura. Através da partilha de informação, o algoritmo manipula vários sub-bandos, nos quais cada partícula segue um comportamento de aprendizagem social. Cada um destes sub-bandos realiza uma procura local em diferentes posições do espaço de procura. Simultaneamente, uma procura global é realizada através da troca de informação do comportamento de partículas de diferentes sub-bandos.

García-Villoria e Pastor [García-Villoria e Pastor 09] introduziram uma velocidade aleatória de forma a aumentar a diversidade e, desta forma, diminuir o risco de convergência prematura. O grau de diversidade é alterado dinamicamente de acordo com a heterogeneidade da população.

Todas estas modificações conduzem a uma melhoria do desempenho do algoritmo, tornando possível a sua aplicação em problemas de optimização nas mais diversas áreas. Actualmente, o algoritmo de Optimização por Bandos de Partículas pode ser aplicado, por exemplo, em problemas de engenharia e gestão. O algoritmo PSO é usado para melhorar o desempenho de máquinas eléctricas através do controlo do fluxo do rotor, no processo de aprendizagem de redes neuronais, organização de *layouts* fabris e calendarização de tarefas, entre outras. O PSO foi ainda aplicado com sucesso a problemas de optimização de projecto estrutural [Perez e Behdinan 07] e identificação de parâmetros de modelos constitutivos [Zheng *et al.* 08].

3.3 Algoritmo de Recozimento Simulado

O algoritmo de Recozimento Simulado (SA[5]) é um método de procura aleatória que simula o modo como um metal arrefece e solidifica na direcção de um estado de energia mínimo.

Recozimento é um tratamento térmico de metais ou ligas que consiste em aquecer o material até uma temperatura pré-determinada e, posteriormente, arrefece-lo lentamente até à temperatura ambiente, de modo a melhorar a ductilidade e reduzir a fragilidade. Durante o arrefecimento, o material atinge um estado de baixa energia e flutuações aleatórias inerentes desta permitem que o sistema escape de mínimos de energia locais de modo a atingir o mínimo global [Goffe *et al.* 94].

3.3.1 O Método

A procura do mínimo é iniciada, para um dado estado de temperatura T, num ponto de partida $\mathbf{x}$ dentro dos limites mínimo ($\mathbf{x}_{\min}$) e máximo ($\mathbf{x}_{\max}$) do problema com D dimensões. Um vector $\mathbf{s}$ com os tamanhos de passo s_i ao longo de cada direcção x_i, para $i = 1, 2, ...D$, é criado. Inicialmente, cada s_i poderá ter um tamanho s_T sendo depois reduzido usando o parâmetro r_s. São criados ainda vectores de aceitação α com D valores iguais a 1 e um factor de redução de temperatura r_T.

O ciclo principal do algoritmo é o ciclo de temperatura na qual esta é definida como $r_T T$. O passo dado por $r_s S_T$ é actualizado no final de cada incremento de temperatura. A cada temperatura, N_T iterações são realizadas, consistindo cada uma delas em N_C ciclos nos quais são dados passos aleatórios sucessivos em cada uma das D direcções de acordo com

$$\mathbf{x}_s = \mathbf{x} + \text{rand}(-1, 1) s_i x_i, \tag{3.12}$$

[4]do inglês *Quantum-behaved Particle Swarm Optimization*.
[5]do Inglês *Simulated Annealing*.

em que $\mathbf{x}_s$ é o novo ponto e rand$(-1, 1)$ é um número gerado aleatoriamente entre -1 e 1.

O novo valor da função objectivo f_s é então avaliado. Caso $f_s \leq f$, o valor de $\mathbf{x}_s$ é aceite e, adicionalmente, se $f_s \leq f_{\text{best}}$ então $\mathbf{x}_{\text{best}} = \mathbf{x}_s$, em que $\mathbf{x}_{\text{best}}$ e f_{best} correspondem, respectivamente, à melhor posição e melhor valor de função objectivo encontrado até ao momento. Se $f_s \geq f$, então o novo valor é aceite com uma probabilidade dada pela equação

$$p = exp\left(-\frac{\Delta f}{T}\right) \geq r, \tag{3.13}$$

em que r é um número gerado aleatoriamente entre 0 e 1.

Sempre que um valor é rejeitado, a razão de aceitação α_i, correspondente à razão entre o número total de aceitações e o número total de tentativas para a direcção i, é actualizado fazendo

$$\alpha_i = \alpha_i - \frac{1}{N_C}. \tag{3.14}$$

No final de N_C ciclos, o valor de α_i é usado para actualizar o tamanho e a direcção do passo. Um valor baixo implica um grande número de rejeições, razão esta que sugere que o tamanho de passo s_i deverá ser reduzido. Um valor elevado pressupõe um maior número de aceitações, pelo que s_i poderá ser demasiado pequeno sendo, então, necessário aumentá-lo. Se $\alpha_i = 0.5$, o número de aceitações e rejeições são iguais, do qual se conclui que o valor actual de s_i é adequado [Belegundu e Chandrupatla 99].

De modo a realizar as actualizações do tamanho do passo, um factor de multiplicação g é introduzido de modo que

$$s_i^{\text{new}} = g s_i^{\text{old}}, \tag{3.15}$$

em que g é dado por

$$g = \begin{cases} 1 + 2\frac{\alpha_i - 0.6}{0.4}, & \text{se } \alpha_i > 0.6 \\ \frac{1}{1+2\frac{0.4-\alpha_i}{0.4}}, & \text{se } \alpha_i < 0.4. \\ 1, & \text{se } \alpha_i = 0.5 \end{cases} \tag{3.16}$$

O esquema para o algoritmo básico de Recozimento Simulado é apresentado na figura 3.4.

3.4 Algoritmos Genéticos e Evolucionários

O Algoritmo Genético (GA[6]), inicialmente adaptado para processos computacionais por Holland [Holland 09], é um método de procura para obtenção de soluções aproximadas de problemas que utiliza a genética como base.

Na Natureza, a maioria dos organismos evolui através de dois processos: selecção natural e reprodução. O primeiro determina quais os membros mais aptos da população que sobrevivem e se reproduzem e o segundo garante uma mistura e recombinação dos genes da sua descendência.

Seguindo este principio, o GA comanda uma população de possíveis soluções, sendo cada uma destas codificada num número binário. É realizada uma selecção dos melhores membros e nestes são aplicados operadores de reprodução que são usados para criar mutações e recombinações de modo a explorar novas soluções do problema. Posteriormente, os novos membros são avaliados e determinam-se quais os que farão parte da nova geração.

O Algoritmo Evolucionário (EA[7]) é um método de optimização similar ao GA. A principal diferença reside na codificação da população. No GA, a população é, geralmente, codificada em numeração binária, pelo que o problema passa a ser tratado como sendo discreto. Esta codificação

[6] do Inglês *Genetic Algorithm*.
[7] do inglês *Evolutionary Algorithm*.

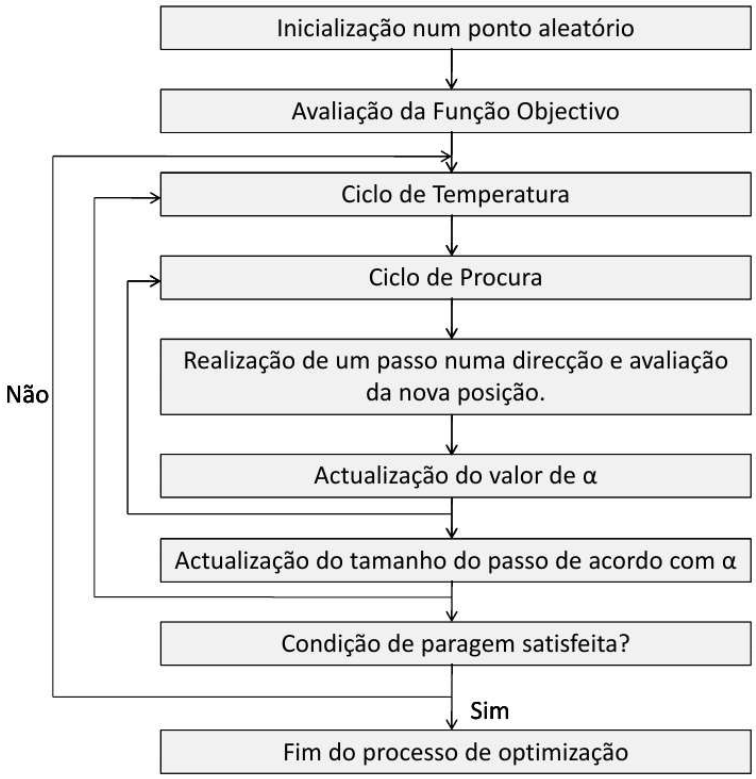


Figura 3.4: Esquema do algoritmo básico de Recozimento Simulado

acarreta uma perda de precisão do processo, uma vez que a codificação limita a qualidade das soluções. No Algoritmo Evolucionário não existe codificação, facto que o torna mais apto para resolução de problemas reais.

3.4.1 O Método

No Algoritmo Genético, considerando uma população **x** de tamanho NP e com dimensões D em que os limites mínimos e máximos de cada dimensão são dados por

$$x_{i,\min} \leq x_i \leq x_{i,\max}, \tag{3.17}$$

o GA necessita de codificar cada variável num número binário com b bits. Por essa razão, o espaço de procura terá de ser dividido em $2^b - 1$ intervalos e cada variável é representada de forma discreta [Belegundu e Chandrupatla 99].

Inicialmente, uma população inicial com NP membros é gerada aleatoriamente no espaço de procura. Para cada dimensão de cada um dos membros é gerado um número r aleatório entre 0 e 1. Se $r \leq 0.5$ a posição corresponde a um bit de valor 0, caso contrario, o bit será 1. Seguidamente, realiza-se a avaliação de cada membro de acordo com o seu valor da função objectivo e, posteriormente, é efectuada uma selecção. Esta corresponde à selecção dos indivíduos para reprodução e é executada de forma aleatória com uma probabilidade dependente do valor relativo da função objectivo dos membros. Isto leva a que os melhores indivíduos sejam escolhidos com maior frequência. Os indivíduos seleccionados formam o local de emparelhamento.

No processo de reprodução, a descendência é gerada a partir dos indivíduos seleccionados. Podem ser realizadas operações de cruzamento e mutação. No cruzamento, são escolhidos dois membros do local de emparelhamento e é gerado um número inteiro aleatório k no intervalo

$[1, b \times D]$. Os primeiros k bits de cada progenitor são permutados para gerar dois descendentes. Na operação de mutação todos os bits de cada membro são percorridos. É escolhida uma probabilidade de permutação de bits b_p, normalmente compreendida entre 0.005 e 0.1, e esta é comparada com um número aleatório r_m pertencente ao intervalo $[0, 1]$. Se $r_m \leq b_p$ então o valor do bit é trocado.

Finalmente, a população é avaliada e são guardados os parâmetros do melhor membro. Se algum dos critérios de paragem tiver sido alcançado, o processo pára. Caso contrário, a população anterior é eliminada e o algoritmo volta a repetir o processo. O esquema do algoritmo básico do GA encontra-se exposto na figura 3.5.

Para mais detalhes sobre estes tipos de métodos ver [Sivanandam e Deepa 08] e [Ashlock 05].

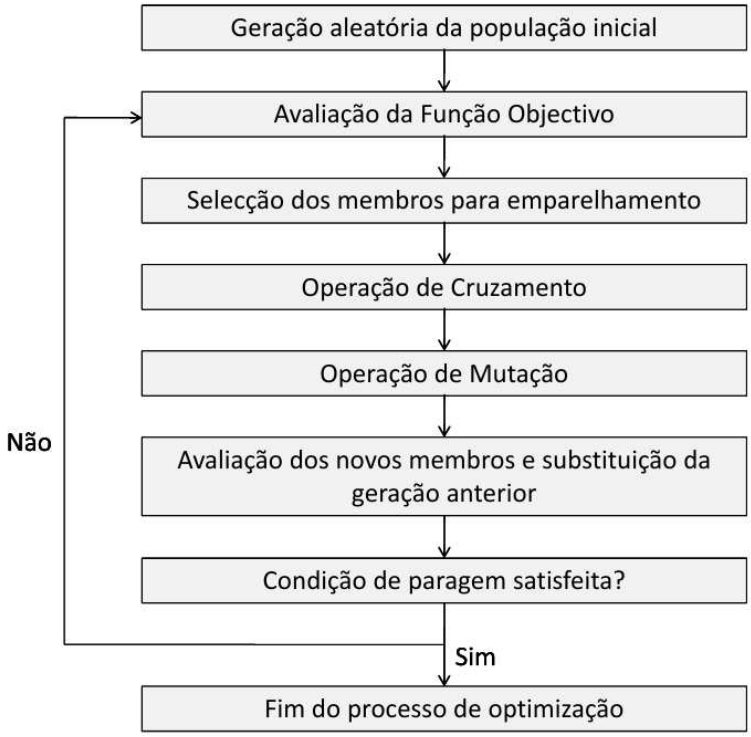


Figura 3.5: Esquema do algoritmo básico do Algoritmo Genético

3.5 Algoritmo de Levenberg-Marquardt

O algoritmo de Levenberg-Marquardt é um método de optimização que combina as vantagens dos métodos clássicos de gradiente (que realizam a minimização ao longo das direcções do gradiente) e do método de Newton (que utiliza um modelo quadrático para acelerar a descoberta de um mínimo de uma função). Esta combinação confere ao algoritmo uma estabilidade operacional e boa velocidade de convergência.

A base do algoritmo de Levenberg-Marquardt consiste numa aproximação linear de f através de uma vizinhança de $\mathbf{x}$. A cada iteração pode escrever-se

$$f(\mathbf{x} + \delta\mathbf{x}) = f(\mathbf{x}) + \mathbf{J}\delta\mathbf{x}, \tag{3.18}$$

em que $\mathbf{J}$ é a matriz Jacobiana $\frac{\delta f(\mathbf{x})}{\delta \mathbf{x}}$. O processo de optimização é iniciado numa posição inicial x_0 e o algoritmo gera uma série de vectores $\mathbf{x}_1$, $\mathbf{x}_2$, ... que convergem na direcção de um mínimo

local $\mathbf{x}^+$ de f. Desta forma, a cada iteração, é necessário encontrar o valor $\delta\mathbf{x}$ que minimiza a quantidade

$$||\mathbf{x} - f(\mathbf{x} + \delta\mathbf{x})|| \approx ||\mathbf{x} - f(\mathbf{x}) - \mathbf{J}\delta\mathbf{x}|| = ||\epsilon - \mathbf{J}\delta\mathbf{x}||. \tag{3.19}$$

Para uma qualquer iteração, o passo $\delta\mathbf{x}$ do algoritmo de Levenberg-Marquardt é dado através da resolução do seguinte sistema

$$(\mathbf{J}^T\mathbf{J} + \mu\mathbf{I})\delta\mathbf{x} = -\mathbf{J}^T\epsilon, \tag{3.20}$$

em que ϵ é o vector de resíduos e $\mathbf{I}$ a matriz identidade. O parâmetro de amortecimento μ promove diferentes comportamentos do método, isto é, μ influência a direcção e tamanho de passo da seguinte forma [Lourakis 05]:

- Para $\mu > 0$, a matriz de coeficientes $(\mathbf{J}^T\mathbf{J} + \mu\mathbf{I})$ é positiva definida, o que implica que $\delta\mathbf{x}$ aponta numa direcção de descida;
- Para valores de μ elevados o algoritmo realiza movimentos pequenos na direcção máxima de descida, pelo que o tamanho de passo $\delta\mathbf{x}$ é reduzido;
- Para valores de μ muito pequenos, o algoritmo consegue uma convergência quadrática.

Se o passo $\delta\mathbf{x}$ conduzir a uma redução do resíduo ϵ, então o passo é aceite e o processo é repetido para um factor de amortecimento inferior. Caso contrário, o valor de μ é aumentado e as equações são resolvidas novamente. O método realiza iterações até encontrar um valor de $\delta\mathbf{x}$ que reduza ϵ, ou atingir um determinado critério de paragem. Pode, por esta razão, afirmar-se que este é um método adaptativo pois permite alternar entre uma descida lenta quando afastado de um mínimo e uma convergência veloz quando na vizinhança de um mínimo. O esquema do algoritmo básico de Levenberg-Marquardt é apresentado na figura 3.6. Para mais detalhes sobre este método ver [Chong e Zak 01].

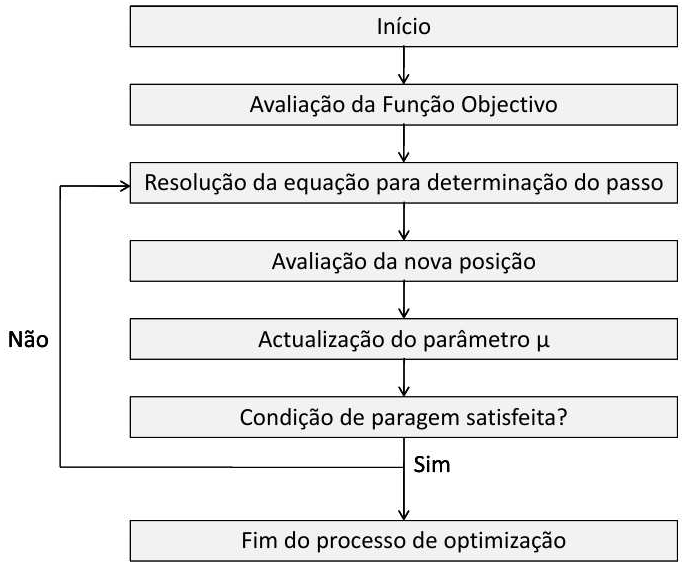


Figura 3.6: Esquema do algoritmo básico de Levenberg-Marquardt

3.6 Programas de Optimização

Existem actualmente diversas implementações de metodologias de optimização de variados fabricantes de computadores e companhias de software. Estes pacotes variam de acordo com a sua complexidade, facilidade de utilização e preço. O Microsoft Excel possui algumas características de optimização de âmbito geral que lhe permitem resolver pequenos problemas. O MATLAB é um programa que oferece uma variedade de métodos nas suas Caixas de Ferramentas de Optimização e um ambiente interactivo para definir modelos, reunir dados e exibir resultados [Ravindran *et al.* 06].

3.6.1 O Programa modeFRONTIER

O programa modeFRONTIER, desenvolvido pela ESTECO srl, é um ambiente de optimização e projecto que permite uma fácil integração com uma grande variedade de programas de engenharia auxiliada por computador (CAE[8]). O modeFRONTIER possui ferramentas de optimização multi-objectivo que incluem várias rotinas de optimização.

3.6.2 O Programa SiDoLo

O programa SiDoLo[9], inicialmente desenvolvido por Philippe Pilvin, é um programa de optimização de âmbito genérico que pode ser utilizado em diversos campos científicos e de engenharia. Embora o corpo do programa seja dedicado à optimização, ele contém sub-rotinas que permitem, em certo tipo de sistemas físicos, programar as equações de um sistema e obter as respostas deste para diferentes solicitações.

[8] do inglês *Computer Aided Engineering*.
[9] do francês Simulation et iDentification automatique de Lois de comportement.

Capítulo 4

Algoritmo Desenvolvido

Descreve-se de forma pormenorizada o algoritmo desenvolvido e quais as diferentes metodologias implementadas.

4.1 Introdução

Actualmente não existe um algoritmo de optimização que seja suficientemente robusto para resolver todo o tipo de problemas. Em problemas complexos e com um elevado número de mínimos locais é importante que o algoritmo possua, inicialmente, uma boa capacidade de exploração global para deste modo conseguir percorrer com algum pormenor todo o domínio do problema. Numa fase mais avançada do problema, o algoritmo deve possuir a capacidade de realizar uma procura local e refinar a solução. Para além disto, é ainda de extrema importância que o algoritmo possua uma boa velocidade de convergência de forma a encontrar a solução de forma rápida e, deste modo, reduzir o tempo de computação necessário.

De forma a tentar satisfazer todos estes critérios, foi desenvolvido um algoritmo híbrido baseado nos algoritmos de Evolução Diferencial e de Optimização por Bandos de Partículas. Pretende-se com isto incorporar num só algoritmo as vantagens de cada um dos algoritmos base. Primeiramente, cada um dos algoritmos foi estudado e desenvolvido separadamente para, numa segunda fase, serem implementados em série.

4.2 Descrição Geral do Processo

Inicialmente é gerada uma população aleatória e são avaliados todos os membros dessa mesma população. Avalia-se, então, a condição que irá determinar qual o algoritmo base a ser usado na iteração seguinte. Esta condição consiste na contagem de iterações subsequentes realizadas pelo algoritmo base da geração anterior. De uma forma simples, são realizadas I iterações recorrendo ao DE, seguido pelo mesmo número de iterações com o PSO, sendo este processo repetido até ser atingido o critério de paragem. A sequência de trabalho do algoritmo desenvolvido encontra-se representada na figura 4.1.

Tal como foi supramencionado, cada um dos algoritmos base foi estudado individualmente. No algoritmo de Evolução Diferencial considerou-se o factor de escala da solução F e a probabilidade de cruzamento CR como sendo parâmetros auto-adaptativos e introduziu-se um factor de mutação caótico. Por sua vez, no algoritmo de Optimização por Bandos de Partículas, considerou-se o modelo *Lbest* com vizinhanças em anel e introduziu-se uma factor de turbulência e um peso

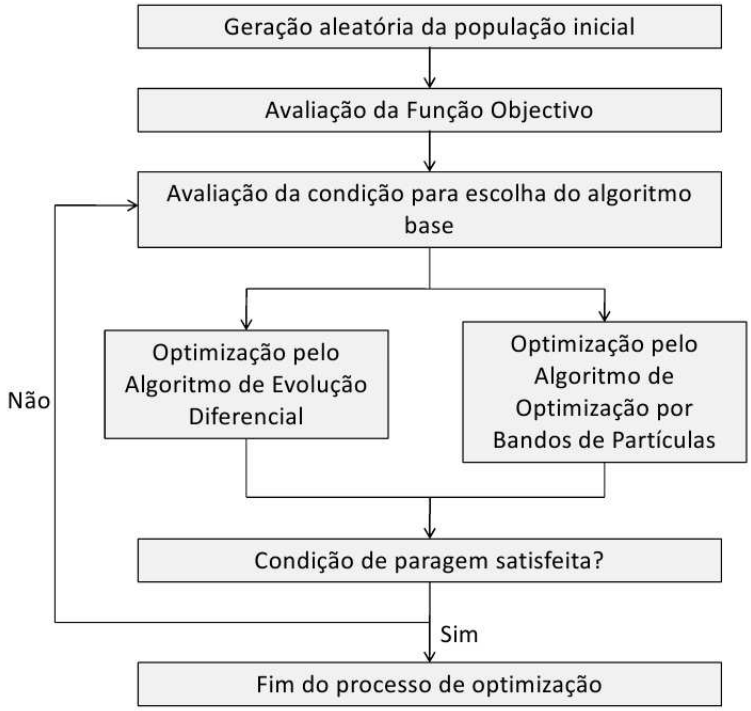


Figura 4.1: Esquema do algoritmo desenvolvido

inercial crescente. Finalmente, implementou-se um critério de estagnação de forma a permitir que o algoritmo consiga explorar novas soluções após ter estagnado. Cada uma destas metodologias é descrita em pormenor nas secções seguinte.

4.3 Evolução Diferencial com F e CR Auto-Adaptativos

Introduzida por Zhenyu *et al.* [Zhenyu *et al.* 06], o algoritmo de Evolução Diferencial Auto-Adaptativo recorre a um factor de agregação e velocidade que permitem, de forma dinâmica, adaptar o factor de escala e a probabilidade de cruzamento.

A razão entre o valor da função objectivo do melhor indivíduo e da média da população, a cada geração, é usado para representar o nível de agregação da população. Deste modo, o factor de agregação é dado por

$$\beta = \frac{\min(F_{\text{best},G}, \overline{F_G})}{\max(F_{\text{best},G}, \overline{F_G})}, \tag{4.1}$$

em que $F_{\text{best},G}$ é o valor da função objectivo do melhor membro e $\overline{F_G}$ é a média da população para uma dada geração G. Por sua vez, o factor de escala passa a ser escrito como

$$F = \beta F_s, \tag{4.2}$$

em que F_s é uma constante. Desta forma, o factor de escala é alterado de forma dinâmica de acordo com o factor de agregação. Um valor de β baixo implica uma grande diferença entre o valor da função objectivo do melhor membro da população e da média de toda a população. Desta forma, pode dizer-se que existe uma grande diversidade, pelo que o factor de escala da solução deverá ser diminuído de modo a reduzi-la e induzir a convergência do algoritmo.

Valores de CR elevados permitem uma convergência mais rápida, mas o algoritmo pode estagnar em mínimos locais. Por outro lado, valores baixos de CR são mais robustos, mas possuem

uma velocidade de convergência baixa. De modo a poder controlar este parâmetro de forma dinâmica, considera-se que a probabilidade de cruzamento passa a ser escrita por

$$CR = \alpha CR_s, \tag{4.3}$$

em que CR_s é um valor constante e α corresponde ao factor de evolução da velocidade. Este é dado pela razão entre o melhor valor da função objectivo obtido na geração actual pelo melhor valor da função objectivo da geração anterior e pode ser expresso através de

$$\alpha = \frac{\min(F_{\text{best},G}, F_{\text{best},G-1})}{\max(F_{\text{best},G}, F_{\text{best},G-1})}, \tag{4.4}$$

em que $F_{\text{best},G-1}$ é o valor da função objectivo do melhor membro da geração anterior. Um valor de α baixo implica que a velocidade de convergência do algoritmo poderá ser demasiado elevada, levando a que o algoritmo fique estagnado num mínimo local com maior facilidade. Por esta razão, é vantajoso reduzir a velocidade de convergência através da diminuição da probabilidade de cruzamento.

4.4 Evolução Diferencial com Factor de Mutação Caótico

Algoritmos de optimização baseados na teoria do caos são metodologias de procura estocásticas que diferem dos algoritmos evolucionários existentes. Este tipo de metodologias baseiam-se em ergodicidade[1], propriedades estocásticas e irregularidades. Estas aproximações permitem que os algoritmos escapem mais facilmente de mínimos locais através da aceitação de soluções piores de acordo com uma dada probabilidade.

Um dos sistemas dinâmicos, estudados por Coelho e Mariani [Coelho e Mariani 06] e Zhenyu *et al.* [Zhenyu *et al.* 06], que evidenciam comportamento caótico é chamado de mapa logístico, cuja equação para geração do indivíduo é dada por

$$x_{i,g} = 4x_{\text{worst},G-1}(1 - x_{\text{worst},G-1}), \tag{4.5}$$

em que $x_{\text{worst},G-1}$ corresponde ao pior membro da geração anterior que será substituído directamente pelo novo membro. Este processo ocorre após a operação de selecção.

4.5 Optimização por Bandos de Partículas com Turbulência

A cada iteração, as partículas são atraídas na direcção do melhor membro da vizinhança e do melhor valor obtido pela própria partícula. A partícula pode, eventualmente, perder a sua capacidade de exploração em futuras iterações. Este facto leva a uma convergência prematura do algoritmo.

Liu e Abraham [Liu e Abraham 05] propõem uma estratégia para evitar que as partículas percam a capacidade de explorar o espaço de procura. Se a velocidade da partícula diminuir para além de um limiar v_c, uma nova velocidade é atribuída de acordo com

$$\widehat{v} = \begin{cases} v_{j,i}, & \text{se } |v_{j,i}| \geq v_c \\ \text{rand}(-1,1)\frac{v\max}{\rho}, & \text{se } |v_{j,i}| < v_c \end{cases}, \tag{4.6}$$

em que $\text{rand}(-1,1)$ é um número aleatório no intervalo $[-1,1]$ e ρ é o factor de escala de controlo do domínio de oscilação da partícula de acordo com v_{max}. Desta forma, a equação de actualização da velocidade passa a ser escrita como

$$v^i_{k+1} = w\widehat{v}^i_k + c_1\text{rand}1_i(0,1)(pbest_i - x^i_k) + c_2\text{rand}2_i(0,1)(gbest - x^i_k). \tag{4.7}$$

[1] atributo de sistemas estocásticos; geralmente, um sistema que tende para um estado limite que é independente das condições iniciais.

O desempenho do algoritmo encontra-se directamente relacionado com os parâmetros v_c e ρ. Um valor de v_c elevado diminui o período de oscilação, o que permite, com uma maior probabilidade, que a partícula *salte* por cima de um mínimo local recorrendo ao mesmo número de iterações. No entanto, um elevado v_c pode impedir que a procura seja refinada. Por esta razão, este parâmetro deverá ser controlado de forma dinâmica, começando em valores elevados que favoreçam uma exploração global e terminando com um valor inferior de modo a facilitar a procura local. O parâmetro ρ modifica directamente o domínio de oscilação da partícula. É possível que as partículas não consigam *saltar* sobre um mínimo local se este for muito largo. No entanto, a trajectória da partícula estaria mais propensa a oscilações com um valor de ρ baixo. Desta forma, um valor crescente de ρ seria uma forma eficaz de controlar o compromisso exploração global/procura local.

4.6 Optimização por Bandos de Partículas com Peso Inercial Crescente

De acordo com os estudo de Zheng *et al.* [Zheng *et al.* 03], um peso inercial w elevado conduz a uma maior probabilidade de convergência. No entanto, um valor de w baixo nem sempre é prejudicial, pois possibilita que o algoritmo saia de um mínimo local e explore novas áreas. Atendendo a estes factos, os referidos autores propõem um novo modelo descrito por

$$v_{k+1}^i = wv_k^i + c_1\phi_1(pbest_i - x_k^i) + c_2\phi_2(gbest - x_k^i), \tag{4.8}$$

em que

$$\phi_i = b_i \text{rand}_i(0,1) + d_i, \text{ com i=1,2.} \tag{4.9}$$

As constantes b_1 e b_2 tomam o valor de 1.5 e d_1 e d_2 assumem-se com sendo iguais a 0.5 de maneira a cooperarem com b_1 e b_2 com o objectivo de manter ϕ_1 e ϕ_2 no intervalo $[0.5, 2]$.

Nos seus estudos, Zheng *et al.* [Zheng *et al.* 03] utilizaram um peso inercial linearmente crescente de 0.4 a 0.9. Os resultados obtidos demonstram que, quando comparado com um peso inercial decrescente, esta aproximação aumenta grandemente a exactidão e velocidade de convergência da procura global, mantendo a mesma carga computacional.

4.7 Critério de Estagnação

Quando um algoritmo entra em estagnação significa que não consegue encontrar soluções aperfeiçoadas. No entanto, a solução na qual o algoritmo ficou preso pode não ser a óptima global. De modo a poder explorar novas soluções no espaço de procura, foi implementado um critério de estagnação.

Se, após um determinado número de iterações, o algoritmo não conseguir encontrar uma solução melhor, considera-se que este convergiu e faz-se uma recolocação aleatória da população. No caso específico do PSO, reinicializam-se as velocidades das partículas. A probabilidade de cruzamento CR_s é diminuída e o factor de escala da solução F_s é aumentada de modo a evitar uma convergência prematura e aumentar a capacidade de exploração global do algoritmo.

Capítulo 5

Validação e Resultados

Realiza-se o estudo para determinar quais os valores ideais dos diferentes parâmetros operacionais utilizados pelo algoritmo desenvolvido. Compara-se o algoritmo desenvolvido com algoritmos existentes recorrendo a funções de teste matemáticas. Aplica-se o algoritmo desenvolvido a diversos problemas de optimização em Engenharia Mecânica.

5.1 Validação do Algoritmo Desenvolvido

Actualmente, muitos algoritmos de optimização tiram proveito do facto das propriedades das funções de teste, tais como mínimos globais na origem, mínimos globais sobre os eixos coordenados ou valores de parâmetros iguais nas diferentes dimensões do mínimo global, serem conhecidas. Para evitar estas situações e para avaliar de forma mais correcta os algoritmos de optimização, foi criada uma série de funções matemáticas complexas com base nos trabalhos de Liang *et al.*[Liang *et al.* 05].

Muitos problemas actuais podem ser expressos da seguinte forma

$$\text{minimizar } f(\mathbf{x}), \tag{5.1}$$

em que

$$\mathbf{x} = (x_1, x_2, ..., x_D), \text{ com } \mathbf{x}\epsilon[x_{\min}, x_{\max}]. \tag{5.2}$$

sendo que D representa a dimensão do problema em estudo. Tendo em conta este tipo de problemas foram criadas várias funções compostas com certas propriedades especificas de modo a evitar algumas das particularidades apresentados pelos algoritmos de optimização, sendo estes os seguintes:

1. **Óptimos globais com os mesmos valores em diferentes dimensões.** A maioria das funções mais populares têm, no mínimo global, os mesmos valores de parâmetros em cada uma das suas dimensões devido ao facto de serem simétricas. Por exemplo, a função de Rastrigin e de Griewank têm o seu mínimo global em $[0, 0, ..., 0]$. Nesta situação, se existir algum operador que copie o valor de uma dimensão para as outras dimensões, o mínimo global pode ser encontrado rapidamente. No entanto, este operador não é útil se o óptimo global não possuir dimensões com valores iguais.

2. **Óptimo global na origem.** Nesta situação o óptimo global encontra-se em $\mathbf{x} = [0, 0, ..., 0]$, pelo que também existem operadores específicos para este tipo de situações mas que são ineficientes quando o óptimo global não se encontra na origem.

3. **Óptimo global localizado no centro do espaço de procura.** Alguns algoritmos possuem a capacidade de convergir para o centro do espaço de procura.

4. **Óptimos globais nas fronteiras.** Se o óptimo global se encontrar nas fronteiras do espaço de procura, alguns algoritmos poderão encontra-lo com facilidade. No entanto, se existir algum óptimo local próximo das fronteiras, é fácil cair neste e ,consequentemente, falhar em encontrar o mínimo global.

5. **Óptimos locais sobre os eixos coordenados ou falta de ligação entre as dimensões.** A maioria das funções de teste, especialmente aquelas com grande número de dimensões, possuem estruturas simétricas e os seus mínimos locais encontram-se sempre sobre os eixos coordenados. Neste caso, a informação dos mínimos locais pode ser usada para localizar o mínimo global. Além disso, nalgumas funções os mínimos globais podem ser localizados fazendo D procuras uni-dimensionais num problema com D dimensões.

De modo a evitar estas particularidades nas funções de teste e testar um novo algoritmo de forma mais correcta, são propostos os seguintes métodos [Liang *et al.* 05]:

1. Mover o óptimo global para uma posição aleatória de modo a que os valores dos vários parâmetros sejam diferentes para cada dimensão. Isto pode ser conseguido considerando

$$F(\mathbf{x}) = f(\mathbf{x} - \mathbf{o}_{\text{new}} + \mathbf{o}_{\text{old}}), \tag{5.3}$$

   em que $F(\mathbf{x})$ é a nova função, $f(\mathbf{x})$ a função antiga, $\mathbf{o}_{\text{old}}$ o antigo mínimo global e $\mathbf{o}_{\text{new}}$ o novo mínimo global com valores diferentes para as várias dimensões e não colocado no centro do espaço de procura. Desta forma, ao atribuir valores aleatórios a $\mathbf{o}_{\text{new}}$, podem ser resolvidos os problemas apresentados de 1 a 3.

2. Para o problema apresentado em 4, considerando que existem problemas reais com óptimo global sobre as fronteiras, é um método aceitável para tratamento das fronteiras colocar a população próxima das fronteiras quando esta sai fora do espaço de procura. No entanto, sugere-se que sejam testadas funções com e sem mínimos globais nas fronteiras, bem como funções com mínimos locais nas fronteiras.

3. Para funções que sofram do problema 5, recorre-se à equação

$$F(\mathbf{x}) = f(\mathbf{R}\,\mathbf{x}), \tag{5.4}$$

   em que $\mathbf{R}$ é uma matriz ortogonal de rotação. Deste modo pode ser evitada a colocação de mínimos locais sobre os eixos coordenados e mantidas as propriedades da função de teste.

5.1.1 Funções Compostas

De acordo com o descrito na secção anterior, Liang *et al.*[Liang *et al.* 05], propõem uma estrutura para criar novas e mais exigentes funções compostas que possuam as propriedades desejadas. A ideia principal dessa estrutura consiste em compor as funções de teste padrão de forma a construir novas funções com o mínimo global localizado aleatoriamente no espaço de procura e com vários mínimos locais profundos, também eles distribuídos aleatoriamente.

As novas funções compostas $F(\mathbf{x})$ podem ser obtidas da seguinte forma

$$F(\mathbf{x}) = \sum_{i=1}^{n}(w_i * [f_i'((\mathbf{x} - \mathbf{o}_i + \mathbf{o}_{i\text{old}})/\lambda_i * \mathbf{M}_i) + \text{bias}_i]) + f\_\text{bias}, \tag{5.5}$$

em que n é o número de funções básicas, $f_i(\mathbf{x})$, com $i = 1, 2, .., n$ são as funções usadas na construção da função composta, $\mathbf{M}_i$ é a matriz de rotação $D \times D$ de cada função $f_i(\mathbf{x})$, $\mathbf{o}_i$ é a nova posição do óptimo global e $\mathbf{o}_{i\text{old}}$ a posição antiga do óptimo global.

O peso w_i de cada função $f_i(\mathbf{x})$ é calculado através de

$$w_i = \exp\left(-\frac{\sum_{k=1}^{D}(x_k - o_{ik} + o_{i\text{kold}})^2}{2D\sigma_1^2}\right). \tag{5.6}$$

Seguidamente determina-se o máximo valor de w_i máximo definindo-se

$$w_i = \begin{cases} w_i, & w_i \neq \max(w_i) \\ w_i(1 - \max(w_i)^{10}), & w_i = \max(w_i) \end{cases}. \tag{5.7}$$

Por fim este valor é normalizado da seguinte maneira

$$w_i = \frac{w_i}{\sum_{i=1}^{n} w_i}. \tag{5.8}$$

Na equação 5.6, σ_i é usado para controlar o alcance de cobertura de cada função $f_i(\mathbf{x})$, onde um valor de σ_i baixo reflecte-se num alcance pequeno. λ_i é usado para alongar ou comprimir a função, em que $\lambda_i < 1$ significa alongar e $\lambda_i < 1$ comprimir. bias_i define qual dos óptimos é o óptimo global, sendo este correspondente ao menor valor de bias_i. Recorrendo a o_i e bias_i o óptimo global pode ser colocado em qualquer local no espaço de procura.

Se $f_i(\mathbf{x})$ forem funções diferentes, então estas irão possuir diferentes propriedades e peso. Deste modo, e de forma a obter uma melhor combinação, estima-se qual o maior valor $f_i^{\max}$ entre as n funções usadas, sendo estas, posteriormente, normalizadas de acordo com

$$f_i'(\mathbf{x}) = C\frac{f_i(\mathbf{x})}{|f_i^{\max}|}, \tag{5.9}$$

em que C é uma constante pré-definida e

$$|f_i^{\max}| = f_i((z/\lambda_i)\mathbf{M}_i), \tag{5.10}$$

em que z corresponde ao valor do limite superior da função composta.

Para testar o algoritmo desenvolvido foi criado em linguagem de programação Fortran um programa que possui como entrada os valores de x_i e que fornece à saída o valor de $F(\mathbf{x})$ correspondente. O pseudo-código encontra-se disponível na tabela 5.1.

5.1.2 Construção das Funções Compostas

Tal como em [Liang *et al.* 05], foram definidas e criadas seis funções compostas para testar os algoritmos. Usaram-se os parâmetros seguintes:

- Número de funções básicas $n = 10$;
- Dimensão $D = 10$;
- $C = 2000$;
- Espaço de procura $[-5, 5]^D$;

Tabela 5.1: Pseudo-código do método de criação de funções compostas

```
Definir f, sigma, bias, f_bias, o, M, e as constantes C e z
do i=1,n
   wi=exp(-(sum(x(:)-o(i,:+)oold(i,:))^2)/(2*D*sigma(i)^2))
   fit(i)=f(((x-o(i)+oold(i))/lambda(i))*M(i))
   fmax(i)=f((z/lambda(i))*M(i))
   f(i)=C*fit(i)/fmax(i)
end do

!Soma e máximo de w----------------------------------------
Sumw=sum(w(:))
Maxw=max(w)

do i=1,n
   if (w(i)/=Maxw) then
      w(i)=w(i)*(1-Maxw^10)
   end if
   w(i)=w(i)/Sumw
end do

Fx=f_bias+sum(w(:)*[f(i)+bias(i)])
```

- $f\_\text{bias} = 0$;
- $\text{bias} = [0, 100, 200, 300, 400, 500, 600, 700, 800, 900]$.

Deste modo, a primeira função $f_1(\mathbf{x})$ será sempre aquela que possui o óptimo global, pois o seu $\text{bias}_1 = 0$. $\mathbf{o}_1, \mathbf{o}_2, ..., \mathbf{o}_9$ são gerados aleatoriamente no espaço de procura, excepto $\mathbf{o}_{10}$ que é definido como $[0, 0, ..., 0]$ de modo a aprisionar algoritmos capazes de convergir para o centro do espaço de procura.

Funções Básicas

As funções básicas usadas na construção das funções compostas são as seguinte:

- Função Esfera

$$f(\mathbf{x}) = \sum_{i=1}^{D} x_i^2 \tag{5.11}$$

- Função de Rastrigin

$$f(\mathbf{x}) = \sum_{i=1}^{D} (x_i^2 - 10\cos(2\pi x_i) + 10) \tag{5.12}$$

- Função de Weierstrass

$$f(\mathbf{x}) = \sum_{i=1}^{D} \left(\sum_{k=0}^{k_{\max}} [a^k \cos(2\pi b^k (x_i + 0.5))] \right) - D \sum_{k=0}^{k_{\max}} [a^k \cos(\pi b^k)] \tag{5.13}$$

  com $a = 0.5$, $b = 3$ e $k_{\max} = 20$

- Função de Griewank

$$f(\mathbf{x}) = \sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{5.14}$$

- Função de Ackley

$$f(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D}x_i^2}\right) - exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right) + 20 + e \qquad (5.15)$$

Função Composta 1

A primeira função composta (CF1) é constituída por:

- $f_{1-10}(\mathbf{x})$: Função Esfera;
- $[\sigma_1, \sigma_2, ..., \sigma_{10}] = [1, 1, ..., 1]$;
- $[\lambda_1, \lambda_2, ..., \lambda_{10}] = [5/100, 5/100, ..., 5/100]$.

Uma representação tridimensional da função obtida encontra-se representada na figura 5.1.

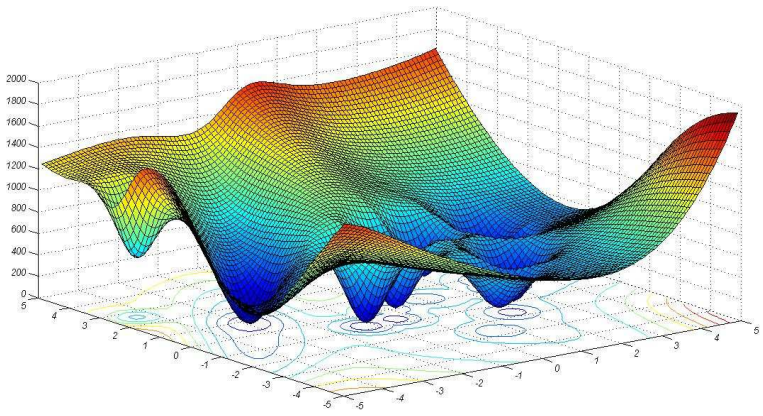


Figura 5.1: Representação tridimensional da função composta 1

Função Composta 2

A segunda função composta (CF2) é constituída por:

- $f_{1-10}(\mathbf{x})$: Função Griewank;
- $[\sigma_1, \sigma_2, ..., \sigma_{10}] = [1, 1, ..., 1]$;
- $[\lambda_1, \lambda_2, ..., \lambda_{10}] = [5/100, 5/100, ..., 5/100]$.

Uma representação tridimensional da função obtida encontra-se representada na figura 5.2.

Função Composta 3

A terceira função composta (CF3) é constituída por:

- $f_{1-10}(\mathbf{x})$: Função Griewank;
- $[\sigma_1, \sigma_2, ..., \sigma_{10}] = [1, 1, ..., 1]$;
- $[\lambda_1, \lambda_2, ..., \lambda_{10}] = [1, 1, ..., 1]$.

Uma representação tridimensional da função obtida encontra-se representada na figura 5.3.

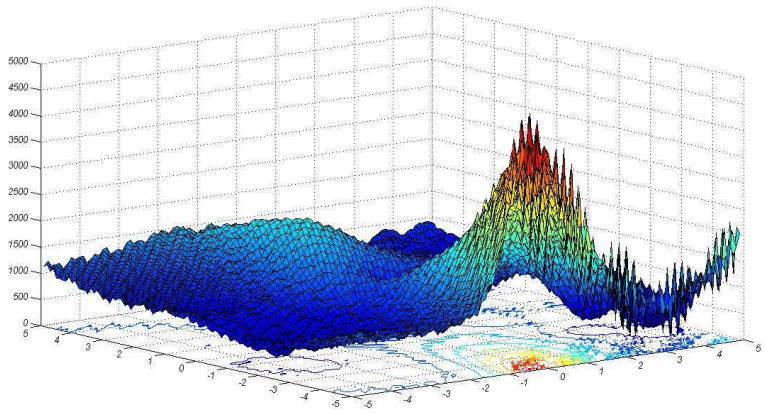


Figura 5.2: Representação tridimensional da função composta 2

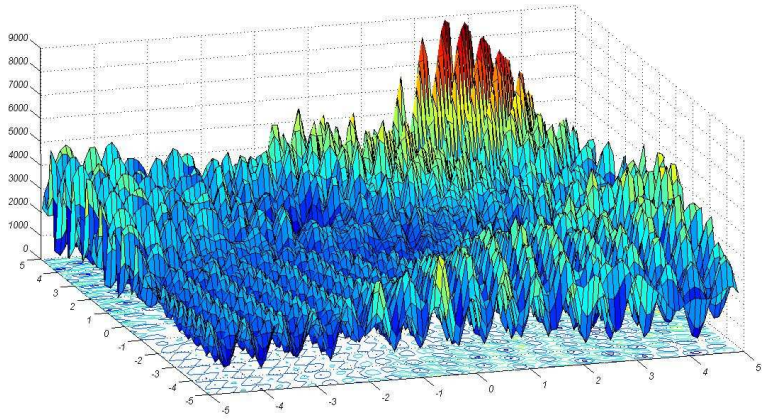


Figura 5.3: Representação tridimensional da função composta 3

Função Composta 4

A quarta função composta (CF4) é constituída por:

- $f_{1-2}(\mathbf{x})$: Função de Ackley; $f_{3-4}(\mathbf{x})$: Função de Rastrigin; $f_{5-6}(\mathbf{x})$: Função de Weierstrass; $f_{7-8}(\mathbf{x})$: Função de Griewank; $f_{9-10}(\mathbf{x})$: Função Esfera;
- $[\sigma_1, \sigma_2, ..., \sigma_{10}] = [1, 1, ..., 1]$;
- $[\lambda_1, \lambda_2, ..., \lambda_{10}] = [5/32, 5/32, 1, 1, 10, 10, 5/100, 5/100, 5/100, 5/100]$.

Uma representação tridimensional da função obtida encontra-se representada na figura 5.4.

Função Composta 5

A quinta função composta (CF5) é constituída por:

- $f_{1-2}(\mathbf{x})$: Função de Rastrigin; $f_{3-4}(\mathbf{x})$: Função de Weierstrass; $f_{5-6}(\mathbf{x})$: Função de Griewank; $f_{7-8}(\mathbf{x})$: Função de Ackley; $f_{9-10}(\mathbf{x})$: Função Esfera;

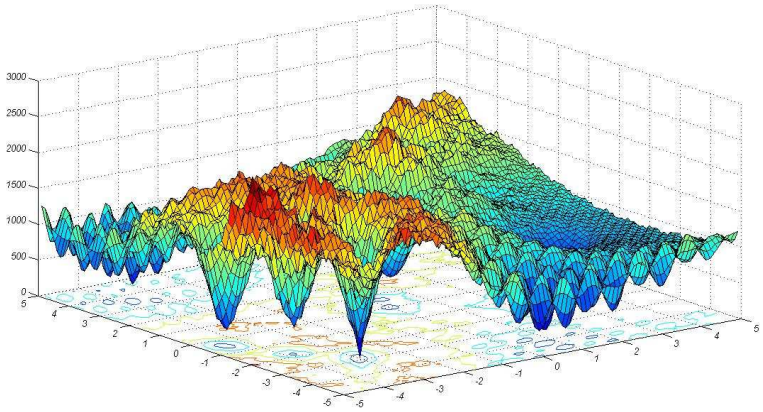


Figura 5.4: Representação tridimensional da função composta 4

- $[\sigma_1, \sigma_2, ..., \sigma_{10}] = [1, 1, ..., 1]$;
- $[\lambda_1, \lambda_2, ..., \lambda_{10}] = [1/5, 1/5, 10, 10, 5/100, 5/100, 5/32, 5/32, 5/100, 5/100]$.

Nesta função considera-se $\lambda_1 = \lambda_2 = 1/5$ de modo a obter-se uma área mais complexa na zona do óptimo global. Uma representação tridimensional da função obtida encontra-se representada na figura 5.5.

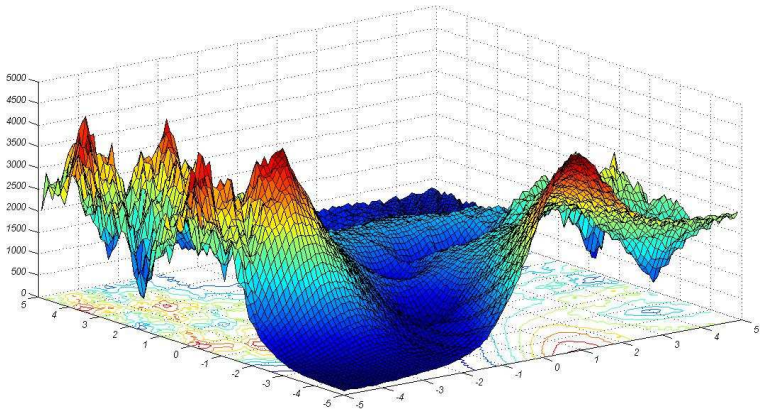


Figura 5.5: Representação tridimensional da função composta 5

Função Composta 6

A sexta função composta (CF6) é constituída por:

- $f_{1-2}(\mathbf{x})$ Função de Rastrigin, $f_{3-4}(\mathbf{x})$ Função de Weierstrass, $f_{5-6}(\mathbf{x})$ Função de Griewank, $f_{7-8}(\mathbf{x})$ Função de Ackley, $f_{9-10}(\mathbf{x})$ Função Esfera;
- $[\sigma_1, \sigma_2, ..., \sigma_{10}] = [0.1, 0.2, 0.0, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]$;
- $[\lambda_1, \lambda_2, ..., \lambda_{10}] = [0.1 \times 1/5, 0.2 \times 1/5, 0.3 \times 10, 0.4 \times 10, 0.5 \times 5/100, 0.5 \times 5/100, 0.7 \times 5/32, 0.8 \times 5/32, 0.9 \times 5/100, 5/100]$.

Uma representação tridimensional da função obtida encontra-se representada na figura 5.6.

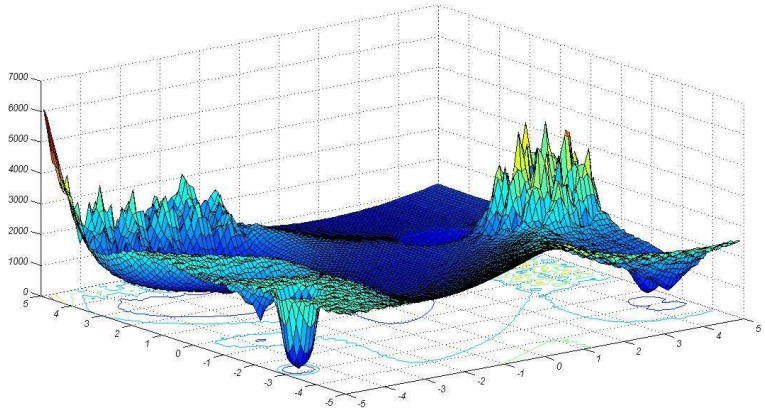


Figura 5.6: Representação tridimensional da função composta 6

5.1.3 Estudo dos Parâmetros Operacionais

De forma a determinar quais os valores adequados de I, NP, F_s e CR_s foram realizados estudos recorrendo às funções compostas anteriormente definidas. Em todos os estudos foram realizados vinte ensaios para cada valor do parâmetro a determinar. Tomou-se como critério de paragem $n_{\max} = 10^3$. Como configuração inicial considera-se uma população inicial de 10, probabilidade de cruzamento 0.15, factor de escala da solução 0.85, peso inercial crescente entre 0.4 e 0.9 e uma estratégia inicial DE/best/1/bin. Os valores dos parâmetros ρ e v_c são dados na tabela 5.2. A população considera-se como estagnada quando a solução não melhora após 750 avaliações pelo que esta é reinicializada. Neste caso CR_s passa a ser igual a 0.1, $F_s = 0.9$ e a estratégia utilizada DE/rand/1/bin.

Nº Avaliações	v_c	ρ
0	1×10^{-6}	2
$\mathrm{n_{max}}/3$	1×10^{-10}	10
$2 \times \mathrm{n_{max}}/3$	1×10^{-20}	100

Tabela 5.2: Valores dos parâmetros v_c e ρ

Estudo do Parâmetro I

O parâmetro I é de grande importância, uma vez que é este que controla o número de iterações realizadas por cada um dos algoritmos base. Optou-se por começar o processo de optimização pelo algoritmo de Evolução Diferencial que realiza, desta forma, as primeiras I iterações.

Na tabela 5.3 e figura 5.8(a) apresentam-se os resultados dos ensaios. Para cada valor de I de cada função composta apresentam-se os valores médios dos vinte ensaios e dentro de parêntesis o valor mínimo encontrado. Nas figuras 5.7(a)-(f) encontram-se representados as curvas para os valores médios das funções compostas para cada I estudado.

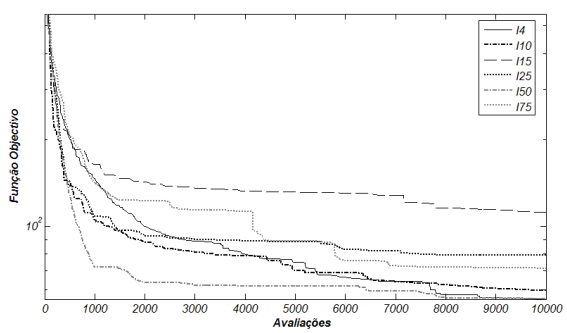
Como se pode observar na tabela 5.3 e figura 5.8(a) quando $I = 50$ o algoritmo obtém bons resultados. Para as funções compostas 2 e 3, considerando a referida escolha, o algoritmo possui o melhor valor da média e mínimo. Nas funções 4 e 6 possui a melhor média e na 5ª o melhor

Tabela 5.3: Médias e mínimos de vinte ensaios para estudo do parâmetro *I*

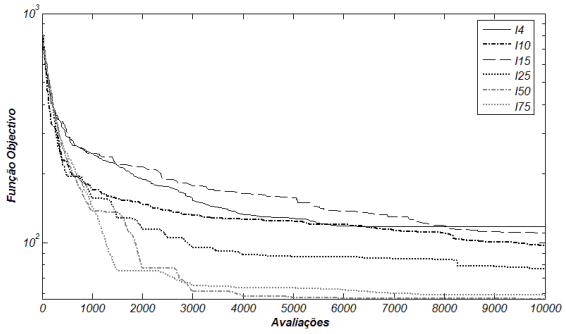
I	CF 1	CF 2	CF 3	CF 4	CF 5	CF 6
4	55.6645 (0.0019)	117.1094 (27.9123)	1038.7381 (381.6504)	391.8568 (337.9232)	95.4752 (12.9087)	562.5662 (363.7035)
10	59.7813 (0.0000)	96.9969 (38.3920)	835.9058 (394.8288)	390.3536 (300.8475)	94.8287 (10.9260)	522.4157 (360.6935)
15	112.052 (0.0000)	109.7430 (40.7453)	849.4123 (407.3240)	381.4866 (294.9719)	76.8234 (10.1863)	529.9411 (365.7981)
25	79.3308 (0.0000)	76.6946 (29.3664)	693.0057 (339.6222)	360.4327 (261.3613)	107.180 (13.0443)	567.4047 (383.6489)
50	55.9408 (0.0002)	56.8760 (25.7853)	485.5980 (118.1664)	313.8496 (253.5850)	64.1871 (6.9621)	508.2306 (362.4486)
75	71.1837 (0.0001)	58.9052 (20.0058)	766.3573 (387.1912)	304.9766 (263.8696)	91.6012 (4.3417)	519.2384 (363.8048)

Tabela 5.4: Médias e mínimos dos ensaios para estudo do parâmetro *NP*

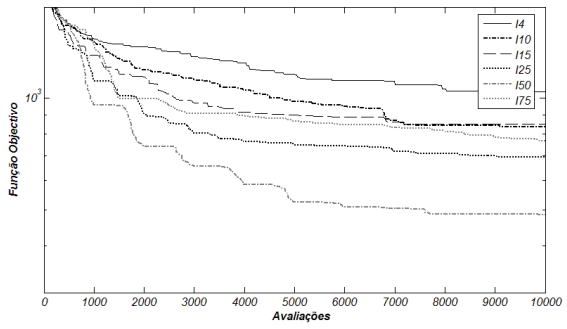
NP	CF 1	CF 2	CF 3	CF 4	CF 5	CF 6
5	94.3864 (0.0013)	72.0236 (18.6025)	513.1302 (115.9891)	380.9807 (261.5473)	75.7468 (7.0144)	625.8343 (363.3034)
10	55.9408 (0.0002)	56.8760 (25.7853)	485.5980 (118.1664)	313.8496 (253.5850)	64.1871 (6.9621)	508.2306 (362.4486)
15	67.2881 (0.0002)	72.7643 (23.6495)	488.7317 (118.8036)	302.1808 (242.6173)	78.2326 (4.0821)	558.7836 (361.7615)
20	63.1458 (0.0001)	45.2036 (16.4012)	499.517 (216.4332)	285.7923 (234.4694)	104.5631 (5.0182)	598.8680 (362.1076)
30	71.6672 (0.0001)	73.9819 (12.7149)	478.5496 (218.3134)	325.9735 (256.1747)	60.2089 (4.0533)	468.0292 (362.7741)



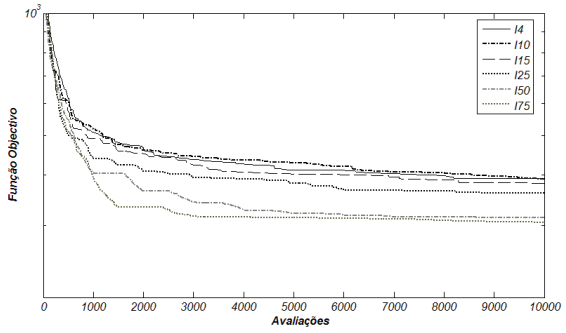
(a) Função composta 1



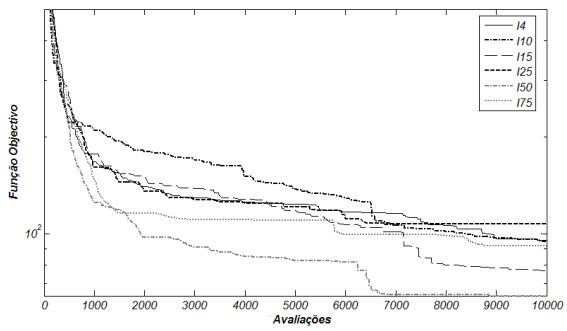
(b) Função composta 2



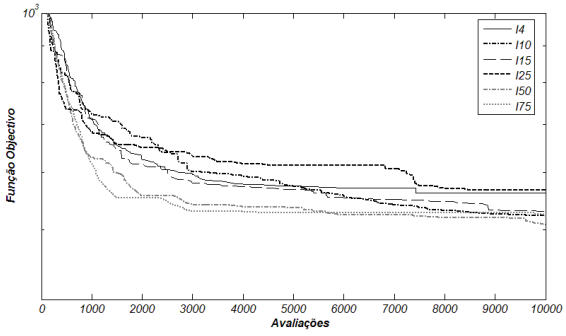
(c) Função composta 3



(d) Função composta 4



(e) Função composta 5



(f) Função composta 6

Figura 5.7: Curvas dos valores médios do estudo do parâmetro *I*

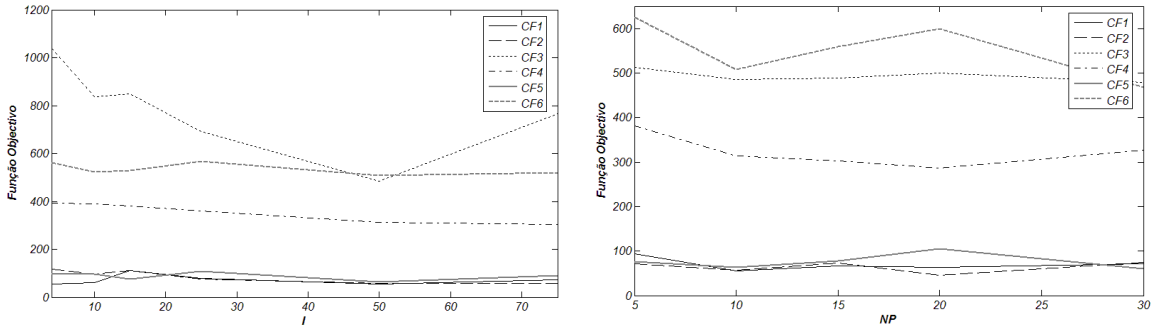
mínimo. A prestação na função composta 1 é muito próxima dos melhores valores encontrados. Nas figuras 5.7(a)-(f) pode observar-se que, quando $I = 50$, o algoritmo apresenta boas velocidades de convergência, especialmente nas funções 1, 3 e 6.

De acordo com os resultados obtidos, pode concluir-se que o valor de I ideal corresponde a 50.

Estudo do Parâmetro *NP*

O tamanho de população NP é um factor importante no processo de optimização. Uma população demasiado pequena não consegue realizar uma exploração detalhada do espaço de procura, enquanto que uma população demasiado grande necessita de um tempo de computação demasiadamente elevado. Torna-se imperativo encontrar o tamanho adequado de modo a manter o equilíbrio entre exploração e tempo de computação.

Com este fim, foi realizado um estudo para determinar qual o tamanho da população a usar. Foram executados vinte ensaios para populações dadas por $0.5D$ (5), D (10), $1.5D$ (15), $2D$ (20) e $3D$ (30), cujas médias são apresentadas na tabela 5.4 e figura 5.8(b). Os valores mínimos encontrados pelo algoritmo encontram-se na referida tabela entre parêntesis. Nas figuras 5.9(a)-(f) apresentam-se as curvas das médias dos vinte ensaios para cada uma das funções compostas.



(a) I (b) NP

Figura 5.8: Variação do parâmetro I e NP para as diferentes funções compostas

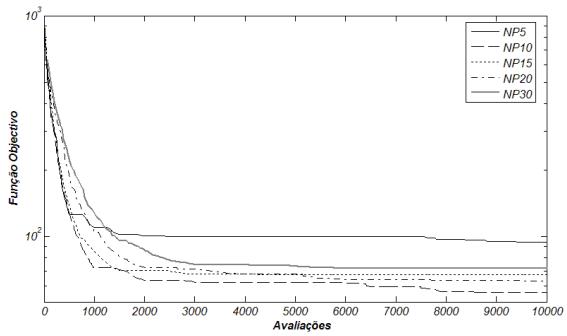
Da tabela 5.4 e figura 5.8(b) pode concluir-se que, na maioria das funções de teste, os valores mínimos atingidos pelos diferentes tamanhos de população encontram-se bastante próximos uns dos outros. De acordo com os valores médios obtidos, pode observar-se que os melhores desempenhos correspondem às populações de 10, 20 e 30.

Na curvas dadas nas figuras 5.9(a)-(f) pode observar-se que, para $NP = 10$, o algoritmo possui uma boa velocidade de convergência. Nas funções 1 a 4, o algoritmo converge mais lentamente quando a população é grande.

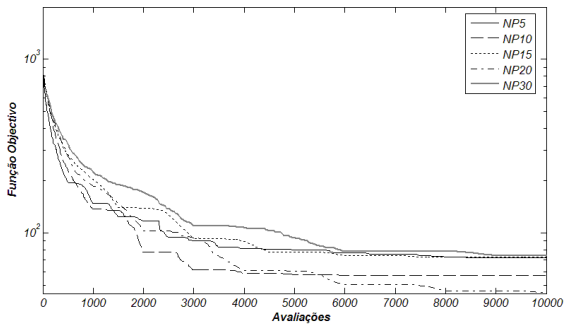
Assim, de acordo com os resultados obtidos, pode concluir-se que uma população inicial de tamanho 10 (D) é a melhor escolha.

Estudo do Parâmetro CR_s

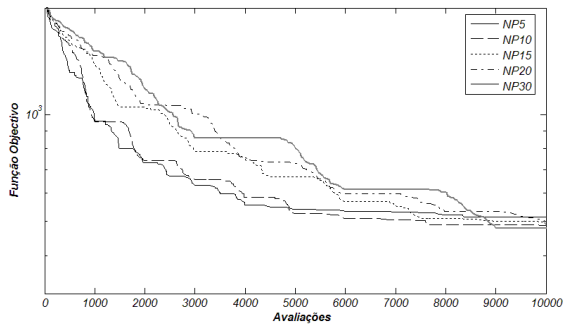
A probabilidade de cruzamento CR_s é um parâmetro que permite controlar a velocidade de convergência do algoritmo. Um valor de CR_s demasiado elevado pode conduzir à convergência prematura do algoritmo, não permitindo uma exploração mais cuidada do espaço de procura. Analisam-se, então, vários valores de CR_s pertencentes ao intervalo [0.05, 0.5]. Os valores médios dos vinte ensaios são listados na tabela 5.5 e figura 5.11(a). Os valores mínimos são dados na



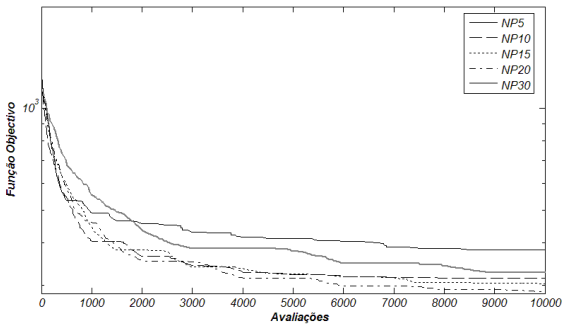
(a) Função composta 1



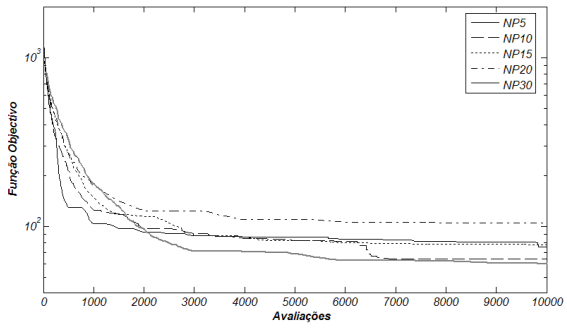
(b) Função composta 2



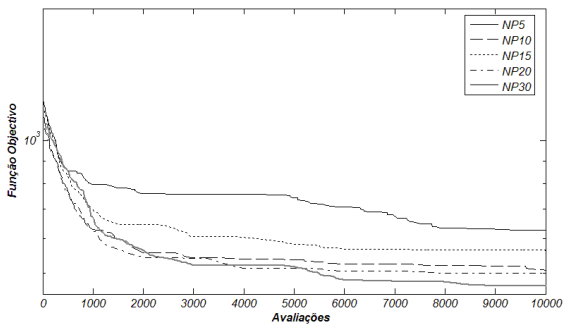
(c) Função composta 3



(d) Função composta 4



(e) Função composta 5



(f) Função composta 6

Figura 5.9: Curvas dos valores médios do estudo do parâmetro *NP*

tabela 5.5 entre parêntesis. Nas figuras 5.10(a)-(f) apresentam-se as curvas das médias dos vinte ensaios para cada uma das funções compostas para o parâmetro *CRs*.

Como se pode observar na tabela 5.5 e figura 5.11(a), os valores mínimos encontrados para cada um dos diferentes parâmetros nas várias funções de teste são, na maioria dos casos, muito próximos uns dos outros. Das figuras 5.10(a), (b), (d) e (e) retira-se que o algoritmo apresenta uma boa velocidade de convergência quando $CR_s = 0.15$.

Assim, e de acordo com os resultados obtidos, pode-se concluir uma probabilidade de cruzamento $CR_s = 0.15$ é ideal.

Estudo do Parâmetro F_s

De forma a combater a redução de diversidade causada pela selecção, é importante obter um factor de escala de solução F_s com magnitude suficiente para evitar uma convergência prematura [Price *et al.* 05]. De acordo com Zahari [Zahari 02], o valor de F_s mínimo que evite este efeito é dado por

$$F_{crit} = \sqrt{\frac{1 - CR/2}{NP}} \approx 0.304. \tag{5.16}$$

Para $F_s = 1$, a combinação de vectores torna-se indistinguível. Quando $F_s > 1$ as soluções tendem a ser mais demoradas e menos fiáveis. Por estas razões, realizou-se o estudo para valores de F_s contidos no intervalo $[0.5, 0.9]$. Os resultados da média dos vinte ensaios, bem como os mínimos, são apresentados na tabela 5.6. A médias encontram-se representadas graficamente na figura 5.11(b). Nas figuras 5.12(a)-(f) apresentam-se as curvas das médias dos vinte ensaios de cada uma das funções compostas para o parâmetro F_s.

Como se pode observar nas figuras 5.12(a), (b), (d) e (f), quando $F_s = 0.85$ o algoritmo apresenta uma boa velocidade de convergência e bons resultados médios. Desta forma, considera-se o factor de escala da solução ideal como sendo igual a 0.85.

5.1.4 Comparação com Outros Algoritmos

Nesta secção faz-se a comparação entre o desempenho do algoritmo desenvolvido e outros algoritmos existentes. Aqui consideram-se o algoritmo de Evolução Diferencial clássico (DE), Optimização por Bandos de Partículas clássico (PSO), Algoritmo Evolucionário (EA), algoritmo de Levenberg-Marquardt (LM), algoritmo de Recozimento Simulado (SA) e o algoritmo desenvolvido (HDEPSO[1]).

Foram realizados vinte ensaios para cada um dos algoritmos e o critério de paragem é o número de avaliações, dado por $n_{\max} = 10^3$. Os parâmetros escolhidos para cada um destes algoritmos são os seguintes:

- DE: População inicial de tamanho 10, $F = 0.58$ e $CR = 0.15$. A estratégia utilizada foi DE/best/1/bin;
- PSO: População inicial de tamanho 10, $c_1 = c_2 = 2$ e $w = 0.9$;
- EA: População inicial de tamanho 10, probabilidade de cruzamento 0.6 do tipo sbx e probabilidade de mutação 0.005 [Andrade-Campos 05];
- LM: Cáculo do Jacobiano por diferenças finitas com perturbação de 0.005 e critério de paragem por estagnação de 10^{-15}. Os valores iniciais de $\mathbf{x}$ são gerados aleatoriamente;

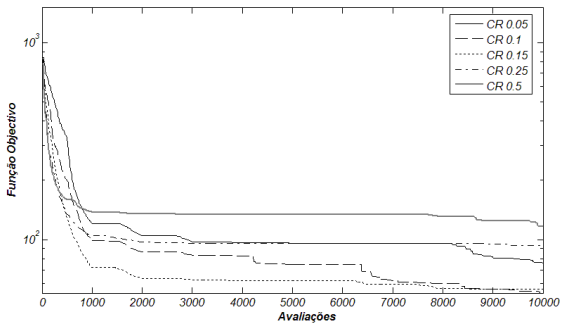
[1] *Hybrid Differential Evolution and Particle Swarm Optimization.*

Tabela 5.5: Médias e mínimos dos ensaios para estudo do parâmetro CR_s

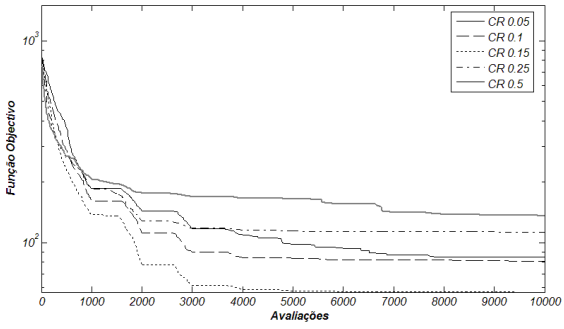
CR_s	CF 1	CF 2	CF 3	CF 4	CF 5	CF 6
0.05	75.8818 (0.0238)	84.7538 (20.0688)	502.1829 (120.8862)	351.1701 (259.5097)	116.3745 (8.4776)	552.4772 (365.3902)
0.1	53.5063 (0.0000)	80.8014 (27.3170)	489.4072 (109.3404)	343.9917 (247.0961)	71.0937 (9.5924)	474.9258 (362.3130)
0.15	55.9408 (0.0002)	56.8760 (25.7853)	485.5980 (118.1664)	313.8496 (253.5850)	64.1871 (6.9621)	508.2306 (362.4486)
0.25	92.9311 (0.0000)	112.8923 (26.3826)	456.0581 (78.7209)	324.6006 (238.0505)	69.0761 (7.8046)	526.5337 (359.1522)
0.5	116.4744 (0.0000)	135.8403 (12.6769)	389.1770 (76.4090)	444.5768 (234.0786)	177.9982 (4.3839)	555.5938 (360.0841)

Tabela 5.6: Médias e mínimos dos ensaios para estudo do parâmetro F_s

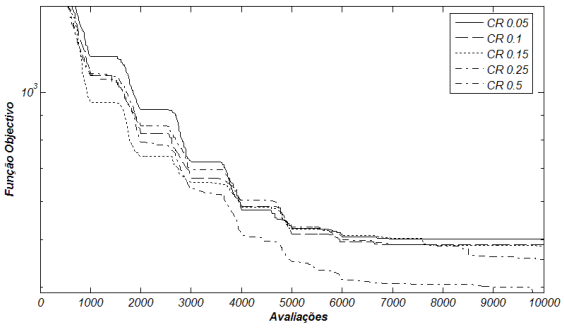
F_s	CF 1	CF 2	CF 3	CF 4	CF 5	CF 6
0.5	98.2731 (0.0009)	77.8354 (21.2904)	413.7861 (66.6051)	324.5057 (235.7127)	114.7004 (3.7150)	653.2284 (360.8028)
0.7	79.2339 (0.0016)	73.1533 (15.6269)	438.0122 (196.5240)	324.7384 (254.9575)	65.8534 (6.9420)	584.1084 (363.1424)
0.8	107.9365 (0.0018)	85.5813 (15.5918)	579.9494 (197.4282)	374.5033 (265.6203)	56.4475 (6.3845)	540.9819 (360.9308)
0.85	55.9408 (0.0002)	56.8760 (25.7853)	485.5980 (118.1664)	313.8496 (253.5860)	64.1871 (6.9621)	508.2306 (362.4486)
0.9	82.3240 (0.0001)	71.3768 (16.1068)	482.9647 (155.3462)	334.2194 (253.9412)	129.8311 (9.2211)	590.7449 (362.4442)



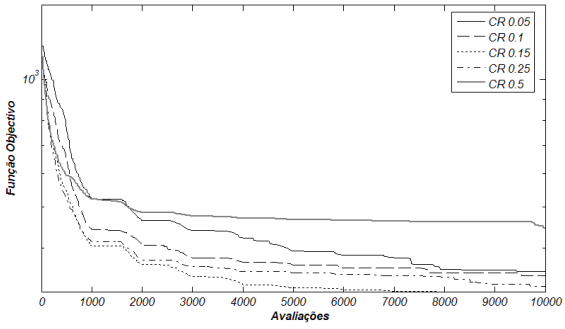
(a) Função composta 1



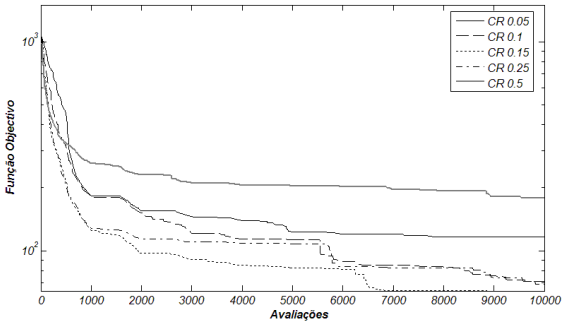
(b) Função composta 2



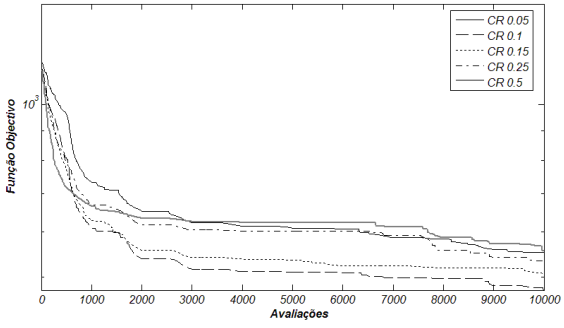
(c) Função composta 3



(d) Função composta 4

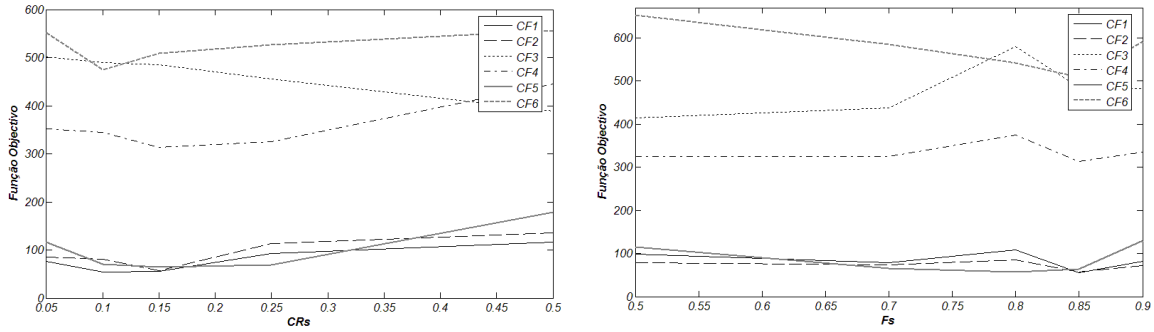


(e) Função composta 5



(f) Função composta 6

Figura 5.10: Curvas dos valores médios do estudo do parâmetro CR_s



(a) CR_s (b) F_s

Figura 5.11: Variação do parâmetro *CRs* e F para as diferentes funções compostas

- SA: temperatura inicial de 100, 5 ciclos de temperatura (N_T) nos quais são realizados 20 ciclos de passos (N_C). O tamanho de passo foi definido com sendo 1, $r_S = 0.9$ e $r_T = 0.85$;
- HDEPSO: população inicial de 10, probabilidade de cruzamento 0.15, factor de escala da solução 0.85, peso inercial crescente entre 0.4 e 0.9, $I = 50$ e uma estratégia inicial DE/best/1/bin. Os valores dos parâmetros ρ e v_c são dados na tabela 5.2. A população considera-se como estagnada quando a solução não melhora após 750 avaliações pelo que esta é reinicializada. Neste caso CR_s passa a ser igual a 0.1, $F_s = 0.9$ e a estratégia utilizada DE/rand/1/bin.

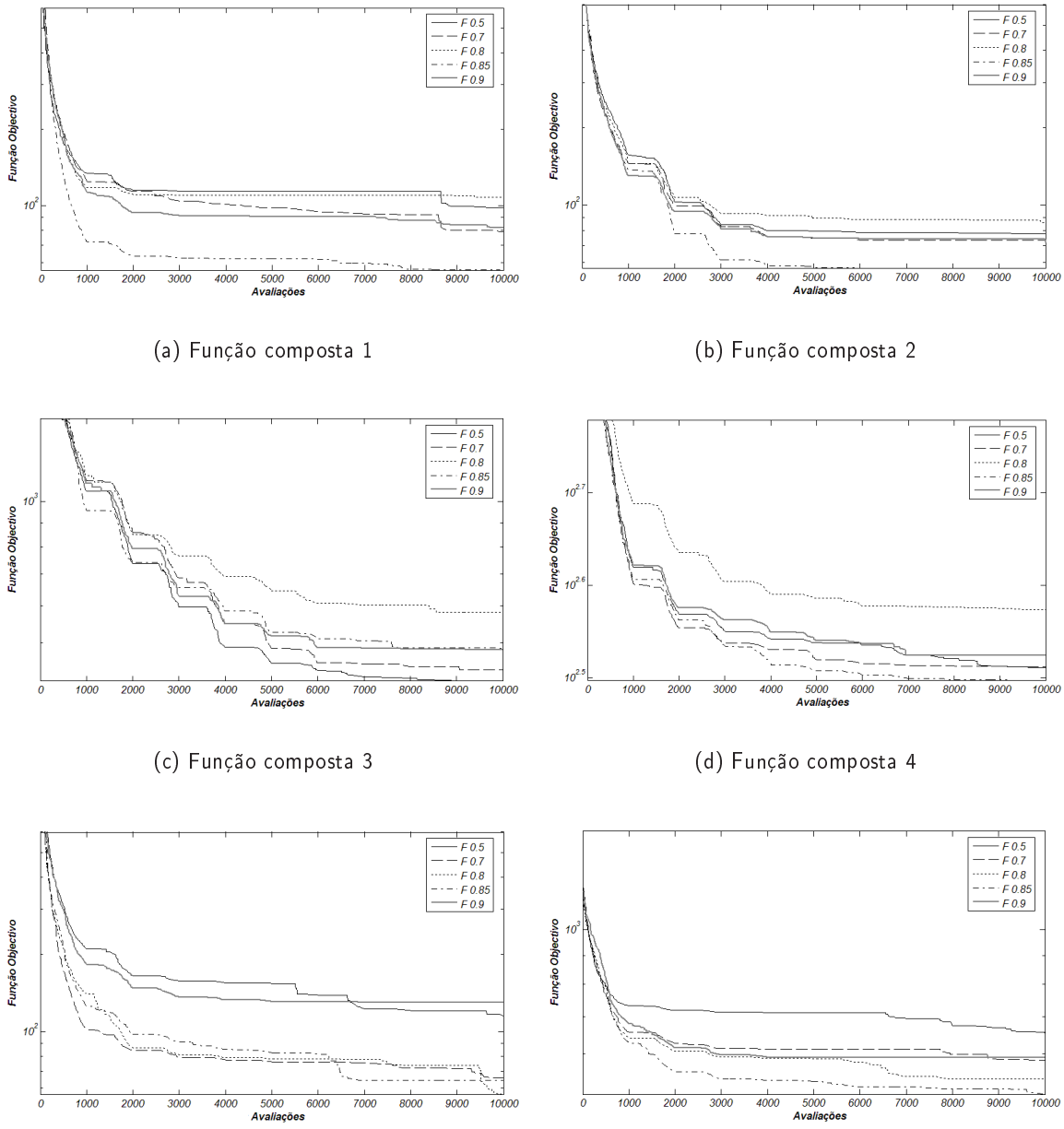
Os resultados obtidos encontram-se expostos na tabela 5.7 e nas figuras 5.13(a)-(f).

Tabela 5.7: Resultados de diferentes algoritmos para as funções compostas

Algoritmo	CF1	CF2	CF3
DE	55.1912 (0.0000)	81.9624 (20.1553)	711.8524 (364.1831)
PSO	207.2380 (111.3733)	228.4954 (146.3336)	1608.6924 (1083.1388)
EA	258.8623 (9.1364)	271.6045 (67.742)	602.5309 (221.2535)
LM	105.9884 (10.2951)	814.8125 (395.7754)	1687.7329 (407.2276)
SA	76.5029 (17.5476)	74.5102 (40.6462)	492.0991 (198.0999)
HDEPSO	55.9408 (0.0002)	56.8760 (25.7853)	485.5980 (118.1664)
Algoritmo	CF4	CF5	CF6
DE	376.8054 (266.9237)	75.5797 (8.2775)	597.7898 (356.5923)
PSO	625.4465 (562.9049)	235.5496 (135.9790)	680.8103 (612.5760)
EA	688.3543 (243.0680)	233.5868 (14.3192)	710.5381 (480.06677)
LM	1213.8741 (614.3252)	1486.3011 (250.2061)	885.3490 (556.1107)
SA	315.4351 (275.0703)	116.2803 (19.8668)	515.2219 (442.9517)
HDEPSO	313.8496 (253.5860)	64.1871 (6.9621)	508.2306 (362.4486)

Como se pode observar, o mínimo das funções apenas foi encontrado para a primeira função composta e apenas pelo algoritmo desenvolvido e pelo DE. As funções 3, 4 e 6 apresentam-se como sendo as de mais difícil minimização. Isto pode dever-se ao facto de estas apresentarem um número muito elevado de mínimos locais (funções compostas 3 e 4) e mínimos locais profundos que abrangem uma grande porção do espaço de procura (função composta 6).

O algoritmo desenvolvido apresenta um bom desempenho. Este consegue obter os melhores



(a) Função composta 1

(b) Função composta 2

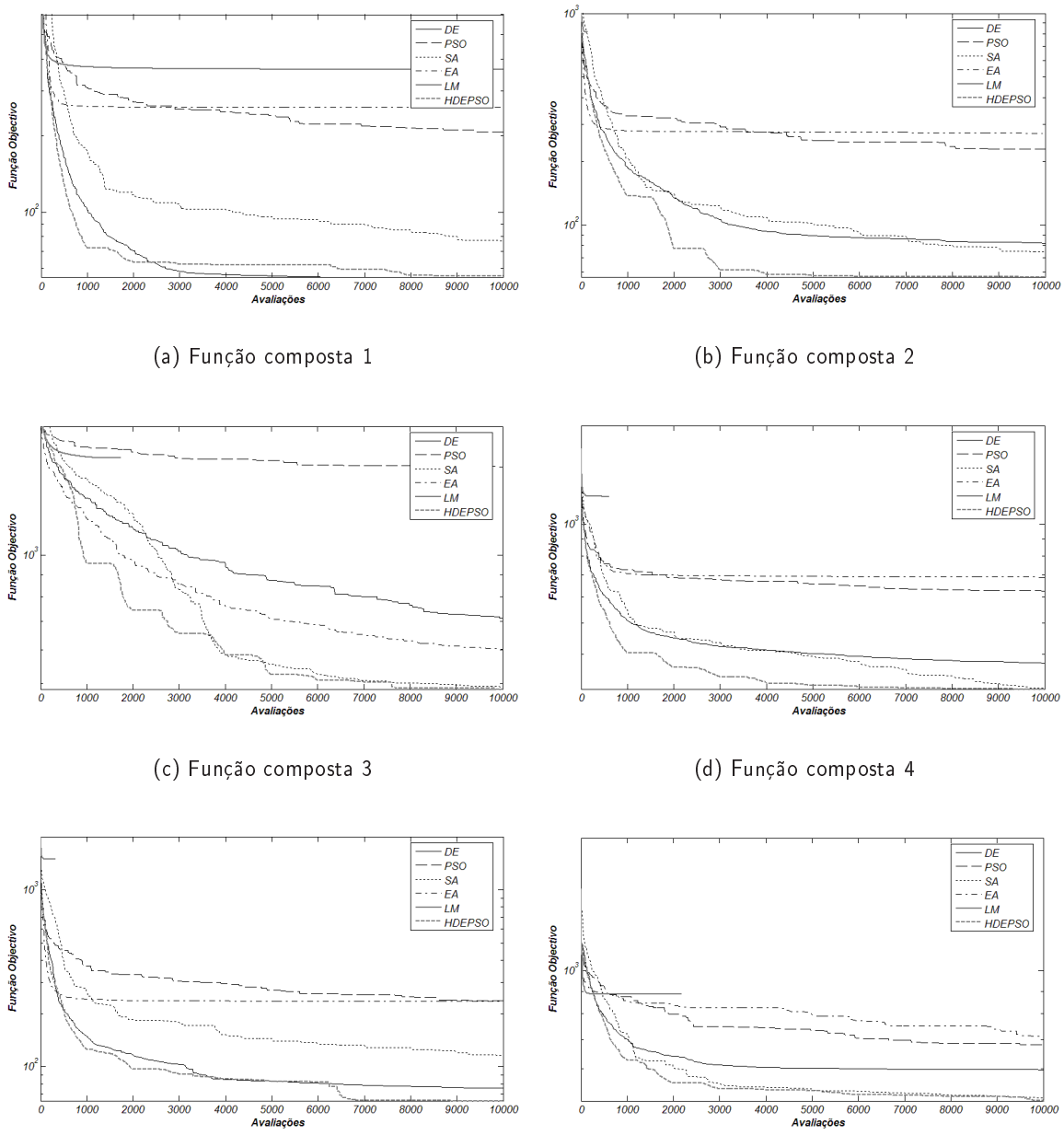
(c) Função composta 3

(d) Função composta 4

(e) Função composta 5

(f) Função composta 6

Figura 5.12: Curvas dos valores médios do estudo do parâmetro F_s



(a) Função composta 1

(b) Função composta 2

(c) Função composta 3

(d) Função composta 4

(e) Função composta 5

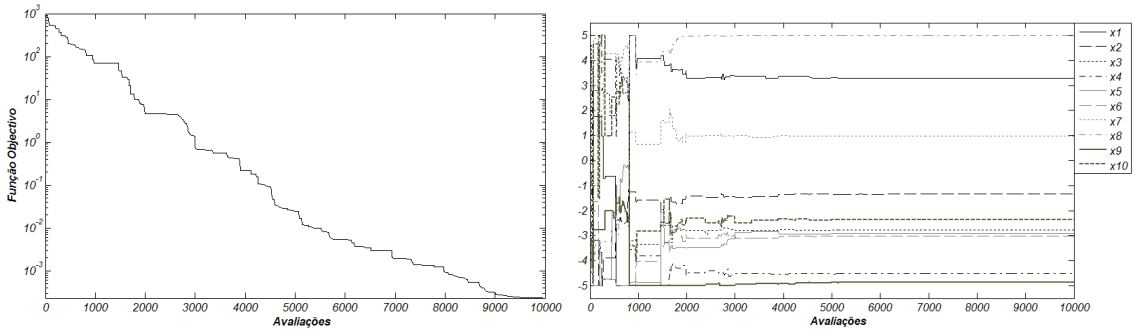
(f) Função composta 6

Figura 5.13: Resultados de diferentes algoritmos para as funções composta

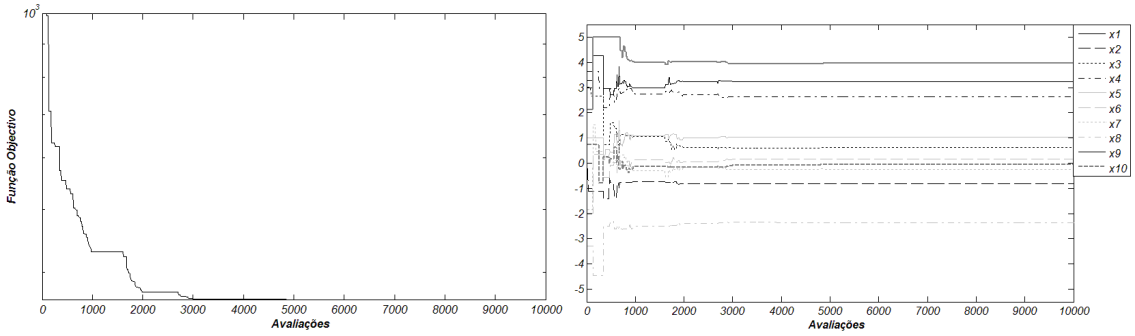
valores do mínimo para as funções compostas 3, 5 e 6, embora nas restantes apresente bons resultados, quando comparado com os restante algoritmos. O algoritmo demonstra ainda boa velocidade de convergência em todas as funções compostas criadas. Os algoritmos DE e PSO que se encontram na base do algoritmo desenvolvido apresentam, em geral, um desempenho inferior.

Pode ainda observar-se que o desempenho do algoritmo de Levenberg-Marquardt é inferior ao de todos os outros. Exceptuando a função composta 1, o LM realiza um número reduzido de avaliações, após as quais termina o processo de optimização. Estes resultados devem-se ao facto de este ser um método baseado no gradiente. Uma vez que as funções criadas possuem um elevado número de mínimos locais profundos, o LM fica estagnado com grande facilidade.

Nas figuras 5.14(a)-(f) apresentam-se os gráficos do melhor ensaio do algoritmo desenvolvido e a variação dos parâmetros no mesmo, para a função composta 1 e 6. Pretende-se com isto mostrar como evolui o algoritmo e as variáveis de optimização.



(a) Função composta 1



(b) Função composta 6

Figura 5.14: Melhor ensaio e variação dos parâmetros nas funções compostas 1 e 6

Como se pode observar na figura 5.14(a), existe na fase inicial do processo uma grande variação dos parâmetros a optimizar, seguido de uma estabilização destes. Isto significa que o algoritmo consegue realizar uma boa procura global nas primeiras avaliações, seguido de um refinamento da solução. Na figura 5.14(b) a variação dos parâmetros de optimização é inferior.

5.2 Problema das 3 Barras

5.2.1 Descrição do Problema

Neste problema pretende-se minimizar o volume da estrutura planar representada na figura 5.15. Nesse sentido, o problema pode ser expresso da seguinte forma

$$\text{minimizar } V(\mathbf{x}) = \sum_{i=1}^{3} L_i x_i, \tag{5.17}$$

quando sujeito a

$$0.01 \leq x_i \leq 2, \tag{5.18}$$

e

$$p_i \times u_i(x) \leq 1, \tag{5.19}$$

em que $V(\mathbf{x})$ representa o volume total da estrutura e L_i e x_i correspondem ao comprimento e área de cada elemento, respectivamente. O parâmetro u_i representa o deslocamento obtido devido à aplicação de uma carga p_i sobre a estrutura. É importante salientar que este problema trata-se de um problema de optimização multiconstrangido.

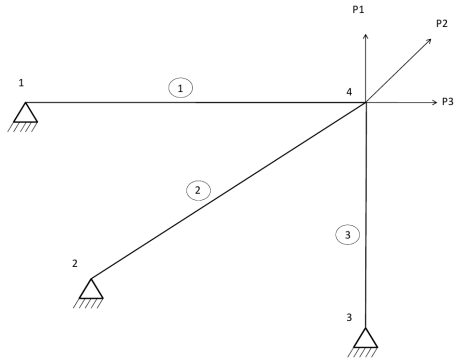


Figura 5.15: Esquema representativo do problema das 3 barras

5.2.2 Formulação do Problema

Devido à simplicidade deste problema, a assemblagem da matriz rigidez foi realizada de forma manual e, posteriormente, implementada em Fortran. Como o principal objectivo deste problema é calcular o volume das barras, pode considerar-se que todas as barras são constituídas pelo mesmo material e que o seu módulo de elasticidade é unitário.

As coordenadas dos nós são $N_1 = (-1, 0)$, $N_2 = (-1/\sqrt{2}, -1/\sqrt{2})$, $N_3 = (0, -1)$ e $N_4 = (0, 0)$ e os vectores de carga são dados por $p_1 = (1, 0)$, $p_2 = (1, 1)$ e $p_3 = (0, 1)$.

De acordo com o Método dos Elementos Finitos (ver secção 2.2), a matriz rigidez do sistema é dada por

$$\mathbf{k} = \begin{bmatrix} x_1 + 0.5x_2 & 0.5x_2 \\ 0.5x_2 & x_3 + 0.5x_2 \end{bmatrix}. \tag{5.20}$$

Seguidamente são calculados os deslocamentos fazendo

$$u_i = \mathbf{k}^{-1} p_i, \tag{5.21}$$

para $i = 1, 2, 3$. Estes deslocamentos são usados para verificar se alguma restrição foi quebrada. Considera-se, deste modo, que existem três restrições (uma para cada barra) dadas por

$$F_{penal,i} = p_i u_i(x). \tag{5.22}$$

O valor da função objectivo, e tomando as restrições pelo método das penalidades exteriores, é então dado por

$$F_{\text{obj}} = V(x) + w \sum_{i=1}^{3} (\max(0, F_{\text{penal},i} - 1)), \tag{5.23}$$

onde $w = 3 \times 10^5$ é o coeficiente de penalidade. Como se pode observar na equação anterior, o valor da função objectivo é dado pela soma do volume da estrutura e de penalidades por violação dos constrangimentos.

5.2.3 Resultados Obtidos

Foram realizados dez ensaios para cada um dos algoritmos. Considerou-se o critério de paragem como sendo $n_{\max} = 150$. Os parâmetros escolhidos para cada um dos algoritmos foram os seguintes:

- DE: População inicial de tamanho 4, $F = 0.85$ e $CR = 0.15$. A estratégia utilizada foi DE/best/1/bin;
- PSO: População inicial de tamanho 4, $c_1 = c_2 = 2$ e $w = 0.9$;
- EA: População inicial de tamanho 4, probabilidade de cruzamento 0.6 do tipo sbx, probabilidade de mutação 0.005 [Andrade-Campos 05];
- LM: Jacobiano calculado a partir de diferenças finitas com perturbação de 0.001 e critério de paragem por estagnação de 10^{-35}. Os valores iniciais de $\mathbf{x}$ são gerados aleatoriamente;
- SA: Temperatura inicial de 100, 2 ciclos de temperatura (N_T) nos quais são realizados 5 ciclos de passos (N_C). O tamanho de passo foi definido com sendo 1, $r_S = 0.9$ e $r_T = 0.85$;
- HDEPSO: População inicial de 4, probabilidade de cruzamento 0.15, factor de escala da solução 0.85, peso inercial crescente entre 0.4 e 0.9, $I = 4$ e uma estratégia inicial DE/best/1/bin. A população considera-se como estagnada quando a solução não melhora após 25 avaliações pelo que esta é reinicializada. Neste caso CR_s passa a ser igual a 0.1, $F_s = 0.9$ e a estratégia utilizada DE/rand/1/bin.

É importante referir que, na resolução deste problema, foram usadas populações de tamanho 4 embora a dimensão do problema seja 3. Esta escolha deve-se principalmente ao facto do DE e HDEPSO necessitarem de uma população mínima para poderem gerar combinações de vectores.

Os resultados obtidos podem ser vistos na tabela 5.8 e na figura 5.16. A evolução do algoritmo desenvolvido e dos parâmetros de optimização para o melhor ensaio encontram-se representados na figura 5.17.

Neste problema o LM foi o que obteve os melhores resultados, embora todos os algoritmo tenham chegado a valores muito semelhantes. Este resultado pode dever-se à simplicidade do problema.

Como se pode observar na figura 5.17, o algoritmo desenvolvido apresenta uma evolução progressiva, embora apresente alguns patamares de estagnação.

Os elevados valores iniciais de função objectivo apresentados por alguns dos algoritmos devem-se à penalização atribuída pela violação das restrições impostas. Esta é uma das desvantagens

Tabela 5.8: Resultados do problema das 3 barras

Algoritmo	Média	Mínimo
DE	3.136894	2.810576
PSO	2.970425	2.796920
EA	3.175360	2.770607
LM	2.920578	2.691929
SA	3.012863	2.901350
HDEPSO	2.957616	2.756577
Teórico	-	2.666666

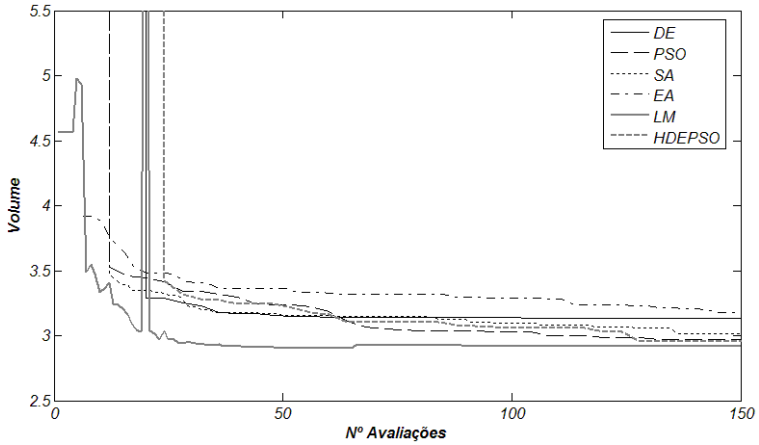


Figura 5.16: Resultados do problema das 3 barras

da aplicação de algoritmos não clássico em problemas de engenharia. Uma vez que a população inicial é gerada de forma aleatória dentro de espaço de procura, isto pode levar a que alguns dos membro dessa população sejam inicializado e mantidos durante as primeiras gerações em zonas nas quais as restrições do problema não são respeitadas. Este facto leva a que algumas avaliações sejam "desperdiçadas". Este é um aspecto importante a ter em conta quando os problemas a optimizar são complexos e as suas análises demoradas, uma vez que, nestas situações, o tempo de computação é um elemento essencial.

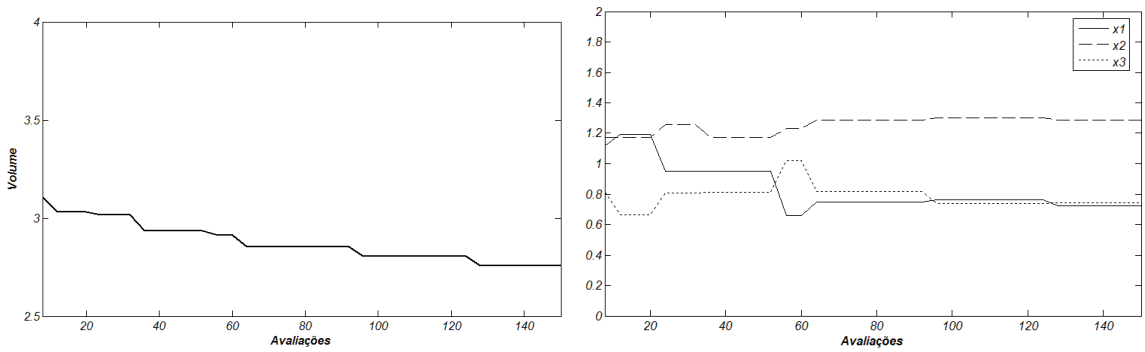


Figura 5.17: Melhor ensaio e variação dos parâmetros no problema das 3 barras

5.3 Problema da Cúpula com 120 Barras

5.3.1 Descrição do Problema

Neste problema pretende-se minimizar o peso da cúpula de 120 barras representada na figuras 5.18 e 5.19. O problema pode ser formulado, recorrendo à densidade (ρ_i), área (A_i) e comprimento

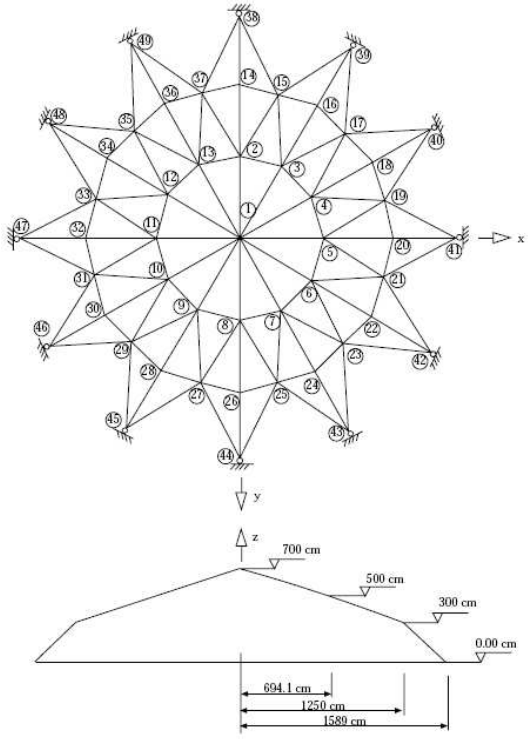


Figura 5.18: Cúpula de 120 barras [Kelesoglu e Ulker 05]

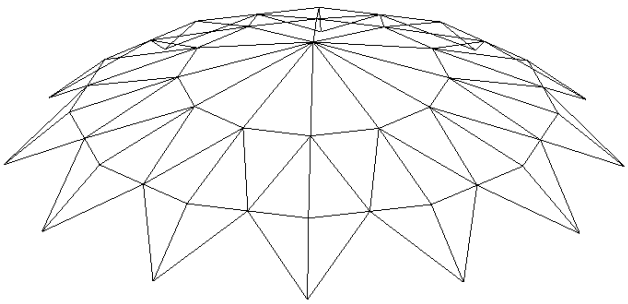


Figura 5.19: Representação tridimensional da cúpula de 120 barras

(L_i) de cada elemento, da seguinte forma:

$$\text{minimizar } W(x) = \sum_{i=1}^{120} \rho_i A_i L_i, \tag{5.24}$$

sujeito a

$$u_{\min} \leq u_j \leq u_{\max}, \quad \text{com } j = 1, 2, ..., 49 \tag{5.25}$$

$$\sigma_{\min} \leq \sigma_i \leq \sigma_{\max}, \quad \text{com } i = 1, 2, ..., 120 \tag{5.26}$$

$$A_{\min} \leq A_i \leq A_{\max} \tag{5.27}$$

em que $u_{\min}$, $u_{\max}$, $\sigma_{\min}$ e $\sigma_{\max}$ são os deslocamentos mínimos e máximos e tensões mínimas e máximas admissíveis, respectivamente. Segundo a norma AISC-ASD98 [AISC 89], as tensões máximas de tracção (σ_i^+) e compressão (σ_i^-) são dadas por

$$\begin{cases} \sigma_i^+ = 0.6\sigma_y, & \sigma_i \geq 0 \\ \sigma_i^-, & \sigma_i < 0 \end{cases} \tag{5.28}$$

em que

$$\sigma_i^- = \begin{cases} \left[\left(1 - \frac{\lambda_i^2}{2C_c^2}\right)\sigma_y\right] / \left(\frac{5}{3} + \frac{3\lambda_i}{8C_c} - \frac{\lambda_i^3}{8C_c^3}\right), & \lambda_i < C_c \\ \frac{12\pi^2 E}{23\lambda_i^2}, & \lambda_i \geq C_c \end{cases} \tag{5.29}$$

onde E é o módulo de elasticidade e σ_y a tensão de cedência do material. A razão de esbelteza λ_i pode ser calculada através de

$$\lambda_i = k\frac{L_i}{r_i}, \tag{5.30}$$

em que k é o factor de comprimento efectivo, L_i o comprimento do elemento e r_i o raio de giração. O parâmetro C_c é o coeficiente de esbelteza que divide as regiões elásticas e inelásticas durante a encurvadura dado por

$$C_c = \frac{\sqrt{2\pi^2 E}}{\sigma_y}. \tag{5.31}$$

O raio de giração r_i pode ser expresso em termos da área da secção através da equação

$$r_i = aA_i^b \tag{5.32}$$

em que, para secções tubulares, $a = 0.4993$ e $b = 0.6777$ [Kaveh e Talatahari 09].

5.3.2 Formulação do Problema

Para o cálculo das forças axiais em cada barra e deslocamento de cada nó foi utilizado o programa FRAN (ver secção 2.2.4). De modo a criar uma ligação entre este e o algoritmo de optimização foi escrito um código de interface. Esta interface transmite a informação dos valores das áreas de cada barra do algoritmo de optimização ao programa FRAN. Após a execução do programa FRAN, a interface lê os resultados obtidos e calcula o valor da função objectivo e da função penalidade. Os valores são exportados para o algoritmo de optimização e, mediante a satisfação, ou não, de um determinado critério de paragem, o algoritmo termina o processo optimização ou realiza uma nova iteração. O fluxograma representativo deste processo encontra-se exposto na figura 5.20

No problema em estudo, as barras podem ser separadas em 7 grupos (tabela 5.9), tornando-se desta forma num problema com apenas 7 variáveis de optimização. Todos os nós que não se encontram fixos estão sujeitos a uma carga vertical e a uma restrição de deslocamento máximo (tabela 5.10). Neste problema considera-se ainda $A_{\min} = 2$ cm^2 e $A_{\max} = 120$ cm^2.

O material utilizado possui uma densidade de 7971.810 kg/m^3 e assume-se que segue uma relação tensão-deformação bilinear (endurecimento linear). Na zona elástica considera-se um módulo de elasticidade $E_e = 210$ GPa e tensão de cedência de 400 MPa, enquanto na zona plástica o módulo de plasticidade é dado por $E_p = 0.4$ GPa e tensão de rotura 600 MPa.

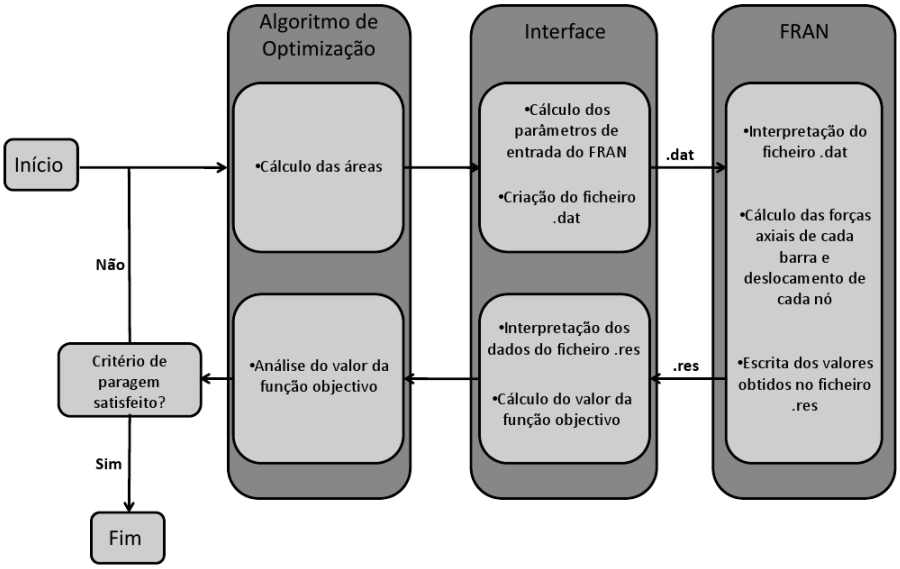


Figura 5.20: Fluxograma de interacção entre o algoritmo de optimização, a interface e o programa FRAN

Grupo	Elementos
1	1-12
2	13-24
3	25-36
4	37-60
5	61-84
6	85-96
7	96-120

Tabela 5.9: Distribuição por grupos das barras do problema da cúpula

5.3.3 Resultados

Foram realizados, para cada algoritmo, 10 ensaios cujo critério de paragem é $n_{\max} = 3500$. Os parâmetros escolhidos para cada um destes algoritmos foram os seguintes:

- DE: População inicial de tamanho 7, $F = 0.85$ e $CR = 0.15$. A estratégia utilizada foi DE/best/1/bin;
- PSO: População inicial de tamanho 7, $c_1 = c_2 = 2$ e $w = 0.9$;
- EA: População inicial de tamanho 7, probabilidade de cruzamento 0.6 do tipo sbx, probabilidade de mutação 0.005 [Andrade-Campos 05];

Nó	Carga (kN)	Deslocamento máximo (mm)
1	60	5
2-13	30	5
14-37	10	5

Tabela 5.10: Cargas e deslocamento admissíveis no problema da cúpula de 120 barras

- LM: Jacobiano calculado através de diferenças finitas com perturbação de 0.005 e critério de paragem por estagnação de 10^{-15}. Os valores iniciais de $\mathbf{x}$ são gerados aleatoriamente;
- SA: temperatura inicial de 100, 5 ciclos de temperatura (N_T) nos quais são realizados 20 ciclos de passos (N_C). O tamanho de passo foi definido com sendo 1, $r_S = 0.9$ e $r_T = 0.85$;
- HDEPSO: população inicial de 7, probabilidade de cruzamento 0.15, factor de escala da solução 0.85, peso inercial crescente entre 0.4 e 0.9, $I = 50$ e uma estratégia inicial DE/best/1/bin. A população considera-se como estagnada quando a solução não melhora após 500 avaliações pelo que esta é reinicializada. Neste caso CR_s passa a ser igual a 0.1, $F_s = 0.9$ e a estratégia utilizada DE/rand/1/bin.

Os resultados obtidos encontram-se expostos na tabela 5.11 e na figura 5.21.

Tabela 5.11: Resultados do problema da cúpula de 120 barras

Algoritmo	Média (kg)	Mínimo (kg)
DE	23122.6802	22800.2084
PSO	28999.5146	26868.2174
EA	28562.8847	27491.7224
LM	56519.1286	26511.5296
SA	23232.6174	23178.1016
HDEPSO	23454.6627	23241.3184

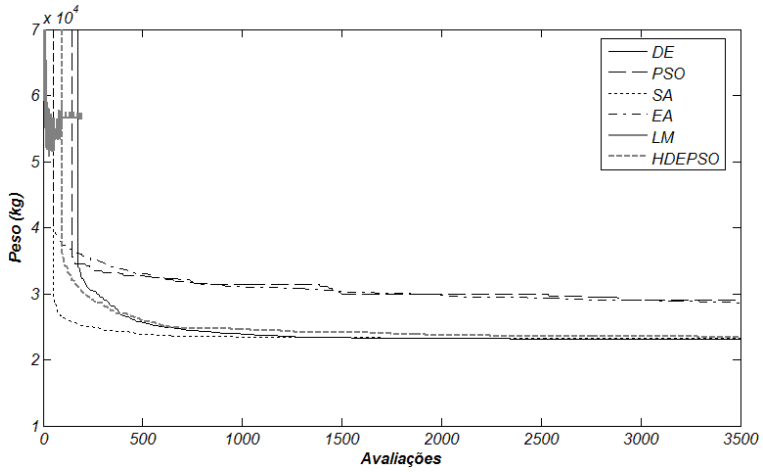


Figura 5.21: Resultados do problema da cúpula de 120 barras

Como se pode observar, em termos de valores médios, o DE, SA e HDEPSO apresentam valores muitos próximos, sendo o SA que possui a melhor velocidade de convergência. O algoritmo DE apresenta o melhor mínimo (22800.2084), seguido pelo SA (23178.1016) e HDEPSO (23241.3184).

O LM não conseguiu atingir bons resultados, parando o processo de optimização por volta das 250 avaliações.

Na figura 5.22 apresenta-se a evolução do processo de optimização para o melhor ensaio do algoritmo desenvolvido, bem como a variação dos seus parâmetros de optimização. A distribuição de deslocamentos da estrutura, para o referido ensaio, encontra-se representado na figura 5.23.

Como se pode observar, o algoritmo desenvolvido entra em estagnação por volta das 1000 avaliações, embora grande parte da evolução do processo de optimização ocorra até às 500 avaliações, demonstrando, desta forma, uma boa velocidade de convergência.

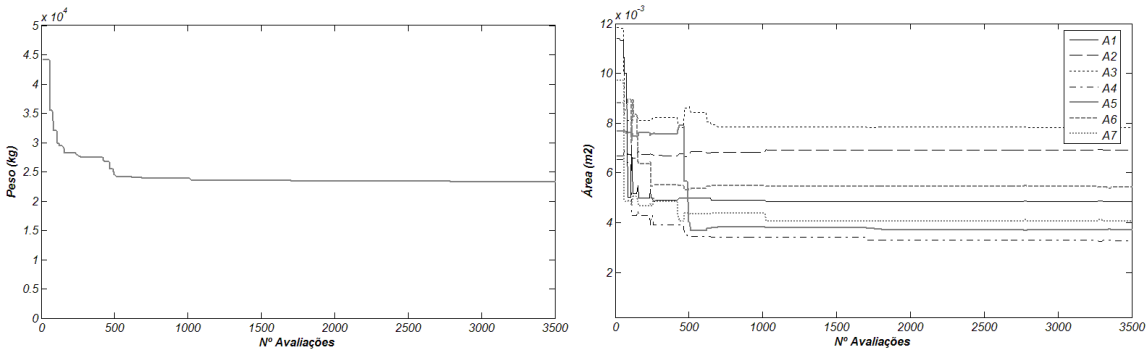


Figura 5.22: Melhor ensaio e variação dos parâmetros no problema da cúpula de 120 barras

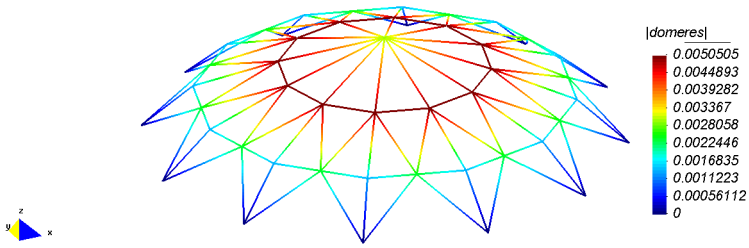


Figura 5.23: Deslocamentos sofridos pela estrutura da cúpula de 120 barras após o processo de optimização

5.4 Problema de Compressão de um Provete Cilíndrico

5.4.1 Descrição do Problema

Num ensaio de compressão, quando um provete cilíndrico é comprimido, as suas paredes laterais tomam uma forma arredondada, formando um barril. Neste problema, tal como exemplificado na figura 5.24, pretende-se determinar qual a forma inicial do provete de modo que, no final do estágio de compressão, as faces laterais do provete se encontrem na vertical, anulando deste modo o efeito barril supramencionado.

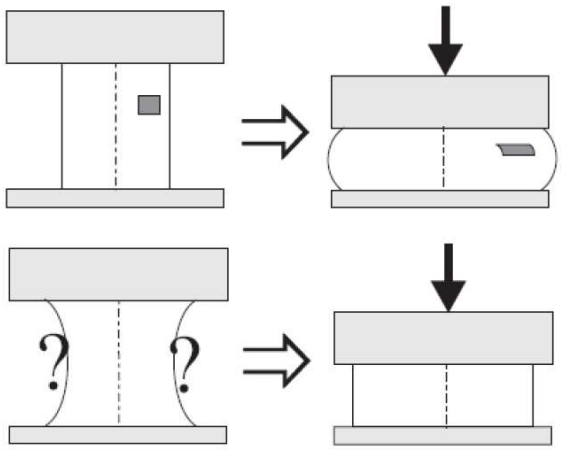


Figura 5.24: Esquema do problema de compressão do provete cilíndrico [Carvalho 07]

5.4.2 Formulação do Problema

Devido à simetria do problema apenas um quarto do provete necessita de ser modelado. Considerou-se um modelo com 120 mm de largura por 100 mm de altura com uma malha de 30×30 elementos do tipo CAX4R. O provete é comprimido em 20% da sua altura. Na face lateral escolheram-se 7 nós equidistantes (ver figura 5.25) que serão usados, recorrendo a splines (ver secção 2.3.1), para determinar as coordenadas dos restantes nós.

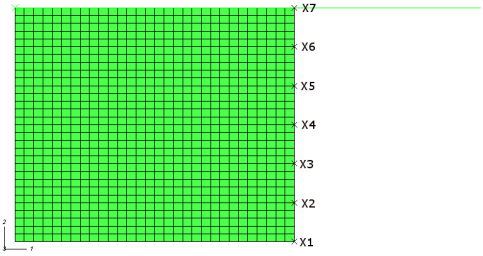


Figura 5.25: Representação das variáveis do problema de compressão do provete cilíndrico

Desta forma, o problema pode ser expresso como sendo a minimização de

$$F(x) = \sum_{i=1}^{n} \left[10^3 (x_i - x_{\text{exp},i})\right]^2, \tag{5.33}$$

sujeito a

$$90 \text{ mm} \ < \ x_i \ < \ 130 \text{ mm}, \tag{5.34}$$

em que x_i e $x_{\text{exp},i}$ correspondem aos deslocamentos após a simulação e aos deslocamentos desejados, respectivamente. Estes valores são multiplicados por um coeficiente de escalonamento de 10^3 e n representa o número de nós da face lateral.

Para proceder à resolução deste problema, foi criado um programa que integra um programa de interface baseado no trabalho desenvolvido por Carvalho [Carvalho 07]. O programa criado recebe à entrada os valores das variáveis e calcula as coordenadas dos restantes pontos da face do provete recorrendo a splines que são escritos no ficheiro de entrada do programa ABAQUS. Após a simulação, o ABAQUS exporta para um ficheiro os valores dos deslocamentos dos nós após a compressão do provete. Estes valores são lidos pela interface e comparados com os deslocamentos desejados. O fluxograma representativo do procedimento de resolução do problema encontra-se exposto na figura 5.26.

5.4.3 Resultados Obtidos

Foram realizados, para cada algoritmo, 5 ensaios cujo critério de paragem é $n_{\max} = 2100$. Os parâmetros escolhidos para cada um destes algoritmos foram os seguintes:

- DE: População inicial de tamanho 7, $F = 0.85$ e $CR = 0.15$. A estratégia utilizada foi DE/best/1/bin;
- PSO: População inicial de tamanho 7, $c_1 = c_2 = 2$ e $w = 0.9$;
- EA: População inicial de tamanho 7, probabilidade de cruzamento 0.6 do tipo sbx, probabilidade de mutação 0.005 [Andrade-Campos 05];
- LM: Calculo do Jacobiano através de diferenças finitas com uma perturbação de 0.005 e critério de paragem por estagnação de 10^{-15}. Os valores iniciais de $\mathbf{x}$ são gerados aleatoriamente;

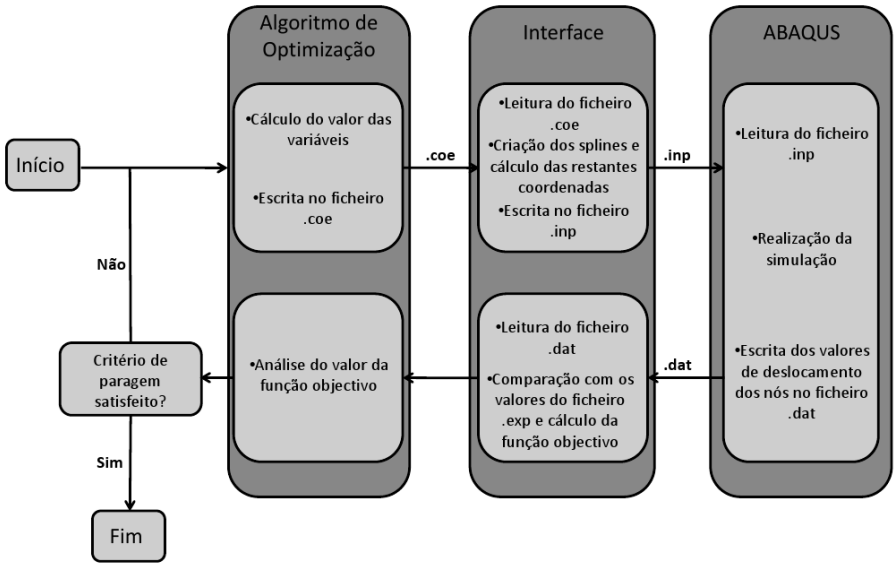


Figura 5.26: Fluxograma de interacção entre o algoritmo de optimização, a interface e o programa ABAQUS para o problema de compressão do provete cilíndrico

- SA: temperatura inicial de 100, 5 ciclos de temperatura (N_T) nos quais são realizados 20 ciclos de passos (N_C). O tamanho de passo foi definido com sendo 1, $r_S = 0.9$ e $r_T = 0.85$;
- HDEPSO: população inicial de 7, probabilidade de cruzamento 0.15, factor de escala da solução 0.85, peso inercial crescente entre 0.4 e 0.9, $I = 50$ e uma estratégia inicial DE/best/1/bin. A população considera-se como estagnada quando a solução não melhora após 500 avaliações pelo que esta é reinicializada. Neste caso CR_s passa a ser igual a 0.1, $F_s = 0.9$ e a estratégia utilizada DE/rand/1/bin.

Os resultados são apresentados na tabela 5.12 e na figura 5.27.

Tabela 5.12: Resultados do problema de compressão do provete cilíndrico

Algoritmo	Média	Mínimo
DE	2.6620	0.0500
PSO	329.766	249.1492
EA	8.2894	5.7732
LM	83.6913	0.5800
SA	50.4199	19.2070
HDEPSO	11.6300	3.0000

Neste problema, o algoritmo com melhor desempenho é o DE. O algoritmo LM converge muito rapidamente e termina o processo de optimização por volta das 200 avaliações, levando a que não consiga encontrar frequentemente bons valores para os parâmetros. O algoritmo desenvolvido, apesar de não obter os melhores resultados, consegue atingir valores satisfatórios inferiores aos obtidos pelo PSO, EA e SA.

Os gráficos da figura 5.28 mostram a evolução do algoritmo desenvolvido e dos seus parâmetros para o melhor ensaio. Como se pode observar, o algoritmo evolui de forma gradual, sendo a solução encontrada refinada a partir das 800 avaliações. Na figura 5.29 apresenta-se a forma do provete antes e após o estágio compressão para o referido ensaio.

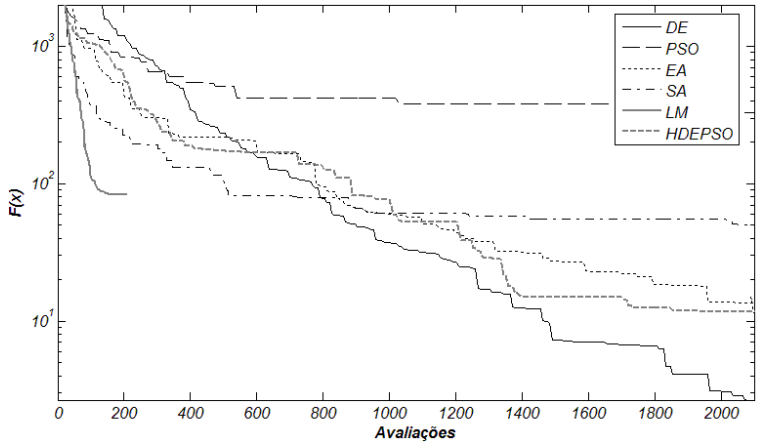


Figura 5.27: Resultados do problema de compressão do provete cilíndrico

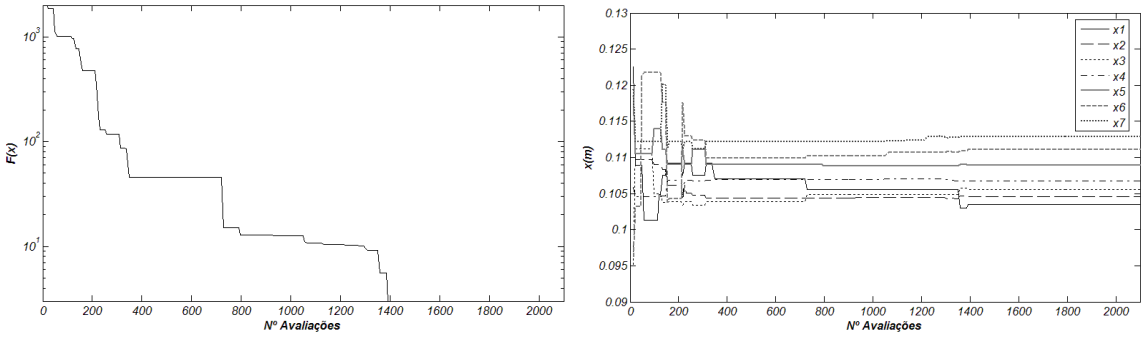


Figura 5.28: Melhor ensaio e variação dos parâmetros no problema de compressão do provete cilíndrico

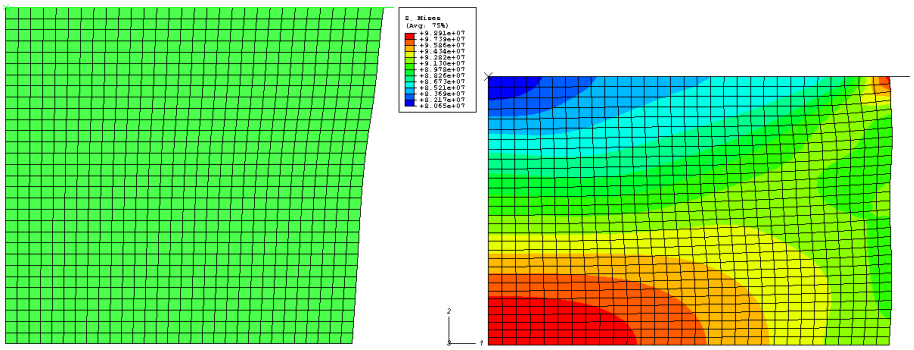


Figura 5.29: Resultados do problema de compressão do provete cilíndrico para o algoritmo desenvolvido

5.5 Problema da Placa com Furo

5.5.1 Descrição e Formulação do Problema

Neste problema pretende-se minimizar a área de uma placa com um furo no centro quando sujeito a cargas distribuídas e respeitando um limite de tensão equivalente máximo. Assim, o problema pode ser formulado da seguinte forma:

$$\text{minimizar } A(\mathbf{x}), \tag{5.35}$$

sujeito a

$$x_{i,\min} < x_i < x_{i,\max} \tag{5.36}$$

e

$$\sigma_{\max} < 7 \text{ N/mm}^2 \tag{5.37}$$

Devido à sua simetria, apenas um quarto da placa necessita de ser modelada, de acordo com o representado na figura 5.30. Para o cálculo das tensões na placa recorre-se ao programa de simulação pelo Método dos Elementos Finitos ABAQUS.

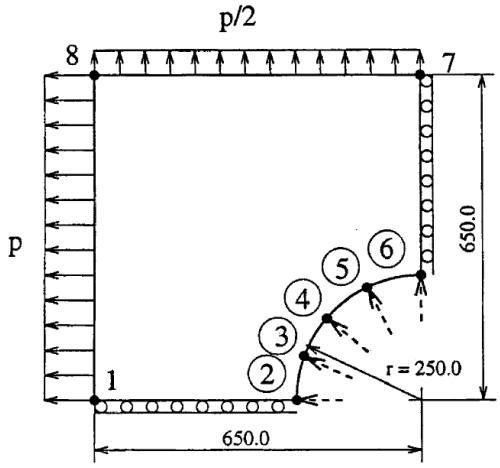


Figura 5.30: Representação do problema do prato com furo [Papadrakakis e Lagaros 03]

Tal como no estudo realizado por Papadrakakis e Lagaros [Papadrakakis e Lagaros 03], as variáveis de optimização correspondem aos raios 2 a 6 (ver figura 5.30). Tomando coordenadas cilíndricas, os ângulos de cada um destes raios são dados por

$$\theta_i = (i-2)\frac{\pi}{8}, \text{ com } i = 2, ..., 6. \tag{5.38}$$

A partir destes valores é possível determinar as coordenadas polares de 5 nós que são, posteriormente, usadas para criar um polinómio de 4º grau que define a forma do furo.

Para resolução deste problema foi criada uma interface que estabelece a ligação entre o algoritmo de optimização e o programa ABAQUS. O algoritmo começa por escrever os valores das variáveis a optimizar num ficheiro que será lido pelo programa de interface. Este programa interpreta os valores presentes no referido ficheiro e, de acordo com os ângulos, converte os pontos em coordenadas cartesianas. Posteriormente, é determinado o polinómio de 4º grau através do método de interpolação de Lagrange (ver secção 2.3.2) e calculadas as coordenadas dos restantes nós de definição do furo. Os valores são escritos num ficheiro parametrizado que é lido pelo ABAQUS. Os resultados do programa ABAQUS são importados pela interface que calcula a função

objectivo tendo em conta a área da placa e as penalidades por violação das restrições. Este valor é escrito num ficheiro *.txt* que será lido pelo algoritmo de optimização que, de acordo com os critérios de paragem, realiza outra iteração ou termina o processo. O fluxograma do processo descrito encontra-se representado na figura 5.31.

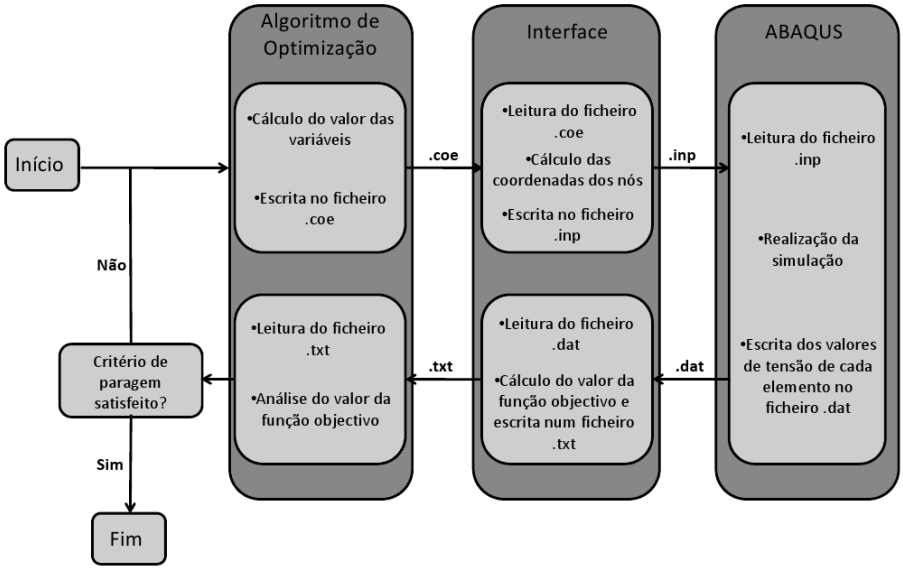


Figura 5.31: Fluxograma de interacção entre o algoritmo de optimização, interface e programa ABAQUS para o problema da placa com furo

Neste problema é admitido um estado plano de tensão e material isotrópico ($E = 210$ GPa e $\nu = 0.3$). A magnitude de p é de 0.65 N/mm^2. Os raios encontram-se no intervalo [250, 649] mm.

5.5.2 Resultados Obtidos

Foram realizados, para cada algoritmo, 5 ensaios cujo critério de paragem é $\mathrm{n_{max}} = 3500$. Os parâmetros escolhidos para cada um destes algoritmos foram os seguintes:

- DE: População inicial de tamanho 5, $F = 0.85$ e $CR = 0.15$. A estratégia utilizada foi DE/best/1/bin;
- PSO: População inicial de tamanho 5, $c_1 = c_2 = 2$ e $w = 0.9$;
- EA: População inicial de tamanho 5, probabilidade de cruzamento 0.6 do tipo sbx, probabilidade de mutação 0.005 [Andrade-Campos 05];
- LM: Cálculo do Jacobiano pelo método das diferenças finitas com uma perturbação de 0.005 e critério de paragem por estagnação de 10^{-15}. Os valores iniciais de $\mathbf{x}$ são gerados aleatoriamente;
- SA: temperatura inicial de 100, 5 ciclos de temperatura (N_T) nos quais são realizados 20 ciclos de passos (N_C). O tamanho de passo foi definido com sendo 1, $r_S = 0.9$ e $r_T = 0.85$;
- HDEPSO: população inicial de 5, probabilidade de cruzamento 0.15, factor de escala da solução 0.85, peso inercial crescente entre 0.4 e 0.9, $I = 50$ e uma estratégia inicial DE/best/1/bin. A população considera-se como estagnada quando a solução não melhora após 500 avaliações pelo que se reinicializa a população. Neste caso CR_s passa a ser igual a 0.1, $F_s = 0.9$ e a estratégia utilizada DE/rand/1/bin.

Os resultados são apresentados na tabela 5.13 e na figura 5.32.

Tabela 5.13: Resultados do problema da placa com furo

Algoritmo	Média (mm^2)	Mínimo (mm^2)
DE	216370.3685	211139.2510
PSO	221284.4593	214062.6972
EA	222817.2525 212540.7490	
LM	281029.18	267192.4213
SA	222258.3375	210375.8125
HDEPSO	217465.6672	211993.2982

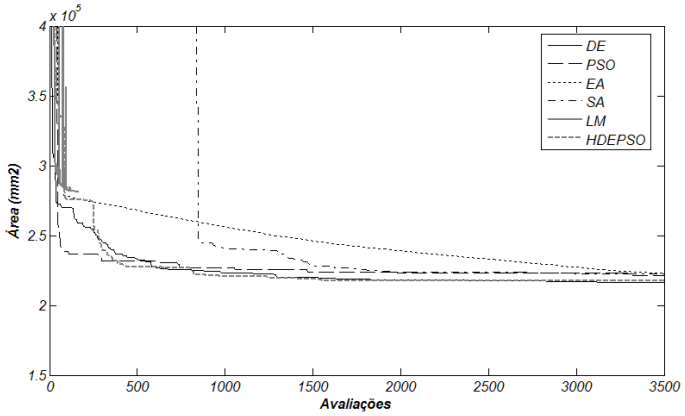


Figura 5.32: Resultados do problema da placa com furo

Neste problema, embora o algoritmo desenvolvido não apresente os melhores resultados em termos de média ou mínimo, o seu desempenho é bom. Este é semelhante ao dos algoritmos base (DE e PSO), tanto em velocidade de convergência como em resultados obtidos.

O algoritmo SA, embora obtenha bons valores finais, começa o processo de optimização com valores muito elevados. Em problemas mais complexos e que necessitem de elevados tempos de computação, este algoritmo poderá não ser a melhor escolha, uma vez que nestas situações o número de avaliações realizadas é essencial.

O algoritmo LM obtém os piores resultados para este problema. O LM realiza um reduzido número de avaliações (cerca de 250) antes de entrar em estagnação e terminar o processo de optimização. Este algoritmo apresenta ainda uma grande variação do valor da função objectivo durante o processo de optimização. Esta variação pode dever-se ao facto do algoritmo realizar passos demasiado grandes, fazendo com que as restrições do problema não sejam respeitadas.

Na figura 5.33 apresenta-se a evolução do algoritmo desenvolvido e respectivos parâmetros de optimização para o melhor ensaio. Como se pode observar, o algoritmo de optimização converge rapidamente, pelo que entra em estagnação após as 800 avaliações. Na figura 5.34 apresenta-se a forma final da placa após o processo de optimização para o referido ensaio. Como se pode observar, existem zonas da placa que se encontram sob a máxima tensão admissível.

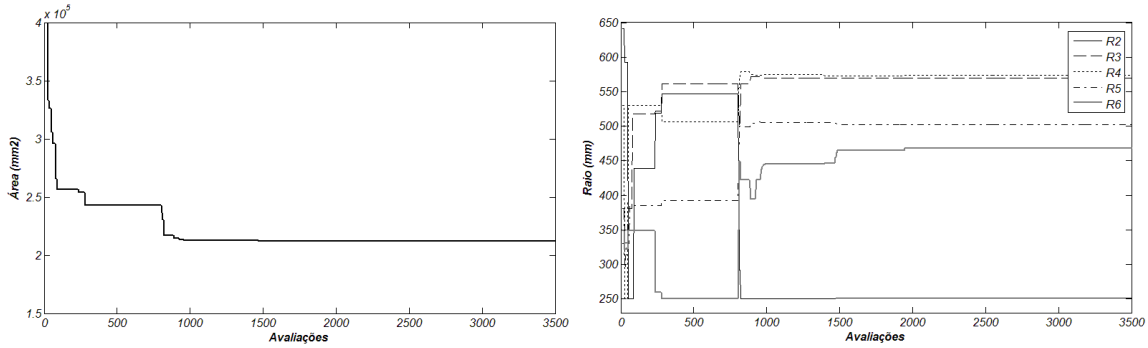


Figura 5.33: Melhor ensaio e variação dos parâmetros no problema do prato com furo

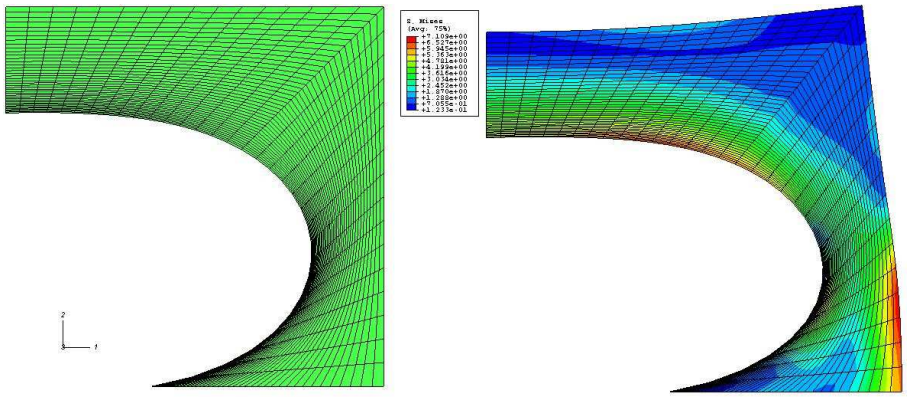


Figura 5.34: Forma final da placa com furo central após optimização pelo algoritmo desenvolvido

5.6 Identificação de Parâmetros de um Modelo Elasto-Plástico

5.6.1 Descrição e Formulação do Problema

Neste problema pretende-se realizar a identificação dos parâmetros de um modelo elasto-plástico para o aço E220BH e, consequentemente, conseguir um ajuste perfeito aos pontos experimentais. Matematicamente, a equação do modelo é dada por

$$\sigma_{11} = \frac{F}{A_0}, \tag{5.39}$$

em que F é a força aplicada sobre o provete de área A_0. Se a tensão de tracção σ_{11} for inferior à tensão de limite elástico R_0, então a deformação plástica é dada por

$$\varepsilon_{11}^p = 0. \tag{5.40}$$

Caso contrário, obtém-se a partir de

$$\varepsilon_{11}^p = -\frac{1}{\gamma} \ln\left(1 - \gamma \frac{\sigma_{11} - R_0}{C}\right), \tag{5.41}$$

em que γ e C são dois coeficientes de endurecimento. Por fim, a deformação axial segundo o eixo principal é dada por

$$\varepsilon_{11}^p = \frac{\sigma_{11}}{E} + \varepsilon_{11}^p, \tag{5.42}$$

e a deformação axial pode ser avaliada através de

$$\varepsilon_{22} = -\nu\frac{\sigma_{11}}{E} - \frac{1}{2}\varepsilon_{11}^{p}. \tag{5.43}$$

Neste tipo de problemas a função objectivo a minimizar é dada por

$$f(\mathbf{x}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\epsilon_i^{sim} - \epsilon_i^{exp}\right)^2}, \tag{5.44}$$

em que $\mathbf{x} = \{\gamma, C, R_0, E\}$ é o vector com os valores dos parâmetros, n o número de pontos experimentais, ϵ_i^{sim} e ϵ_i^{exp} são, respectivamente, as deformações do ponto i experimental e numérico.

5.6.2 Resultados Obtidos

Foram realizados, para cada algoritmo, 10 ensaios cujo critério de paragem é $n_{\max} = 500$. Os parâmetros escolhidos para cada um destes algoritmos foram os seguintes:

- DE: População inicial de tamanho 5, $F = 0.85$ e $CR = 0.15$. A estratégia utilizada foi DE/best/1/bin;
- PSO: População inicial de tamanho 5, $c_1 = c_2 = 2$ e $w = 0.9$;
- EA: População inicial de tamanho 5, probabilidade de cruzamento 0.6 do tipo sbx, probabilidade de mutação 0.005[Andrade-Campos 05];
- LM: Perturbação por diferenças finitas de 0.005 e critério de paragem por estagnação de 10^{-15}. Os valores iniciais de $\mathbf{x}$ são gerados aleatoriamente;
- SA: temperatura inicial de 100, 5 ciclos de temperatura (N_T) nos quais são realizados 20 ciclos de passos (N_C). O tamanho de passo foi definido com sendo 1, $r_S = 0.9$ e $r_T = 0.85$;
- HDEPSO: população inicial de 5, probabilidade de cruzamento 0.15, factor de escala da solução 0.85, peso inercial crescente entre 0.4 e 0.9, $I = 25$ e uma estratégia inicial DE/best/1/bin. A população considera-se como estagnada quando a solução não melhora após 50 avaliações pelo que se reinicializa a população. Neste caso CR_s passa a ser igual a 0.1, $F_s = 0.9$ e a estratégia utilizada DE/rand/1/bin.

Os resultados obtidos apresentam-se na tabela 5.14 e na figura 5.35.

Tabela 5.14: Resultados do problema de identificação de parâmetros

Algoritmo	Média	Mínimo
DE	29.8920	7.8241
PSO	15.6357	6.5985
EA	169.9245	39.1360
LM	82.5517	4.7819
SA	9.9761	6.3661
HDEPSO	11.1737	4.7991

Como se pode observar o algoritmo desenvolvido apresenta bons resultados finais, superiores aos obtidos pelos algoritmos base. O LM termina o processo de optimização antes de atingir as

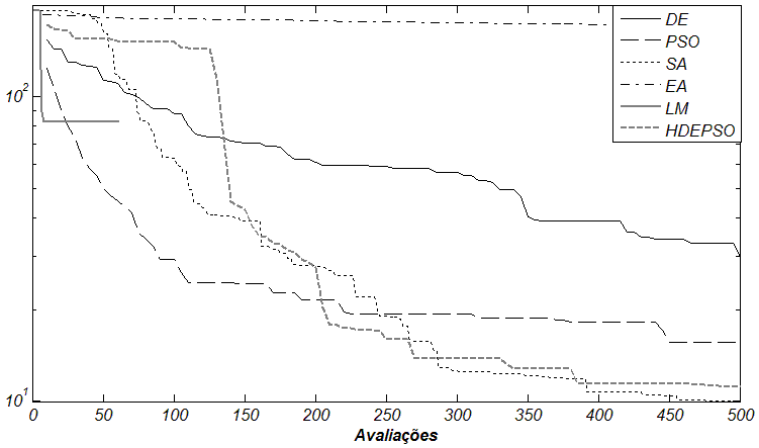


Figura 5.35: Resultados para o problema de identificação de parâmetros

100 avaliações da função objectivo e o EA entra em estagnação rapidamente. Estes algoritmos apresentam elevados valores médios. Este facto pode dever-se à sensibilidade do problema aos parâmetros R_0 e C, que, por se encontrarem num operador logarítmico, podem conduzir a valores não possíveis, fazendo com que os algoritmos entrem em estagnação.

Na figura 5.36 apresenta-se a curva ajustada aos pontos experimentais através do algoritmo desenvolvido. Como se pode observar, na zona plástica o HDEPSO consegue realizar um bom ajuste aos pontos experimentais.

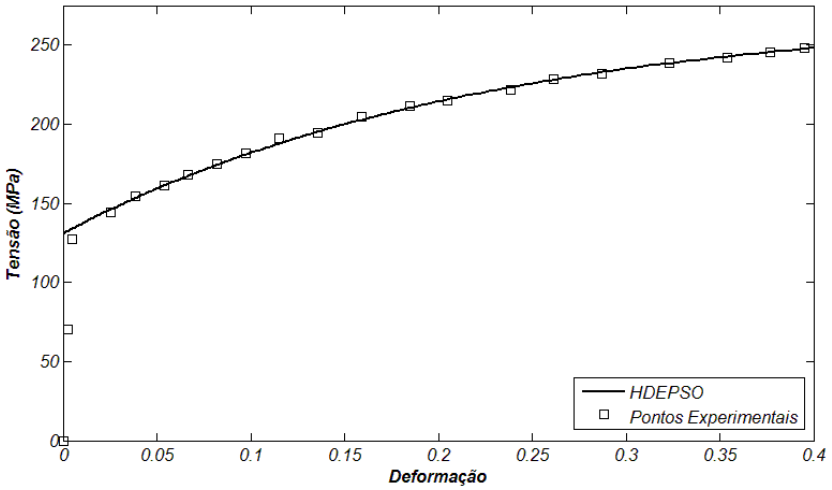


Figura 5.36: Curva ajustada pelo algoritmo desenvolvido

Capítulo 6

Discussão e Conclusões Gerais

Apresentam-se as conclusões gerais e discutem-se os resultados obtidos na realização deste trabalho.

O objectivo principal deste trabalho consiste em desenvolver um algoritmo baseado na teoria evolucionária que possa ser aplicado em problemas específicos de Engenharia Mecânica e Mecânica Computacional. Para tal, são escolhidos como base os algoritmos de Evolução Diferencial e Optimização por Bandos de Partículas. Aplicam-se, em cada um dos referidos métodos, diferentes estratégias que visam melhorar aspectos como a capacidade de exploração global e velocidade de convergência. O algoritmo desenvolvido consiste numa implementação em série dos algoritmos base modificados.

Após a implementação do algoritmo, e numa primeira fase, este é validado recorrendo a um conjunto de funções de teste. Estas possuem características especificas de forma a testar de forma rigorosa o desempenho do algoritmo. Nesta fase, o algoritmo desenvolvido (HDEPSO) é comparado com os seus algoritmos base, bem como um Algoritmo Evolucionário, algoritmo de Recozimento Simulado e o método clássico de Levenberg-Marquardt. Na maioria das funções de teste, o HDEPSO apresenta os melhores valores médios da função objectivo e boas velocidades de convergência. Os valores mínimos encontrados pelo algoritmo desenvolvido são melhores ou aproximados aos obtidos pelos restantes algoritmos. Nesta fase é ainda importante realçar que o algoritmo de Levenberg-Marquardt obteve os piores resultados em todas as funções compostas. Estes resultados devem-se ao facto de, por ser um método clássico baseado no gradiente, este segue a direcção dos menores gradientes fazendo com que o algoritmo fique preso com facilidade em mínimos locais.

Numa segunda fase, o algoritmo desenvolvido é aplicado a problemas de optimização estrutural e de identificação de parâmetros de modelos constitutivos. Nestes problemas, o HDEPSO apresenta bons resultados e boas velocidades de convergência. Embora não seja o que obtém os melhores resultados, os valores obtidos num dado problema são, na maioria dos casos, equiparáveis aos alcançados pelo algoritmo com o melhor desempenho nesse mesmo problema.

Pode ainda concluir-se que o uso de métodos de optimização não clássicos em problemas de engenharia mecânica é uma opção viável. Estes métodos permitem resolver com fiabilidade problemas com vários mínimos locais, com várias restrições e/ou discretos. Além disto, os métodos evolucionários, diferenciais e baseados em comportamentos de animais são de fácil implementação devido à sua estrutura simples e número reduzido de parâmetros operacionais. No entanto, estes também possuem algumas desvantagens, como a determinação do valor ideal dos parâmetros operacionais (como o tamanho de população ou probabilidade de cruzamento) que variam de acordo com o problema em estudo. No caso de problemas que necessitem de um tempo de computação

elevado em cada avaliação, estes métodos poderão não ser os mais adequados. Uma vez que nestes métodos é gerada uma população aleatória no espaço da procura, alguns membros poderão ser colocados em zonas que violem restrições, levando a que sejam desperdiçadas avaliações.

Lista de Figuras

Lista de Tabelas

Bibliografia

[Ali 07] Ali, M.M., *Differential Evolution With Preferential Crossover*, European Journal of Operational Research, Elsevier, **181**, pp. 1137-1147, 2007.

[AISC 89] American Institute of Steel Construction (AISC), *Manual of Steel Construction Allowable Stress Design*, 9^{th} Ed., Chicago, IL, 1989.

[Andrade-Campos 05] Andrade-Campos, A., *Modelação e Análise Numérica do Comportamento Mecânico e Térmico de Ligas de Alumínio*, Tese de Doutoramento, Universidade de Aveiro, 2005.

[Antoniu e Lu 07] Antoniou, A., Lu, W.S., *Pratical Optimization - Algorithms and Engineering Aplications*, Springer, New York, 2007.

[Ashlock 05] Ashlock, D., *Evolutionary Computation for Modeling and Optimization*, Springer, 2005.

[Babu e Jehan 03] Babu, B.V., Jehan, M.M.L, *Differential Evolution for Multi-Objective Optimization*, The 2003 Congress on Evolutionary Computation (CEC '03), **4**, pp. 2696-2703, 2003.

[Belegundu e Chandrupatla 99] Belegundu, A.D., Chandrupatla, T.R., *Optimization concepts and applications in engineering*. Prentice-Hall, Upper Saddle River, New Jersey, 1999.

[Byrd *et al.* 90] Byrd, R.H., Dert, C.L., Kan, A.H.G.R., Schnabel, R.B., *Concurrent Stochastic Methods for Global Optimization*, Mathematical Programming, **46**, 1990.

[Chong e Zak 01] Chong, E.K.P., Zak, S., *An Introduction to Optimization*, 2^{nd} *Ed.*, John Wiley & Sons, Inc., 2001.

[Carvalho 07] Carvalho, J.F.S., *Metodologia de Optimização de Processos de Conformação Plástica*, Tese de Mestrado, Universidade de Aveiro, 2007.

[Chapra e Canale 90] Chapra, S.C, Canale, R.P., *Numerical Methods for Engineers*, 2^{nd} Ed., McGraw-Hill International Editions,1990.

[Christensen e Klarbring 09] Christensen, P.W., Klarbring, A., *An Introduction to Structural Optimization*, Springer, 2009.

[Coelho e Mariani 06] Coelho, Ld.S., Mariani, V.C., *Combining of Chaotic Differential Evolution and Quadratic Programming for Economic Dispatch Optimization With Valve-Point Effect*, IEEE Transactions on Power Systems, **21**, pp. 989-996, 2006.

[Eberhart e Kennedy 95] Eberhart, R.C.,Kennedy, J., *A New Optimizer Using Particle Swarm Theory*, Proceedings of the Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan, pp. 39-43, 1995.

[Fish e Belytschko 07] Fish, J., Belytschko, T., *A First Course in Finite Elements*, John Wiley & Sons, Ltd, 2007.

[García-Villoria e Pastor 09] García-Villoria, A., Pastor, R., *Introducing Dynamic Diversity Into a Discrete Particle Swarm Optimization*, Computers and Operations Research, **36**, pp. 951-966,2009.

[GiD Reference Manual] GiD Reference Manual em http://www.gid-usa.com/support/gid_toc.subst, consultado a 10 de Maio de 2009.

[Goffe *et al.* 94] Goffe, W.L., Ferrier, G.D., Roger, J., *Global Optimization of Statistical Functions With Simulated Annealing*, Journal of Econometrics, **60**, pp. 65-99, 1994.

[Gong *et al.* 08] Gong, W., Cai, Z., Jiang, L., *Enhancing the Performance of Differential Evolution Using Orthogonal Design Method*, Applied Mathematics and Computation, **206**, pp. 56-69, 2008.

[Hibbit *et al.* 08] Hibbitt, Karlsson, Sorensen, *ABAQUS/Standard - User's manual*, EUA, Hibbitt, Karlsson & Sorensen, Inc, 1998.

[Holland 09] Holland J. H., *Genetic Algorithms* em http://www.econ.iastate.edu/tesfatsi/holland.GAIntro.htm consultado a 31 de Março de 2009.

[Jie *et al.* 08] Jie, J., Zeng, J., Han, C., Wang, Q., *Knowledge-Based Cooperative Particle Swarm Optimization*, Applied Mathematics and Computation, **205**, pp. 861-873, 2008.

[Kaveh e Talatahari 09] Kaveh, A., Talatahar,i S., *Particle Swarm Optimizer, Ant Colony Strategy and Harmony Search Scheme Hybridized for Optimization of Truss Structures*, Computers and Structures, **87**, pp. 267-283, 2009.

[Kelesoglu e Ulker 05] Kelesoglu, O., Ulker, M., *Fuzzy Optimization Geometrical Nonlinear Space Truss Design*, Turkish J. Eng. Environ. Sci., **29**, pp. 321-329, 2005.

[Kennedy e Eberhart 09] Kennedy, J., Eberhart, R.C., *Particle Swarm Optimizer* em http://www.engr.iupui.edu/ shi/Coference/psopap4.html, consultado em Março de 2009.

[Kleinermann e Ponthot 03] Kleinermann, J.P., Ponthot, J.P., *Parameter Identification and Shape/Process Optimization in Metal Forming Simulation*, Journal of Materials Processing Technology, **139**, pp. 521-526, 2003.

[Liang *et al.* 05] Liang, J.J., Suganthan, P.N., Deb, K., *Novel Composition Test Functions for Numerical Global Optimization*, IEEE Swarm Intelligence Symposium, pp. 68-75, 2005, disponível em http://www3.ntu.edu.sg/home/epnsugan/ em Fevereiro de 2009.

[Liu e Abraham 05] Liu, H., Abraham, A., *Fuzzy Adaptive Turbulent Particle Swarm Optimization*, Fifth International Conference on Hybrid Intelligent Systems (HIS'05), 2005.

[Lourakis 05] Lourakis, M.I.A, *A Brief Description of the Levenberg-Marquardt Algorithm Implemened by levmar*, Foundation for Research and Technology - Hellas (FORTH), Grécia, 2005.

[Noman e Iba 03] Noman, N., Iba, H, *Enhancing Differential Evolution Performance with Local Search for High Dimensional Function Optimization*, Proceedings of the 2005 conference on Genetic and evolutionary computation (GECCO '05), pp. 967-974, 2005.

[Onwubolu e Davendra 09] Onwubolu, G.C., Davendra D., *Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization*, Studies in Computational Intelligence, **175**, Springer, 2009.

[Papadrakakis e Lagaros 03] Papadrakakis, M., Lagaros, N.K.,*Soft Computing Methodologies for Structural Optimization*, Applied Soft Computing, **3**, pp. 283-300,2003.

[Perez e Behdinan 07] Perez, R.E., Behdinan, K., *Particle Swarm Optimization in Structural Design*, disponível em http://intechweb.org/book.php?id=17&PHPSESSID=gb8tibrkk20omfpmqhslfuii73, consultado em Abril de 2009.

[Price e Storn 09] Price, K., Storn, R., http://www.icsi.berkeley.edu/ storn/code.html, consultado em Fevereiro de 2009.

[Price *et al.* 05] Price, K.V., Storn, M.R., Lampinem J.A., *Differential Evolution - A Pratical Approach to Global Optimization*, Springer, 2005.

[Ravindran *et al.* 06] Ravindran, A., Ragsdell, K.M., Reklaitis, G.V., *Engineering Optimization - Methods and Applications*, 2^{nd} Ed., John Wiley & Sons, Inc., New Jersey, 2006.

[Shi e Eberhart 98] Shi, Y., Eberhart, R.C., *A Modified Particle Swarm Optimizer*, IEEE World Congress on Computational Intelligence Evolutionary Computation Proceedings, pp. 69-73, Anchorage, AK, USA, 1998.

[Shi e Eberhart 98-2] Shi, Y., Eberhart, R.C., *Empirical Studies of Particle Swarm Optimization*, Evolutionary Programming VII: Proceedings of EP 98, pp. 591-600, 1998.

[Singiresu 96] Singiresu, S.R., *Engineering Optimization - Theory and Practice*, 3^{rd} Ed., John Wiley & Sons, Inc., 1996.

[Sivanandam e Deepa 08] Sivanandam, S.N., Deepa, S.N., *Introduction to Genetic Algorithms*, Springer-Verlag Berlin Heidelberg, 2008.

[Smith e Griffiths 88] Smith, I.M., Griffiths, D.V., *Programming the Finite Element Method*, 2^{nd}*Ed*., John Wiley & Sons Ltd., 1988.

[Stacey *et al.* 03] Stacey, A., Jancic, M., Grundy, I., *Particle Swarm Optimization with Mutation*, Congress on Evolutionary Computation 2003 (CEC 2003), **3**, pp. 1425-1430, 2003.

[Storn e Price 95] Storn, R., Price, K., *Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces*, Technical Report TR-95-012, International Computer Science Institute, Berkeley, 1995.

[Storn e Price 97] Storn, R., Price, K. *Differential Evolution - a Simple and Efficient Heuristic for Global Optimization Over Continuous Spaces*, Journal of Global Optimization, **11**, pp. 341-59, 1997.

[Suli e Mayers 03] Suli, E., Mayers, D., *An Introduction to Numerical Analysis*, Cambridge University Press, 2003.

[Sunganthan 99] Suganthan, P.N., *Particle Swarm Optimiser With Neighbourhood Operator*, Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1999), Piscataway, NJ, pp. 1958-1962, 1999.

[Tang *et al.* 08] Tang, H., Xue, S., Fan, C., *Differential Evolution Strategy for Structural System Identification*, Computers & Structures, **86**, pp. 2004-2012, 2008.

[Trelea 03] Trelea, I.C., *The Particle Swarm Optimization Algorithm: Convergence Analysis and Parameter Selection*, Information Processing Letters, **85**, pp. 317-325, 2003.

[van der Berg 01] van der Berg, F., *An Analysis of Particle Swarm Optimizers*, Tese de Doutoramento, University of Pretoria, Pretoria, 2001.

[Vcerumachancni *et al.* 03] Vcerumachancni, K., Pcram, T., Mohan, C.K., Osadciw, L.A., *Optimization Using Particlc Swarms With Near Neighbor Interactions*, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003), Chicago, Illinois, USA, pp.110-121, 2003.

[Wu *et al.* 07] Wu, C.H., Tseng, K.Y., Lin, W.C., *Topology Optimization of Structure Using Differential Evolution*, The 11th World Multi-Conference on Systemics, Cybernetics and Informatics, Orlando, Florida, USA, 2007.

[Xi *et al.* 08] Xi, M., Sun, J., Xu, W., *An Improved Quantum-Behaved Particle Swarm Optimization Algorithm With Weighted Mean Best Position*, Applied Mathematics and Computation, **205**, pp. 751-759, 2008.

[Yang *et al.* 07] Yang, X., Yuan, J., Yuan, J., Mao, H.,*A Modified Particle Swarm Optimizer With Dynamic Adaptation*, Applied Mathematics and Computation ISSN 0096-3003, **189**, pp. 1205-1213, 2007.

[Yang *et al.* 08-1] Yang, Z., He, J., Yao, X., *Making a Difference to Differential Evolution*, Advances in Metaheuristics for Hard Optimization, Z. Michalewicz and P. Siarry, Eds. Springer, pp. 397-414, 2008.

[Yang *et al.* 08-2] Yang, Z., Tang, K., Yao, X., *Self-Adaptive Differential Evolution With Neighborhood Search*, IEEE Congress on Evolutionary Computation, pp. 1110-1116, 2008.

[Zahari 02] Zahari, D., *Critical Values for the Control Parameters of Differential Evolution Algorithms*, MENDEL 2002, 8th International Conference on Soft Computing (Matousek R. and Osmera P. eds). University of Technology, Brno 62-67, 2002.

[Zheng *et al.* 08] Zheng, X., Lin, G., Wang, J., Zhang, Y., *Parameter Identification of Nonlinear Viscoelastic-Plastic Constitutive Equation of Soybean and Cottonseed Based on Particles Swarm Optimization*, Proceedings of the First International Workshop on Knowledge Discovery and Data Mining, pp. 546-552, 2008.

[Zheng *et al.* 03] Zheng, Y., Ma, L., Zhang, L., Qian, J., *Empirical Study of Particle Swarm Optimizer With an Increasing Inertia Weight*, Proceedings of IEEE Congress on Evolutionary Computation 2003 (CEC 2003), Canbella, Australia. pp. 221-226, 2003.

[Zhenyu *et al.* 06] Zhenyu, G., Cheng, B., Min, Y., Binggang, C., *Self-Adaptive Chaos Differential Evolution*, Lecture notes in computer science, **4221**, pp. 972-975, 2006.