VASCO MIGUEL DA FONSECA RAMOS

Licenciado em Ciências de
Engenharia Eletrotécnica e de Computadores

DECOPT - SISTEMA PARA AUXÍLIO NO PROJETO DE SISTEMAS

Dissertação para obtenção do Grau de Mestre em
Engenharia Eletrotécnica e de Computadores

Orientador: Fernando José Almeida Vieira do Coito,
Professor Doutor, FCT-UNL

Júri:

Presidente: Prof. Doutor Luís Filipe Figueira de Brito Palma
Vogais: Prof. Doutora Maria Helena Silva Fino
Prof. Doutor Fernando José Almeida Vieira do Coito

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Abril 2013

Copyrigth

DECOPT - SISTEMA PARA AUXÍLIO NO PROJETO DE SISTEMAS

Dedicatória e Agradecimentos

Dedico esta dissertação aos meus pais que me apoiaram e amaram sempre. Muito obrigado por me terem proporcionado a possibilidade de terminar os meus estudos.

Agradeço também à minha irmã e aos meus avós por me motivarem e apoiarem incondicionalmente ao longo de toda a minha vida.

Gostaria de agradecer ao meu orientador Professor Doutor Fernando José Almeida Vieira do Coito pela sua colaboração e auxílio prestado. Estou também grato pela sua completa disponibilidade no desenvolvimento da minha dissertação, bem como pela partilha da sua experiência e conhecimento.

A todos os meus colegas de curso, um grande obrigado por percorrerem este caminho a meu lado ao longo destes últimos 6 anos e pela partilha de momentos únicos na vossa companhia.

Um agradecimento especial a todos os meus amigos pelas suas amizades, apoios e disponibilidades.

Finalmente, quero agradecer a uma pessoa muito especial, Ana Sofia Fidalgo Dias, por iluminar e enriquecer todos os dias a minha vida. Muito Obrigado.

Resumo

Este trabalho tem como objetivo o desenvolvimento de uma aplicação que auxilie no projeto de sistemas. Projetar um sistema consiste em procurar o melhor comportamento possível, tomando em consideração certas especificações inicialmente propostas. Ou seja, maximizar o desempenho do sistema, minimizar o consumo de recursos e cumprir diversas restrições ao seu funcionamento. Para se atingir estes objetivos no presente trabalho, recorre-se a técnicas de otimização, mais concretamente, a algoritmos de procura através de enxames de partículas (*PSO*). Começa-se por usar o algoritmo *PSO* original, pois é o algoritmo básico, a partir do qual derivam todos os outros algoritmos desta classe. O algoritmo *MOPSO* desenvolvido é uma variação do algoritmo *PSO* original, de modo a caracterizar os possíveis compromissos associados a situações onde existe mais do que um objetivo a otimizar. Nesta situação não existe uma solução ótima, mas sim um conjunto de soluções ótimas. Finalmente, escolheu-se o algoritmo *PSO* com a utilização da topologia em anel. Este algoritmo permite que haja mais do que um agrupamento de partículas, promovendo a criação de sub-enxames, de forma a encontrar, não só a solução ótima global, mas também soluções localmente ótimas. A combinação deste algoritmo em conjunto com a técnica *Maxmin* permite a resolução de problemas de otimização multi-objetivo. A análise dos resultados experimentais obtidos mostra que, com os devidos parâmetros introduzidos, o algoritmo *PSO* original pode atingir uma convergência de 100%. Mostra também que o algoritmo *MOPSO* produz um conjunto de soluções de compromisso, sendo uma boa aproximação da *Fronteira de Pareto* real do problema. Finalmente, o algoritmo *PSO* em anel é utilizado de duas formas: pode ser utilizado como forma de melhorar a convergência na otimização de um único objetivo ou, em projetos em que coexistem múltiplos objetivos, é usado para melhorar o detalhe com que a *Fronteira de Pareto* é determinada.

palavra-chave: otimização, algoritmo de otimização através de partículas (*PSO*), algoritmo multi-objetivo de otimização através de enxame de partículas (*MOPSO*), *PSO* em anel.

Abstract:

This work aims to create an application that is able to assist in the design of systems. Designing a system is to seek for the best possible performance, taking into consideration certain specifications initially proposed. In other words, maximize system performance, minimize resource consumption and obey several restrictions on its performance. To achieve these objectives in the current work, we resort to optimization techniques, more specifically, the particle swarm optimization algorithm (*PSO*). First we use the standard *PSO* algorithm, since it is the basic algorithm, from which all other algorithms branch. The *MOPSO* algorithm developed is a variation of the standard *PSO* algorithm, in order to feature the possible commitments of the conditions combined with more than one objective to optimize. In this situation, the algorithm doesn't return an optimal solution, but rather a set of optimal solutions. Finally, the *PSO* algorithm using a *Ring* Topology was chosen. This algorithm allows more than one set of particles to exist, furthe*Ring* the creation of sub-swarms, so that it can find, not only the global optima solution, but also the local optima solutions. This algorithm was fused with the *Maxmin* technique, so it could optimize multi-objective problems. Analyzing the experimental results shows that, with the proper parameters, the standard *PSO* may get a particle convergence on the optimal solution of 100%. It also shows that the *MOPSO* algorithm produces an approach set of commitment solutions to the real Pareto Frontier. Finally, the *PSO* algorithm using a *Ring* Topology is used in two different ways: it can be used to improve the convergence in a single-objective optimization or it can calculate the Pareto Frontier in multi-objective optimization.

keyword: optimization, particle swarm optimization (*PSO*), multi-objective particle swarm optimization (*MOPSO*), *PSO* using a *Ring* Topology.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Símbolos e de Acrónimos

PSO	Otimização baseada em enxame de partículas
MOPSO	Otimização multi-objetivo baseada em enxame de partículas
PSO Ring	Otimização baseada em enxame de partículas recorrendo à topologia em anel
x^*	Ótimo global
x_L^*	Ótimo local
V_i^{k+1}	Velocidade da partícula i na iteração k+1
V_i^k	Velocidade da partícula i na iteração k
ω	Constante de inércia da partícula
P_i^k	Posição do ótimo local da partícula i na iteração k
P_g^k	Posição do ótimo global do enxame na iteração k
X_i^k	Posição da partícula i na iteração k
X_i^{k+1}	Posição da partícula i na iteração k+1
c_1	Constante de peso do ótimo local na velocidade
c_2	Constante de peso do ótimo global na velocidade
r_1	Valor aleatório de distribuição uniforme
r_2	Valor aleatório de distribuição uniforme
f_k^i	Atributo k de uma solução i
f_k^j	Atributo k de uma solução j

Glossário

Otimização é o processo de maximizar/minimizar uma função objetivo de forma a encontrar a solução ótima.

Função objetivo é um modelo matemático que represente de forma quantificada o grau de desempenho alcançado no cumprimento do objetivo do projeto.

Solução ótima é uma solução que resulte no melhor desempenho possível do sistema, dentro da região de procura.

Região de procura é a região delimitada pelas restrições feitas à função objetivo e às variáveis de projeto.

Variáveis de projeto são as grandezas usadas para descrever o sistema. O resultado do projeto é a seleção de uma combinação de valores destas grandezas que satisfaz o objetivo de projeto. Definem a posição na região de procura.

Problema unimodal é um problema de otimização onde apenas existe uma solução ótima.

Problema multimodal é um problema de otimização onde existem várias soluções ótimas.

Problema multi-objetivo é um problema de otimização onde se pretende otimizar vários objetivos em simultâneo.

Custos é o fator de qualidade quantificado inerente a cada posição da região de procura na minimização.

Partícula é um individuo computacional com memória e velocidade próprias.

Enxame é um conjunto de partículas.

Solução dominada é uma solução de um problema multi-objetivo que apresenta um pior compromisso que outra solução.

Solução não dominada é ume solução de um problema multi-objetivo que apresenta um compromisso ótimo.

Ótimo global é o ótimo absoluto do problema.

Ótimo local é um ótimo relativo do problema.

Fronteira de Pareto é o conjunto de soluções não dominadas.

Topologia em anel é uma topologia que cria mini enxames dentro do enxame principal, agrupando as partículas em conjuntos de igual número.

Niching é uma técnica utilizada pela otimização de forma a encontrar as várias soluções ótimas de um problema multimodal.

Maxmin é uma técnica utilizada na minimização multi-objetivo para encontrar a *Fronteira de Pareto*. Encontra as partículas não dominadas.

Stretching é uma técnica utilizada na minimização. A função objetivo é transformada de modo a eliminar os mínimos locais e a preservar os mínimos globais.

Subpopulation é uma técnica utilizada na otimização, onde o enxame principal é dividido em sub-enxames, cada um com o seu próprio ótimo do sub-enxame.

Breeding é uma técnica utilizada na otimização que permite a criação e eliminação de uma partícula.

Crowding distance é uma técnica utilizada na otimização para encontrar a *Fronteira de Pareto*. Esta técnica ordena as soluções por ordem crescente de valor da distância entre si. De seguida calcula a distância média entre a partícula anterior e posterior. Caso alguma solução esteja entre esta distância, é considerada dominada.

ClusteRing é uma técnica utilizada na otimização para encontrar a *Fronteira de Pareto*. Esta técnica agrupa as soluções que apresentam o mesmo comportamento, escolhendo apenas uma solução para representar esse grupo.

1. Introdução

Um projeto de Engenharia consiste num conjunto de atividades, que incluem analisar, projetar, construir, pesquisar, desenvolver e avaliar o desempenho de sistemas. Num projeto pretende-se obter o melhor resultado possível, tendo em vista os objetivos inicialmente propostos, e as limitações existentes. Uma dessas limitações é o tempo disponível para o desenvolvimento do próprio projeto.

Otimização consiste em maximizar o desempenho de um sistema, minimizar os custos dos recursos disponíveis e cumprir diversas restrições ao seu funcionamento. O resultado final deste processo é uma solução ótima, isto é, uma solução que resulte no melhor desempenho possível do sistema.

Os problemas de otimização podem ser alocados em dois grupos diferentes, os problemas unimodais e os problemas multimodais. Os problemas unimodais têm apenas uma solução ótima, considerado um ótimo global, x^*, sujeito a $f(x^*) \leq f(x) \forall\, x \in \mathbb{R}^n$, assumindo a minimização, onde $f(x)$: $\mathbb{R}^n \longrightarrow \mathbb{R}$ representa a função objetivo e n a dimensão da região de procura. Os problemas multimodais, por outro lado, têm mais do que uma solução ótima. Estas soluções podem ser todas soluções ótimas globais, ou uma mistura de soluções ótimas globais e locais. Uma solução ótima local, x_L^*, está sujeita a $f(x_L^*) \leq f(x) \forall\, x \in L$, assumindo novamente a minimização, onde $L \subset \mathbb{R}^n$.

O desenvolvimento de algoritmos computacionais para otimização tem sido alvo de extensas pesquisas, tendo originado diversas técnicas de aprendizagem tais como algoritmos evolucionários, caso de Jeffrey Horn [12], assim como algoritmos baseados no comportamento social dos animais, casos de Marco Dorigo, Mauro Birattari e Thomas Stützle [15].

Neste trabalho desenvolve-se uma aplicação que utiliza um algoritmo baseado no comportamento social dos animais, a otimização baseada em enxame de partículas (*PSO*), bem como dois métodos baseados em versões modificadas do algoritmo *PSO*, casos da otimização multi-objetivo baseada em enxame de partículas (*MOPSO*) e da otimização baseada em enxame de partículas recorrendo à topologia em anel (*PSO Ring*), para resolver problemas de otimização.

1.1. Motivação

Esta aplicação permite solucionar a maioria dos problemas de otimização. Desta forma, poupa bastante tempo de trabalho, ao engenheiro, na otimização manual dos problemas. Existem vários algoritmos de otimização. A razão das escolhas dos algoritmos estudados e implementados neste trabalho serão inumerados de seguida.

O algoritmo *PSO* é semelhante a outros algoritmos evolucionários, uma vez que apresenta um grupo de candidatos à solução final. Porém consegue resolver uma grande variedade de problemas de otimização, com uma taxa de convergência mais rápida. Para além disso, possui outra vantagem, requer apenas alguns parâmetros a serem introduzidos, tornando-o atrativo do ponto de vista da implementação. Estas foram as razões que levaram ao estudo deste algoritmo.

No mundo real, os engenheiros deparam-se com problemas que têm diversos objetivos (multi-objetivos). A utilização do algoritmo *MOPSO* permite a obtenção de um conjunto de soluções ótimas, isto é, a *Fronteira de Pareto*. Através da *Fronteira de Pareto*, os engenheiros podem escolher a solução que melhor se adequa aos seus objetivos. Graças à *Fronteira de Pareto*, existe uma diversidade de escolha, que vai colmatar qualquer necessidade apresentada. Outra vantagem deste algoritmo é a sua rápida convergência, visto ser baseado no algoritmo *PSO*. Para além disso, o algoritmo *MOPSO* é um dos algoritmos multi-objetivos mais rápidos no processamento computacional, o que nos dias de hoje representa uma grande vantagem.

Os algoritmos *PSO* que recorrem ao método de *Niching* são bastante úteis na resolução de problemas multimodais, pois para além de apresentarem os ótimos globais, também apresentam alguns ótimos locais que correspondem a uma solução considerada ótima. Porém estes algoritmos necessitam de informação previamente fornecida sobre o problema em causa, o que condiciona o seu uso para problemas existentes no Mundo real. É neste contexto que surge o algoritmo *PSO* com o uso da topologia em anel.

Para além do algoritmo *PSO* com uso à topologia em anel resolver grande parte das desvantagens do método *Niching*, também apresenta resultados superiores e é mais consistente.

Este algoritmo vai-se socorrer à técnica de *Maxmin* para poder resolver problemas de otimização multi-objetivos, encontrando a *Fronteira de Pareto*. Existem alguns algoritmos *Niching* que resolvem problemas de otimização multi-objetivos, porém um algoritmo que combine o *PSO* com a topologia em anel e a técnica *Maxmin* é inovador. A principal vantagem deste algoritmo na construção da *Fronteira de Pareto* em ralação ao *MOPSO* é o fato de ter uma convergência mais lenta, o que permite encontrar não só os ótimos globais, como também os

ótimos locais. Caso algum ótimo local não seja muito dominado pelos ótimos globais, apresenta uma solução viável que seria descartada à partida pelo algoritmo *MOPSO*. Para além deste aspeto, uma solução que se encontra próxima de outra solução no espaço de critérios, poderá estar bastante longe da mesma no espaço de projeto, realçando a importância dos ótimos locais que não são muito dominados. Desta forma, esta nova abordagem de otimização de problemas multi-objetivos apresenta uma maior liberdade de escolha e um maior espaço de manobra.

1.2. Contribuições

Toda esta aplicação foi construída em *Matlab* Os algoritmos implementados foram testados com várias funções objetivo, de forma a registar o comportamento de cada um deles. O *PSO* atingiu melhores resultados com uma elevada população de partículas e um elevado número de iterações, com uma convergência de 100 % em todas as funções teste. O *MOPSO* mostrou ser capaz de produzir uma *Fronteira de Pareto* bastante fiável à original. O *PSO Ring* pode ser utilizado com diferentes objetivos. Caso se pretenda encontrar apenas a solução ótima, deve-se criar subpopulações constituídas por cinco partículas, a população deve ser constituída por um elevado número de partículas e devem ser atualizadas por um elevado número de iterações. Desta forma, este algoritmo atingiu uma convergência de 100% em todas as funções teste. Caso se pretenda encontrar o conjunto de soluções consideradas ótimas, deve-se criar subpopulações constituídas por três partículas, a população deve ser constituída por um elevado número de partículas e devem ser atualizadas por um pequeno número de iterações. Desta forma, este algoritmo conseguiu encontrar mais do que uma solução ótima em cada função teste, porém não conseguiu encontrar todas as soluções ótimas. Finalmente, o algoritmo *PSO Ring* utilizando a técnica *Maxmin* mostrou ser capaz de produzir uma *Fronteira de Pareto* bastante similar à produzida pelo algoritmo *MOPSO*, com a vantagem de incorporar as soluções dominadas, o que fornece ao utilizador um maior grau de liberdade na implementação da solução final.

1.3. Composição da Dissertação

Esta dissertação tem a seguinte organização:

Capítulo 2: Breve explicação do funcionamento dos algoritmos escolhidos, bem como se procede à sua implementação e o motivo inerente às escolhas realizadas;

Capítulo 3: Descrição do trabalho realizado, tanto na implementação dos algoritmos, como na interatividade gráfica produzida;

Capítulo 4: Apresenta os resultados obtidos na análise do comportamento dos algoritmos implementados;

Capítulo 5: Conclusões retiradas a partir dos resultados obtidos e enumeração de oportunidades futuras de pesquisas na sequência desta linha de trabalho.

2. Técnicas para projeto ótimo baseado em *PSO*

2.1. Introdução

2.1.1. O Processo de Otimização

Um projeto de engenharia pode ser um processo bastante complexo. Frequentemente é necessário desenvolver modelos matemáticos que possam ser analisados e sujeitos a experiências. Durante a fase de formulação do problema têm de ser considerados inúmeros fatores e possibilidades. Na maior parte das vezes, o projeto tem de ser dividido em vários sub-problemas que, por sua vez, são tratados como um problema individual. Cada sub-problema pode ser associado a um problema de otimização.

Projetar é um processo iterativo. A experiencia, a intuição e o talento do projetista são necessários no projeto de sistemas nos diversos ramos da engenharia. É um processo iterativo, uma vez que projetar implica a análise de vários projetos experimentais até se encontrar um que seja aceitável. Neste processo, o projetista avalia uma versão experimental do sistema baseando-se na sua experiência, intuição e em análises matemáticas. O projeto experimental é analisado de forma a determinar se é aceitável. Se for, o processo de projetar chega ao fim. No processo de otimização, o projeto experimental é analisado para determinar se é o melhor. A palavra "melhor" pode ter diferentes significados, consoante o sistema a ser tratado. Geralmente, implica rentabilidade, eficiência, segurança e duração do sistema.

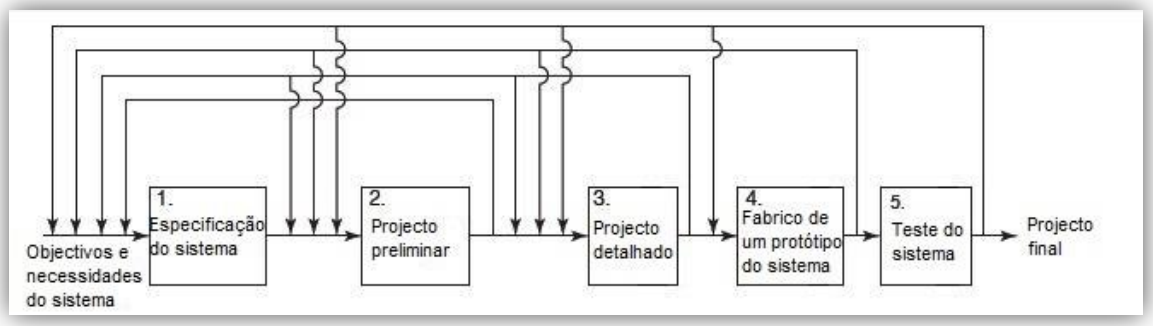


Figura 2.1- Modelo da evolução de um sistema.

O processo de projetar deve ser uma atividade bem organizada, tal como mostra a Figura 2.1. Primeiro é necessário definir especificações precisas para o sistema. Normalmente, da interação entre o engenheiro e o responsável do projeto, resulta uma versão quantificada das especificações do sistema. Assim que este processo acabar, pode-se começar a projetar o sistema.

No segundo passo deste processo é preciso produzir um projeto preliminar do sistema. São estudados vários conceitos para o sistema. Uma vez que o tempo é precioso, são usados modelos simplificados. São identificados vários subsistemas e, consequentemente, são estimados os seus projetos preliminares. As decisões tomadas nesta altura geralmente afetam o desempenho final do sistema. No final desta fase de projeto, são identificados alguns conceitos que precisam de ser posteriormente analisados.

No terceiro passo deste processo é produzido um projeto detalhado para todos os subsistemas, utilizando um processo iterativo. Este processo deve ser feito a todos os conceitos previamente identificados, de modo a avaliar todas as possibilidades. Os parâmetros de projeto dos subsistemas têm de ser identificados. Os requisitos do desempenho do sistema têm de ser identificados e cumpridos. Os subsistemas têm de ser projetados de forma a maximizar o valor do sistema ou de forma a minimizar os custos. No final do processo, está disponível uma descrição do sistema.

Frequentemente, pode-se dar como concluído o processo destas duas fases e passar-se para a fase de implementação. No entanto, em algumas situações (produção em grandes séries, segurança de pessoal, etc.) o projeto inclui também a fabricação de um protótipo do sistema e o seu teste.

O processo de projetar calcula as dimensões e formas de diferentes partes do sistema de forma a ir ao encontro do desempenho requisitado. O projeto de um sistema é um procedimento de tentativa e erro. Estima-se uma solução que de seguida é analisada de forma a verificar se o seu desempenho corresponde às especificações. Se corresponder, tem-se um projeto possível de realizar, embora ainda o possa alterar de forma a melhorar o seu desempenho.

Todo este processo de projetar mencionado baseia-se, como já foi referido, na experiência, intuição e habilidades do projetista. Este processo depende da presença humana, o que por vezes conduz a falsos resultados na síntese do sistema complexo.

A escassez de recursos e a necessidade de eficiência no mundo competitivo de hoje obrigaram os engenheiros a evidenciar um maior interesse em projetos mais económicos. O processo *CADO* (*computer-aided design optimization*) pode ajudar no que a isto diz respeito. A principal vantagem no processo de projetar convencional é que a experiência e intuição do

projetista podem ser utilizadas de forma a fazer alterações no sistema ou fazer especificações adicionais no procedimento.

No que diz respeito a projetos detalhados, o processo de projetar convencional tem algumas desvantagens, incluindo o tratamento de restrições complexas e o processo de projetar convencional poder levar a um projeto dispendioso. O processo de otimização do projeto obriga o projetista a identificar especificamente um conjunto de variáveis de projeto, uma função objetivo a ser otimizada e a função de restrições do sistema. Esta formulação rigorosa do problema de projeto ajuda o projetista a ganhar uma maior compreensão do sistema.

2.1.2. Formulação de um Problema de Otimização

A elaboração de um problema de otimização envolve transformar o problema real numa formulação matemática, de forma a poder ser tratado. O primeiro passo é identificar o problema e desenvolver uma descrição que o traduza. Este processo é normalmente efetuado pelo responsável do projeto. A descrição deve conter os objetivos e os requisitos do problema.

O segundo passo é desenvolver uma equação matemática que traduza a descrição feita do problema. Para tal, é necessário ter em consideração as propriedades do material, o desempenho pretendido, a limitação dos recursos, os custos do material, entre outros aspetos. Além disso, nesta fase, o procedimento e as ferramentas de análise são também identificados. Alguma informação do projeto e expressões dependem de variáveis de projeto que são identificadas no próximo passo. Portanto, tal informação vai ser especificada mais à frente do processo de formulação.

O próximo passo no processo de formulação é identificar um conjunto de variáveis que descrevam o sistema, denominadas de variáveis de projeto. Normalmente, são também referidas como variáveis de otimização. Diferentes valores das variáveis produzem diferentes projetos. As variáveis de projeto devem ser independentes umas das outras o máximo possível. Caso sejam dependentes, então os seus valores não podem ser especificados de forma independente. O número de variáveis de projeto independentes traduz o número de graus de liberdade do projeto do problema.

Em alguns problemas, diferentes conjuntos de variáveis podem descrever o mesmo sistema. A formulação do problema vai depender do conjunto escolhido. Assim que são atribuídos valores numéricos às variáveis de projeto, obtém-se o projeto do sistema. Se este projeto satisfaz todos os requisitos é uma questão à parte que vai ser respondida mais adiante

Caso não sejam escolhidas variáveis de projeto adequadas para o problema, a formulação ou foi mal executada, ou então é impossível fazer uma. Na fase inicial da formulação do problema, todas as variáveis de projeto identificadas devem ser investigadas. Por vezes é aconselhável designar um maior número de variáveis de projeto que graus de liberdade de projeto. Desta forma, obtém-se uma maior flexibilidade na formulação do problema.

No quarto passo, todos os projetos realizáveis do sistema são comparados segundo um critério, de forma a encontrar o melhor projeto. O critério precisa de ser uma função escalar cujo valor numérico possa ser obtido assim que o projeto seja especificado, isto é, a função tem de depender das variáveis de projeto. Normalmente, esse critério é denominado de função objetivo nos problemas de otimização. A função objetivo é maximizada ou minimizada dependendo do objetivo do problema em questão. O critério que se pretende minimizar é denominado por função custo na engenharia. Uma função objetivo válida tem de ser diretamente ou indiretamente influenciada pelas variáveis de projeto, caso contrário não é uma função objetivo significativa. Um projeto ótimo tem o melhor valor para a função objetivo.

A seleção da correta função objetivo é uma decisão bastante importante no processo de projeto. Em certas ocasiões é fácil identificar a função objetivo, por exemplo, quer-se sempre minimizar os custos da manufatura ou maximizar o retorno de um investimento. Em algumas situações, pode-se identificar mais que uma função objetivo. Por exemplo, numa viagem de carro pode-se querer maximizar a rapidez do automóvel de forma a se chegar mais depressa ao destino, e ao mesmo tempo pode-se querer minimizar o consumo do carro. Estes problemas são chamados problemas de otimização multi-objetivo.

O quinto passo, e último, é identificar todas as restrições (limitações existentes no projeto) e desenvolver expressões para as mesmas. A maior parte dos sistemas, precisam de ser projetados e fabricados dentro do limite de recursos e de encontro ao desempenho pretendido. As restrições devem depender das variáveis de projeto, uma vez que apenas desta forma o seu valor é alterado quando se altera o projeto experimental.

2.1.3. Síntese da Otimização

Em qualquer otimização é necessário existir uma função objetivo que represente matematicamente o problema que se quer otimizar. A tarefa da otimização é, neste trabalho, minimizar uma função objetivo. Para além de minimizar, a otimização também pode ser encarada como uma maximização. Porém, qualquer função objetivo que se pretenda maximizar f, pode ser transformada numa função objetivo que se queira minimizar $-f$. A função objetivo f

pode ter apenas uma variável, $f(x)$, ou pode ter n variáveis independentes $x_1, x_2, \dots, x_n$, $f(x) = f(x_1, x_2, \dots, x_n)$.

O objetivo da minimização é encontrar um conjunto de soluções válidas que represente o melhor desempenho da função objetivo. Uma solução válida é uma solução que esteja situada dentro do espaço de projeto. A solução válida que representar o melhor desempenho é denominada de solução ótima. As variáveis de projeto pertencem a um espaço de projeto n dimensional, $x \in \mathbb{R}^n$, onde cada posição no espaço de projeto é representado por $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$.

O espaço de projeto, ou região de procura, é delimitado pelas restrições efetuadas ao sistema, uma vez que as restrições dependem das variáveis de projeto. As restrições podem ser efetuadas de duas maneiras distintas, ou as restrições são verificadas individualmente, ou então formam uma função matemática onde cada restrição vai possuir um peso dentro desta mesma função. Os pesos atribuídos a cada restrição vão determinar a importância e a influência que cada uma delas tem no desempenho final do sistema.

A solução ótima encontrada pode ser considerada um ótimo global ou um ótimo local. Um ótimo global x^* está sujeito a $f(x^*) \leq f(x) \forall\, x \in \mathbb{R}^n$, assumindo a minimização. Por outro lado, um ótimo local x_L^* está sujeito a $f(x_L^*) \leq f(x) \forall\, x \in L$, assumindo novamente a minimização, onde $L \subset \mathbb{R}^n$, em que L é uma vizinhança do x^*.

De forma a encontrar o mínimo global de uma função objetivo contínua, não sujeita a restrições, basta calcular o seu gradiente. Porém, caso a função objetivo não seja contínua, o cálculo do seu gradiente torna-se impossível. É necessário, portanto, utilizar outras formas de proceder à minimização. Para minimizar uma função objetivo descontínua é preciso recorrer aos custos inerentes a cada posição no espaço de projeto da função objetivo. Os custos traduzem os parâmetros, ou variáveis de projeto, da função objetivo num fator de qualidade quantificado.

2.1.4. Técnicas computacionais para otimização

Nesta secção vão ser descritas algumas das técnicas mais utilizadas na literatura para otimização. Vão ser apresentadas as suas limitações de cada técnica e a razão da escolha da técnica final a ser implementada.

2.1.4.1. Algoritmos Genéricos

A fundamentação dos Algoritmos Genéticos é baseada na genética natural. Desta forma, é comum o uso de termos como indivíduos de uma população, cromossomos, genes e alelos. Nos Algoritmos Genéticos, a população de indivíduos é um conjunto de pontos do domínio da função a ser maximizada ou minimizada. A quantidade de pontos depende do número de variáveis de projeto do problema em questão.

Algoritmos genéticos são algoritmos iterativos, em que a cada iteração a população é modificada, usando as melhores características dos elementos da geração anterior e submetendo-as aos três tipos básicos de operadores, para produzir melhores resultados. Para atingir estes objetivos são usados os seguintes processos:

Reprodução: é um processo no qual cada cadeia é copiada tendo em conta os valores da função de adaptação f_c. A aptidão de cada indivíduo é um valor que representa o grau de adaptabilidade deste, ou seja, a distância que o indivíduo se encontra da solução do problema em comparação com o resto dos indivíduos da população. Esta probabilidade é medida com auxílio da função objetivo e pode ser dada pela seguinte expressão:

$$p_i = \frac{f_c(x)}{\sum f_c(x)}, \text{ sendo } \sum p_i = 1..$$

Cruzamento: é um processo no qual a combinação de cada par de cromossomos gera um novo descendente.

Mutação: é a modificação aleatória ocasional (com baixa probabilidade) do valor de um alelo da cadeia.

Em síntese, o procedimento deste algoritmo consiste em criar, aleatoriamente, uma população inicial de indivíduos {C1, C2,..., Cn}. Em seguida, todos os indivíduos dessa população são modificados, submetendo-os aos operadores genéticos; reprodução, cruzamento e mutação, de forma a encontrar a solução ótima.

Os Algoritmos Genéticos, assim como muitos algoritmos evolutivos de otimização, são desenvolvidos para resolver problemas que não possuem restrições. Assim, no caso de problemas com restrições, é necessário a utilização de outro algoritmo.

2.1.4.2. Método de Gradiente

O Método de Gradiente é um algoritmo de otimização de primeira ordem, onde a escolha da direção do mínimo da função f é a direção oposta ao gradiente:

$$\nabla f(x_i)$$

O processo de busca começa num ponto arbitrário x_0 e caminha sobre a linha determinada pelo gradiente, até estar suficientemente próxima da solução. Isto é, o processo iterativo pode ser descrito como:

$$x_{k+1} = x_k - \alpha_k \nabla f(x_i)$$

A cada passo do processo iterativo, o valor de α pode ser alterado (aumentado) para permitir uma convergência mais rápida, certificando-se que a função converge com cada iteração. Caso contrário, o valor de α deve ser reduzido para que o processo se estabilize.

Porém, o Método de Gradiente apenas consegue tratar funções objetivo que sejam contínuas, o que limita bastante o leque de problemas que este algoritmo consegue resolver.

2.1.4.3. *Simulated anealing*

O *Simulated Annealing* (SA) é um método iterativo da busca inspirado pelo recozimento de metais. Começando com uma solução inicial, e equipado com funções adequadas de perturbação e avaliação, o algoritmo executa uma busca parcial estocástica do espaço de soluções. Os movimentos que pioram a avaliação da solução corrente são aceites ocasionalmente com uma probabilidade controlada por um parâmetro (T) chamado temperatura. A probabilidade da aceitação de movimentos que pioram a avaliação diminui à medida que T diminui. Com uma temperatura alta, a busca é quase aleatória, quando a temperatura é baixa a busca torna-se bastante seletiva. Quando a temperatura é igual a zero apenas os movimentos bons são aceitos.

Apesar dos sucessos do SA quando aplicado a uma grande variedade de problemas, o problema de partição numérica (*number partitioning problem*) é particularmente difícil para o SA.

2.1.4.4. *Particle Swarm Otimization*

O *Particle Swarm Otimization* (*PSO*) é um algotitmo baseado nos comportamentos sociais dos animais. Este algoritmo espalha inicialmente, de forma aleatória, um conjunto de partículas pela região de projeto. Estas partículas vão procurar a solução ótima. Todo este processo vai ser eximinado de forma mais detalhada no próximo capítulo. Este algortimo consegue resolver todas as limitações mencionadas pelos métodos anteriores.

2.2. *PSO*

2.2.1. Definição do *PSO*

Em 1995 Eberhart e Kennedy [16] introduziram a otimização baseada em enxame de partículas (*PSO*), um algoritmo baseado nos comportamentos sociais dos animais. Ou seja, este algoritmo investiga o espaço de projeto ajustando as trajetórias de vetores individuais, chamados de partículas, que são consideradas de pontos no espaço multidimensional. Cada partícula possui uma memória da melhor posição que encontrou (melhor pessoal). Também existe um mecanismo de comunicação, no enxame, que permite a cada partícula conhecer a melhor posição encontrada entre todas as partículas (melhor global). Estas posições armazenadas na memória vão influenciar o movimento de cada partícula em busca de uma posição melhor, uma vez que cada partícula movimenta-se em direção ao seu melhor pessoal e

ao melhor global do enxame encontrados até ao momento.Como o movimento de cada partícula depende da melhor posição encontrada pelo enxame (melhor global), como essa posição se altera à medida que o algoritmo evolui, este mecanismo origina uma forma de inteligência coletiva do enxame.

Todo o enxame tem conhecimento dos melhores pessoais de cada partícula, bem como do melhor global da população. Através desta comunicação, ao longo das iterações do algoritmo o enxame converge para uma solução melhor do que se cada partícula atuasse de forma isolada.

A população de partículas é inicializada com posições e velocidades aleatórias. A função objetivo é avaliada, utilizando as coordenadas posicionais de cada partícula como parâmetros.

A cada iteração que passa as partículas vão deslocar-se na direção do melhor global, alterando a posição e velocidade. Desta forma, a cada iteração, o enxame é atualizado através das seguintes equações:

$$V_i^{k+1} = \omega V_i^k + c_1 r_1 \left(P_i^k - X_i^k\right) + c_2 r_2 (P_g^k - X_i^k) \qquad \textbf{(1)}$$

$$X_i^{k+1} = X_i^k + V_i^{k+1} \qquad \textbf{(2)}$$

onde P_i é a posição com a melhor solução encontrada pela partícula i, ou seja o seu melhor pessoal, e P_g é a posição com a melhor solução encontrada entre todas as partículas do enxame, ou seja o melhor global. A equação (1) traduz a velocidade de uma partícula enquanto a equação (2) traduz a posição da mesma partícula.

A melhor posição encontrada pela partícula é atualizada de acordo com a expressão:

$$P_i^{k+1} = \begin{cases} P_i^k : f\left(X_i^{k+1}\right) \geq P_i^k \\ X_i^{k+1} : f\left(X_i^{k+1}\right) < P_i^k \end{cases} \qquad \textbf{(3)}$$

O ótimo global corresponde a:

$$P_g \in \{P_0, P_1, \dots, P_M\} \mid f\left(P_g\right) = \min(f(P_0), f(P_1), \dots, f(P_M)) \qquad \textbf{(4)}$$

onde f representa a função objetivo, M é o número total de partículas.

Na equação (1), r_1 e r_2 são distribuições uniformes entre 0 e 1. Os parâmetros c_1 e c_2 determinam a velocidade com que a partícula é atraída. ω, que toma valores entre [0,1], é o peso de inércia. Um maior peso de inércia facilita a exploração de áreas maiores, enquanto um menor peso de inércia é mais aconselhável para diferenciar pequenos detalhes. Desta forma, estes parâmetros são importantes para o desempenho do *PSO*.

Quando uma partícula encontra um exemplar que apresente um desempenho melhor que o seu atual melhor pessoal, guarda as suas coordenadas na memória. A diferença entre as coordenadas do melhor pessoal e da sua posição atual é adicionada à velocidade atual, obrigando um movimento oscilatório por parte da partícula em torno desse ponto. Para além

disso, também vai ser adicionado à velocidade a diferença entre as coordenadas do melhor global e da posição atual. Desta forma, estes ajustamentos na movimentação da partícula fazem com que a procura seja feita em torno destas duas melhores posições.

2.2.2. Evolução do algoritmo

Eberhart e Kennedy introduziram este algoritmo em 1995 [16]. Diversos exemplos de aplicação demonstram que o *PSO* é capaz de resolver problemas de otimização unimodais com sucesso. Como no artigo de Shi e Eberhart em 1999 [1], onde o algoritmo *PSO* foi aplicado a quatro funções objetivo diferentes. O algoritmo *PSO* mostrou convergir rapidamente sempre em direção da posição ótima. Porém mostrou ter uma convergência mais lenta quando se encontra perto de mínimos.

Até à data, foram propostas diversas alterações a este algoritmo. Em 2001, Løvbjerg, Rasmussen e Krink propuseram um algoritmo *PSO* modificado, combinando uma técnica de *breeding* com um esquema de divisão em subpopulações (*subpopulation*).De acordo com os autores a técnica de *breeding*, permite obter uma convergência mais rápida e encontrar uma solução melhor [2].

Também em 2001, Parsopoulos, Plagianakos, Magoulas e Vrahatis propuseram incluir no algoritmo *PSO* uma função *stretching*, de modo a diminuir o peso na solução final por parte dos máximos/mínimos locais. *PSO* obteve um bom desempenho em encontrar os máximos/mínimos globais [3].

Para além destas técnicas, muitas outras foram apresentadas de forma a melhorar o desempenho do algoritmo *PSO*. Algumas dessas técnicas tentam resolver problemas de otimização multimodais através do *PSO*, recorrendo a técnicas de *niching*. Em 2001, Parsopoulos e Vrahatis apresentaram uma modificação no *PSO*, que torna possível localizar múltiplos ótimos locais. Para tal, sempre que uma nova solução era localizada, o panorama da função objetivo é alterado [4].

Mais recentemente, em 2010, Xiaodong Li apresentou uma proposta bastante interessante na resolução de problemas otimização multimodais. Este algoritmo *PSO* recorre à topologia em anel de modo a colmatar as desvantagens da técnica de *Niching*. Os resultados obtidos demonstraram que este algoritmo apresentava soluções melhores e um desempenho mais consistente que as técnicas de *Niching* [6]. No capítulo 2.4 este algoritmo será tratado com mais detalhe.

Outras técnicas tentam adaptar o algoritmo *PSO* à resolução de problemas com vários objetivos. Em 2003, Balling propôs uma técnica de otimização multi-objetivo recorrendo à estratégia *Maxmin* [5]. Esta abordagem vai ser aprofundada mais à frente no capítulo 2.3.

Existem muitos outros algoritmos baseados no *PSO* que não foram mencionados aqui. Devido aos resultados apresentados em comparação a outros algoritmos de otimização [7, 8], o algoritmo *PSO* continua, na atualidade, a ser alvo de evolução.

2.2.3. Detalhes de Implementação

O algoritmo *PSO* é um algoritmo baseado em população, ou seja, é necessário inicializar as partículas de forma a ficarem posicionadas dentro da região de projeto. Após cada partícula ser corretamente inicializada, as partículas vão guardar as suas posições correspondentes na memória de cada uma como ótimos pessoais. O passo seguinte é definir quais dos ótimos pessoais é também o ótimo global, ou seja, quais dos ótimos pessoais apresenta um menor custo. Esta informação também é guardada na memória. Desta forma, fica completo o processo de inicialização. A Figura 2.2 sumariza todo este processo:

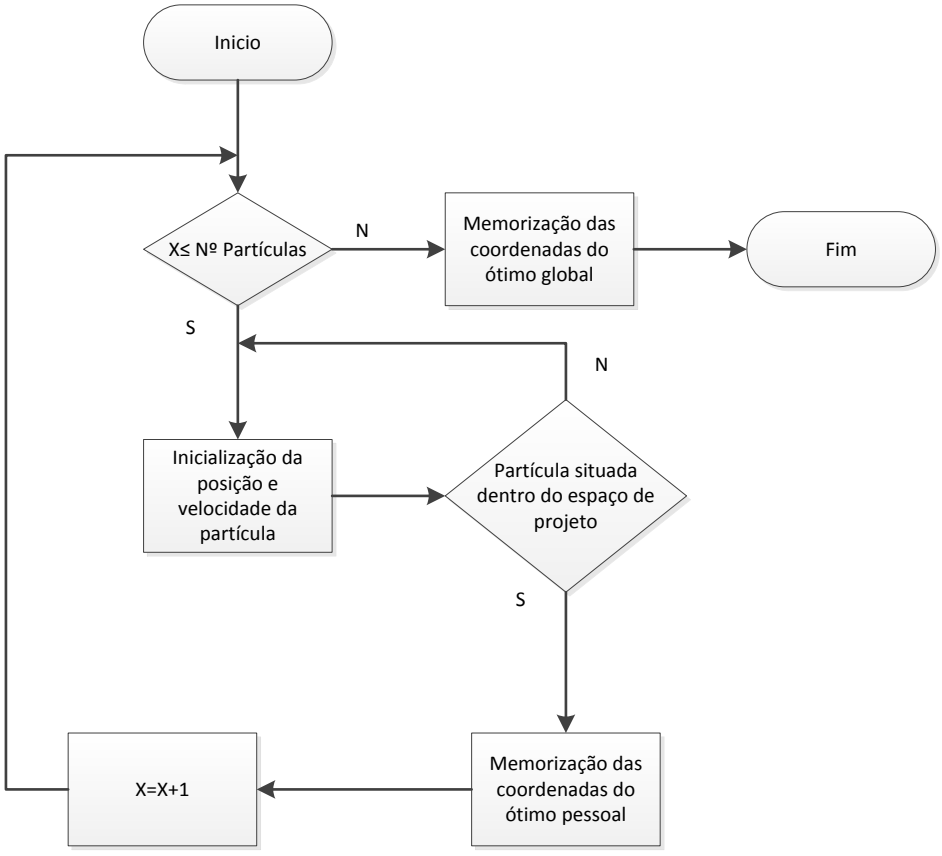


Figura 2.2- Processo de inicialização das partículas do algoritmo *PSO*.

De seguida, é necessário atualizar a posição, através da expressão (2), e velocidade, através da expressão (1), de cada partícula após as iterações. Esta fase só fica concluída quando a posição final de cada partícula, após a atualização, se mantiver dentro da região de projeto. É também necessário atualizar o ótimo pessoal de cada partícula após cada iteração. Se uma partícula chegou a uma posição, onde apresenta um menor custo em relação ao seu ótimo pessoal, essa posição passa a ser considerada o novo ótimo pessoal dessa partícula e é guardado na memória. Finalmente, o ótimo global vai sofrer atualizações após cada iteração ter chegado ao final. Se houver alguma partícula que atinge uma posição com menor custos que a posição guardada na memória como ótimo global, essa nova posição é considerada como o novo ótimo global e é guardada na memória. A Figura 2.3 representa todo este processo de iterações:

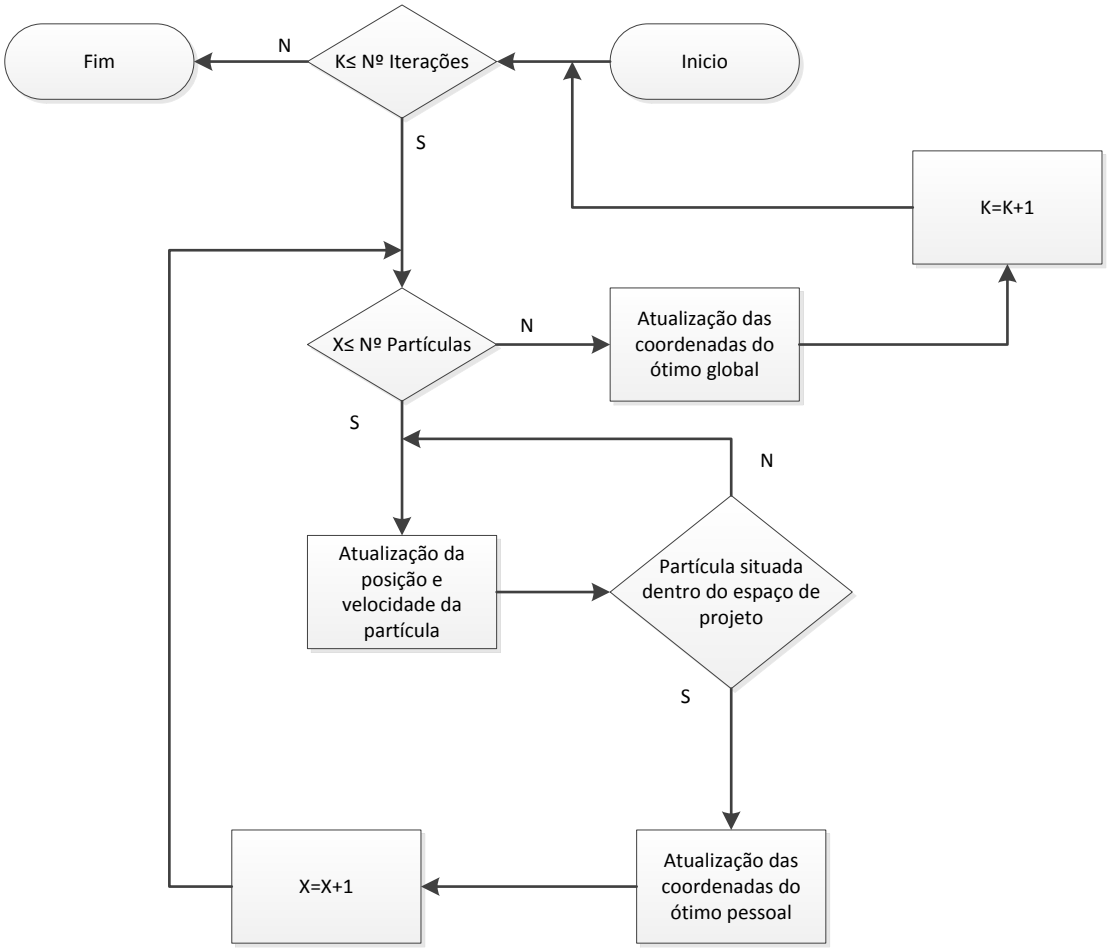


Figura 2.3- Processo inerente às iterações das partículas do algoritmo *PSO*.

No final de todo este processo, a posição do ótimo global, será a posição que representa menos custos, logo será a melhor posição encontrada por este algoritmo dentro da região de soluções admissíveis.

2.3. *MOPSO*

2.3.1. Problema de otimização multi-objetivo

Um problema de otimização multi-objetivo é definido por:

$$\boldsymbol{minimizar}\ \boldsymbol{f}(\boldsymbol{x}) = (\boldsymbol{f_1}(\boldsymbol{x}), \boldsymbol{f_2}(\boldsymbol{x}), \dots, \boldsymbol{f_k}(\boldsymbol{x}))$$

sujeito a

$$\boldsymbol{h_i}(\boldsymbol{x}) = \boldsymbol{0};\ \boldsymbol{i} = \boldsymbol{1}\ \boldsymbol{at\acute{e}}\ \boldsymbol{p}$$

$$\boldsymbol{g_i}(\boldsymbol{x}) \leq \boldsymbol{0}; \boldsymbol{j} = \boldsymbol{1}\ \boldsymbol{at\acute{e}}\ \boldsymbol{m}$$

onde $\boldsymbol{k}$ é o número de funções objetivos, $\boldsymbol{p}$ é o número de equações de restrição, e $\boldsymbol{m}$ é o número de inequações de restrição. $\boldsymbol{f}(\boldsymbol{x})$ é um vetor $\boldsymbol{k}$-dimensional de funções objetivo. Este género de problema normalmente não possui uma única solução ótima, mas sim um conjunto de soluções ótimas.

Nos problemas de otimização com um único objetivo, as restrições e os contornos das funções objetivo são traçados no espaço de projeto como funções que dependem das variáveis de projeto, porém nos problemas de otimização multi-objetivos, essas restrições e funções objetivos são traçados num espaço de critérios, onde os eixos representam diferentes funções objetivo.

Um conceito que está relacionado com a viabilidade dos pontos de projeto é a acessibilidade. A viabilidade de um projeto implica que nenhuma restrição seja violada no espaço de projeto. A acessibilidade implica que um ponto no espaço de critérios possa estar relacionado com um ponto no espaço de projeto. Enquanto cada ponto no espaço de projeto traduz um ponto no espaço de critérios, o contrário pode não acontecer. Ou seja, um ponto no espaço de critérios pode não traduzir apenas um ponto no espaço de projeto e até pode não ter correspondência.

Num ponto de vista mais clássico, a otimização de uma única função consiste na determinação de um conjunto de pontos fixos, identificando o ótimo global. Porém, o processo de encontrar a solução na otimização de um problema multi-objetivo é ligeiramente mais complexo. Ou seja, é difícil de especificar no que consiste o mínimo de um problema multi-objetivo em que as várias funções objetivo têm comportamento diferente, uma vez que o que diminui o valor de uma função pode aumentar o valor de outra função.

Neste seguimento surge o conceito de *Fronteira de Pareto*. Num critério multi-objetivo, as soluções ótimas resultam do compromisso entre os diferentes objetivos. A *Fronteira de Pareto* é constituída por um conjunto de pontos em que não é possível reduzir uma função objetivo sem o aumento de outra.

Quando um ponto tem outro ponto que reduza várias funções objetivo, este ponto diz-se dominado pelo outro ponto. A questão de dominância apenas se pode aplicar quando se trata de um espaço de critérios. Quando se trata de um espaço de projeto diz-se que o ponto é mais eficiente que o outro ponto.

As soluções que se encontram na *Fronteira de Pareto* designam-se por soluções não dominadas. A Figura 2.4 retrata um exemplo de uma *Fronteira de Pareto* para um caso bidimensional, mas esta pode ser extensível a mais dimensões:

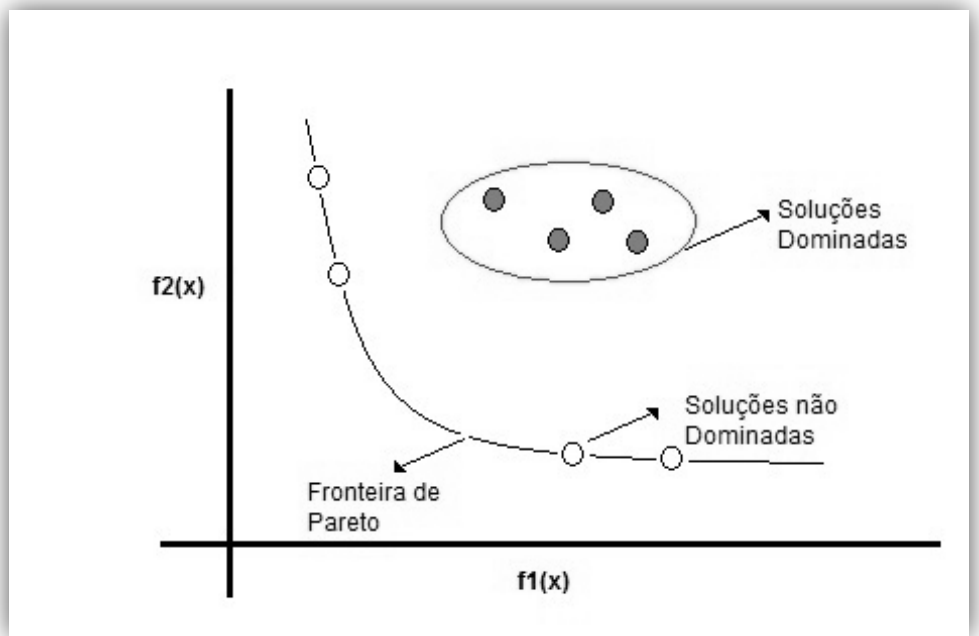


Figura 2.4- *Fronteira de Pareto* de um problema com dois objetivos.

As soluções não dominadas que formam a *Fronteira de Pareto* são chamadas de soluções de compromisso. Isto é, são as soluções consideradas ótimas globais. Qualquer solução que esteja situada na *Fronteira de Pareto* representa uma solução viável para o problema multi-objetivo.

Matematicamente existem infinitos pontos situados sobre a *Fronteira de Pareto*. Portanto é necessário fornecer informação adicional para se escolher apenas um ponto. Essa informação deve refletir a preferência do utilizador. A Figura 2.5 representa um esquema deste processo:

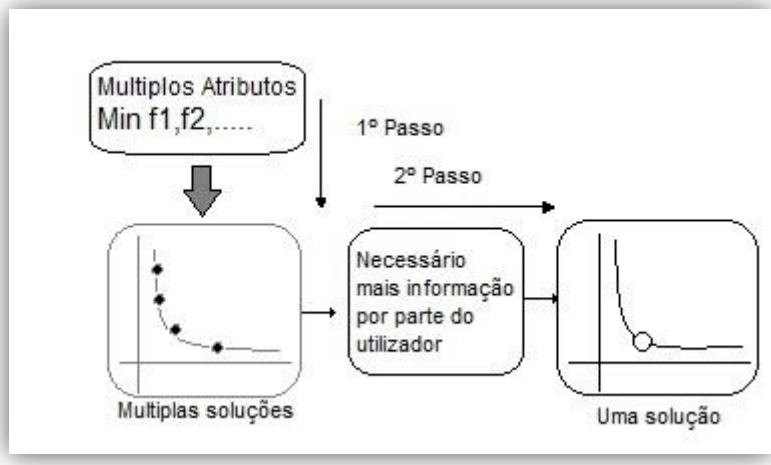


Figura 2.5- Representação esquemática da otimização de um problema multi-objetivo.

2.3.2. Definição do *MOPSO*

O algoritmo MOPSO surgiu com intuito de resolver a principal restrição que o algoritmo *PSO* possuía, resolver problemas de otimização onde haja mais que um objetivo a otimizar. Ou seja, imagine-se uma pessoa que necessita viajar de carro desde Lisboa até Porto o mais depressa possível, a resolução deste problema está ao alcance do algoritmo *PSO*. Porém, se à condição anterior se juntar a condição de minimizar o consumo do carro, o algoritmo *PSO* deixa de conseguir resolvê-lo. É, portanto necessário recorrer ao algoritmo *MOPSO*.

O algoritmo *MOPSO* consiste na determinação do vetor de variáveis $X = [x_1, \dots, x_n]^T$ que minimize o vetor das funções objetivo $f(X) = [f_1(X), \dots, f_k(X)]^T$. Nos problemas de otimização multi-objetivos, como o caso do *MOPSO*, a solução x^* é ótima, se e só se, $x^* \in S$, onde S é o conjunto das soluções possíveis e $f_i(x^*) \leq f_i(x)$ para todo o i e para todo o $x \in S$ num problema de minimização.

Contudo, pode-se criar conflitos devido aos atributos. Devido aos atributos, o algoritmo *MOPSO* pode devolver várias soluções consideradas ótimas. Para solucionar esta questão, as soluções vão ser comparadas entre si, criando soluções dominadas por outras soluções. O conjunto de soluções que não são dominadas por nenhuma outra solução é denominado por *Fronteira de Pareto*, caso se esteja a falar de problemas de otimização com dois objetivos, ou Superfície de Pareto, caso se esteja a falar de problemas de otimização com mais do que dois objetivos. Como se pode constatar na Figura 2.4, quando se quer melhorar um atributo de uma solução não dominada, vai-se sempre piorar a solução em relação a pelo menos um dos outros atributos.

Existem várias técnicas para se obter a *Fronteira de Pareto*. Entre essas técnicas está o C*lusteRing* de soluções. *ClusteRing* agrupa as soluções que apresentem o mesmo comportamento, ou seja, agrupa todas as soluções não dominadas em vários grupos pré-definidos, escolhendo apenas uma solução de cada grupo para representá-las. Desta forma, o *ClusteRing* assegura diversidade nas soluções. *Crowding distance* é outra técnica utilizada para produzir a *Fronteira de Pareto*. Caso exista uma aglomeração de todas as soluções numa determinada zona da *Fronteira de Pareto*, é impossível identifica-la em toda a sua extensão. Desta forma, usa-se o *Crowding* distance para obrigar as partículas a afastarem-se umas das outras. Para se calcular o *Crowding distance* é necessário, primeiro ordenar o conjunto de soluções por ordem crescente de valores da função objetivo. Depois é preciso calcular a distância média entre a partícula anterior e posterior. Assim, as soluções que estejam entre esta distância são consideradas soluções dominadas, o que obriga ao algoritmo procurar novas soluções noutra zona da *Fronteira de Pareto*, obtendo uma melhor definição desta. Na Figura 2.6 está representado um agrupamento:

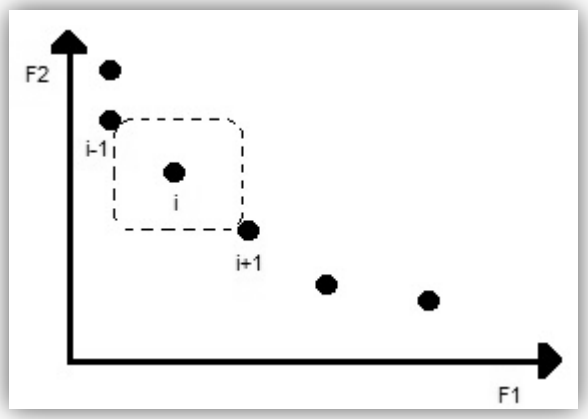


Figura 2.6- Exemplo de *Crowding distance*.

A técnica *Maxmin* é uma técnica bastante utilizada na construção da *Fronteira de Pareto*. Esta técnica começa por comparar os objetivos de uma partícula *i*, com os objetivos de todas as outras partículas, como é possível ver na equação (4):

$$\min_k\left(f_k^i - f_k^j\right) \geq 0 \qquad \textbf{(4)}$$

Desta forma, caso alguma partícula tenha um menor compromisso num dos objetivos que a partícula *i*, esse compromisso é considerado melhor. Porém, para uma partícula dominar outra partícula, é necessário ter um melhor compromisso em todos os objetivos. É por essa razão que se encontra a mínima diferença dos objetivos. Caso esse mínimo seja positivo, a partícula *i* é considerada dominada.

Depois é necessário pegar no conjunto de partículas que dominam a partícula i e verificar quais são aquelas que não são dominadas por outras partículas, tal como está presente na equação (5):

$$\max_{j \neq i}(\min_k(f_k^i - f_k^j)) \geq 0 \qquad \textbf{(5)}$$

Desta forma, o resultado final da técnica *Maxmin* será apenas as soluções não dominadas, ou seja, as soluções pertencentes à *Fronteira de Pareto*.

2.3.3. Detalhes de Implementação

O algoritmo *MOPSO* é baseado no algoritmo *PSO* anteriormente mencionado. Desta forma, o algoritmo *MOPSO* funciona quase da mesma forma que o algoritmo *PSO*. Na inicialização das partículas, é necessário que estas fiquem posicionadas dentro da região de projeto. Após cada partícula ser corretamente inicializada, as partículas vão guardar as suas posições correspondentes na memória de cada uma como ótimos pessoais. Como se trata de um algoritmo multi-objetivo, não vai existir apenas um ótimo global, mas sim um vetor com dimensão do número de partículas, em que cada partícula possui o seu próprio ótimo global. Esta informação também é guardada na memória. Desta forma, fica completo o processo de inicialização. A Figura 2.7 sumariza todo este processo:

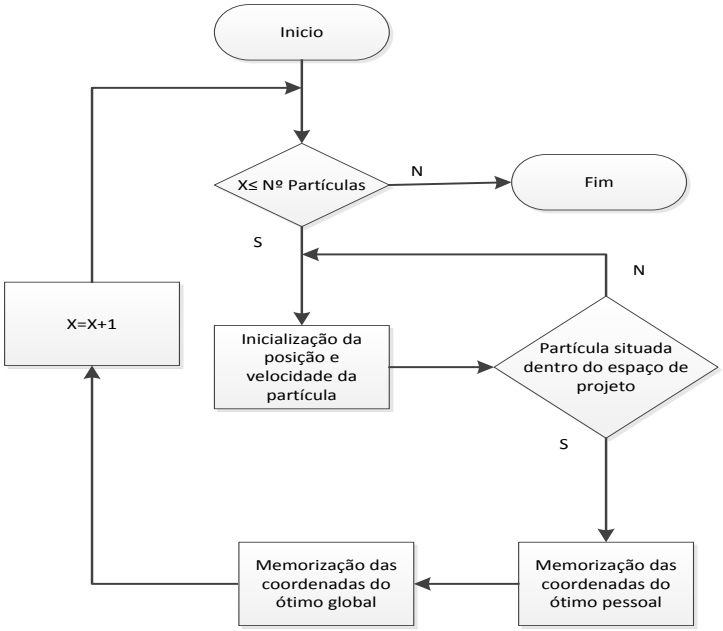


Figura 2.7 - Processo de inicialização das partículas do algoritmo *MOPSO*.

De seguida, é necessário atualizar a posição, através da expressão (2), e velocidade, através da expressão (1), de cada partícula após as iterações, tal como no algoritmo *PSO*. Esta fase só fica concluída quando a posição final de cada partícula, após a atualização, se mantiver dentro da região de projeto. É também necessário atualizar o ótimo pessoal de cada partícula após cada iteração. Se uma partícula chegou a uma posição, onde apresenta um menor custo em todos os atributos do que o seu ótimo pessoal, essa posição passa a ser considerada o ótimo pessoal dessa partícula e é guardado na memória. Isto é, se encontrar uma nova posição que domine o ótimo pessoal, passa a ser considerado o novo ótimo pessoal. Finalmente, o vetor de ótimos globais vai sofrer atualizações após cada iteração ter chegado ao final. Se houver alguma partícula que atinge uma posição com menor custos em todos os atributos que a posição guardada na memória como ótimo global, essa nova posição é considerada como o novo ótimo global e é guardada na memória. A Figura 2.8 representa todo este processo de iterações:

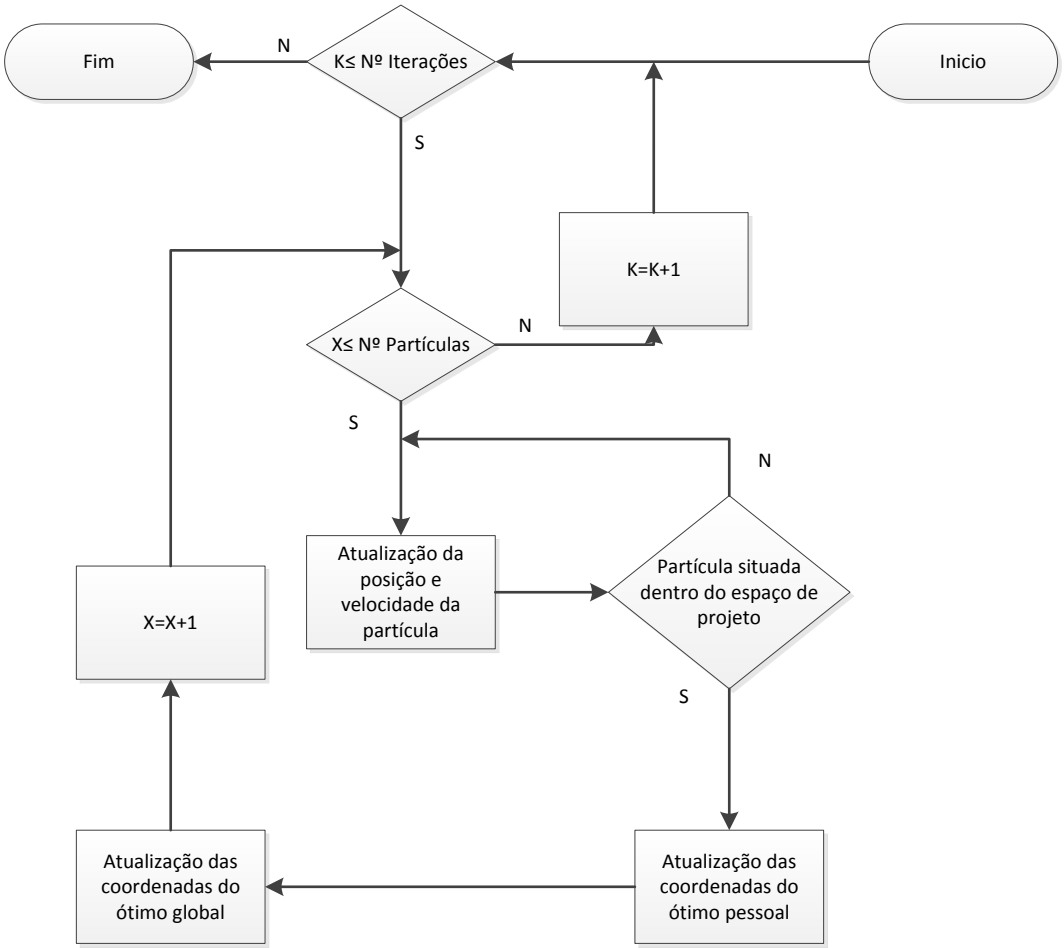


Figura 2.8 - Processo inerente às iterações das partículas do algoritmo *MOPSO*.

A próxima fase é a construção da *Fronteira de Pareto*. Para tal, é preciso a utilização de uma das técnicas apresentadas anteriormente para separar as soluções não dominadas das soluções dominadas. Essa técnica vai ser utilizada no vetor de ótimos globais. O resultado final vai ser um conjunto de soluções ótimas que formam a *Fronteira de Pareto*.

2.4. *PSO Ring*

2.4.1. Definição do *PSO Ring*

Num problema de otimização que tenha múltiplos ótimos globais e locais, pode ser desejável obter-se todos os ótimos globais e alguns ótimos locais que reproduzam uma solução considerada satisfatória. Para determinar as múltiplas soluções ótimas usam-se algoritmos incorporando técnicas de *Niching*, em que se promove o desenvolvimento de subpopulações dentro da população principal. As subpopulações trabalham em paralelo, por isso a probabilidade da população principal ficar presa num ótimo local é reduzida.

No entanto, estes métodos apresentam vários inconvenientes:

- Necessita que os parâmetros de *Niching* sejam especificados, tais como a distância entre dois ótimos ou o número de ótimos;
- Dificuldades em manter as soluções encontradas até ao final do processo;
- Perda de precisão nos resultados caso duas partículas de subpopulações diferentes se intersectam;
- Apenas os ótimos globais são localizados, ignorando os ótimos locais;
- Elevada complexidade computacional.

Foi no âmbito de resolver estes problemas que surgiu o algoritmo *PSO* recorrendo ao uso da topologia em anel. Neste algoritmo, a população principal é dividida em subpopulações com o mesmo número de partículas. Cada partícula pode pertencer a mais do que uma subpopulação. Considerando o exemplo da Figura 2.9 em que o número de partículas que cada subpopulação tem é 3. Cada partícula vai formar uma subpopulação com as suas duas partículas vizinhas e vai, também pertencer às subpopulações que as suas partículas vizinhas vão formar.

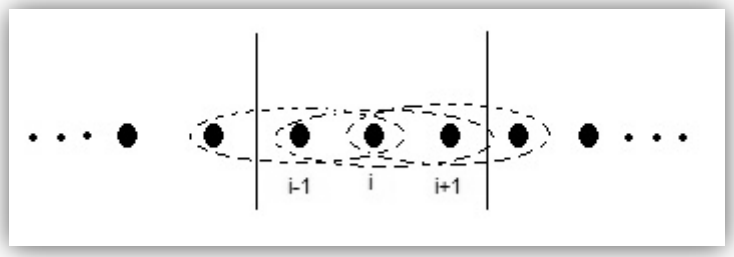


Figura 2.9 – Exemplo de subpopulações composta por 3 partículas da topologia em anel.

A vantagem que este algoritmo apresenta em relação ao *PSO* é o facto de ter uma convergência mais lenta, ideal para encontrar os diversos ótimos da função objetivo. Isto porque, ao possuir uma convergência mais lenta, o ótimo global não vai ter uma influência tão grande no deslocamento do enxame.

2.4.2. Detalhes de Implementação

O algoritmo *PSO* com uso da topologia em anel é baseado no algoritmo *PSO* anteriormente mencionado. Desta forma, o algoritmo funciona quase da mesma forma que o algoritmo *PSO*. Na inicialização das partículas, é necessário que estas fiquem posicionadas dentro da região de projeto. Após cada partícula ser corretamente inicializada, as partículas vão guardar as suas posições correspondentes na memória de cada uma como ótimos pessoais. Não vai existir apenas um ótimo global, mas sim um vetor com dimensão do número de partículas, em que cada partícula possui o seu próprio ótimo global. Esta informação também é guardada na memória. Após todo este processo, vai-se definir as subpopulações existentes, bem como a que subpopulações pertence cada partícula. Desta forma, fica completo o processo de inicialização. A Figura 2.10 sumariza todo este processo.

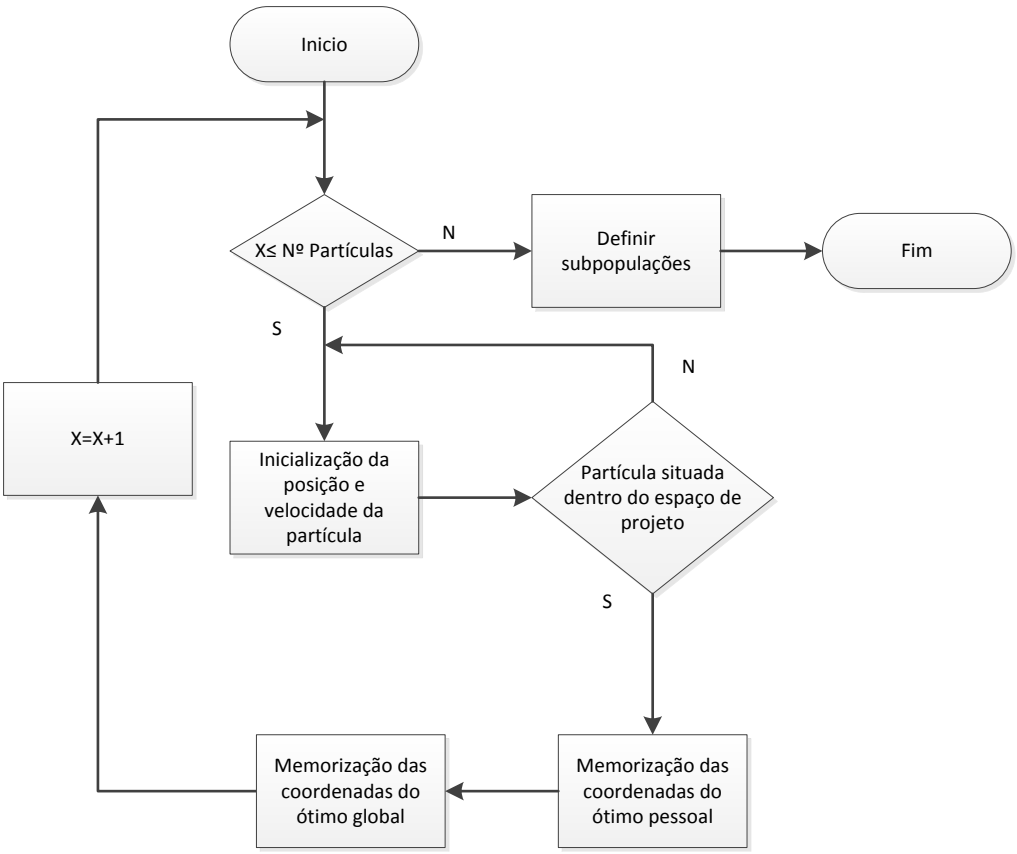


Figura 2.10 – Processo de inicialização do algoritmo *PSO* com uso da topologia em anel.

De seguida, é necessário atualizar a posição, através da expressão (2), e velocidade, através da expressão (1), de cada partícula após as iterações, tal como no algoritmo *PSO*. Esta fase só fica concluída quando a posição final de cada partícula, após a atualização, se mantiver dentro da região de projeto. É também necessário atualizar o ótimo pessoal de cada partícula após cada iteração. Se uma partícula chegou a uma posição, onde o apresenta um menor custo em ambos os atributos do que o seu ótimo pessoal, essa posição passa a ser considerado o ótimo pessoal dessa partícula e é guardado na memória. Isto é, se encontrar uma nova posição que domine o ótimo pessoal, passa a ser considerado o novo ótimo pessoal, tal como no algoritmo *MOPSO*. Finalmente, o vetor de ótimos globais vai sofrer atualizações após cada iteração ter chegado ao final. Se houver alguma partícula de uma subpopulação que atinge uma posição com menor custos em ambos os atributos que a posição guardada na memória como ótimo global, essa nova posição é considerada como o novo ótimo global e é guardada na memória. Este processo é executado para cada subpopulação existente a Figura 2.11 representa todo este processo de iterações.

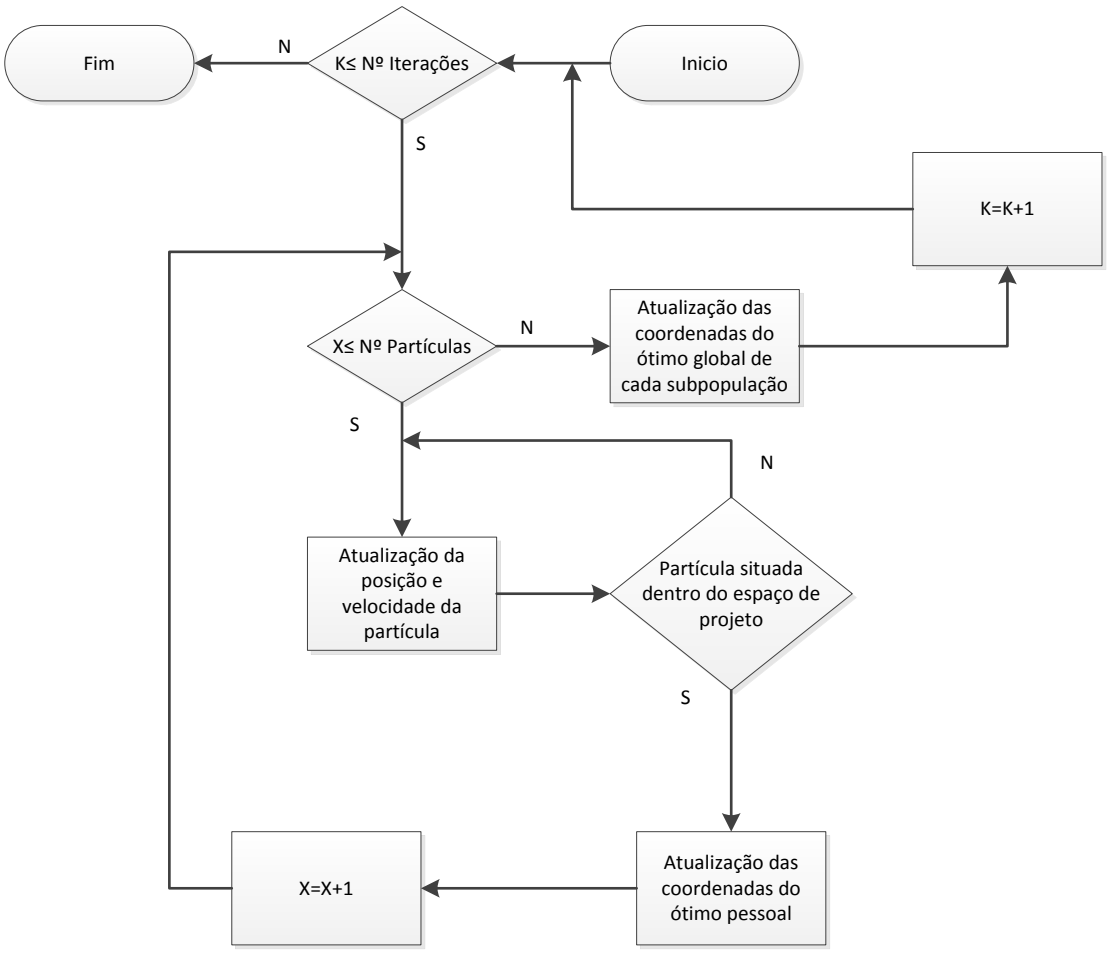


Figura 2.11 - Processo inerente às iterações das partículas do algoritmo *PSO* com uso da topologia em anel.

2.4.3. *PSO Ring* Multi-objetivo

Existem várias adaptações dos algoritmos *Niching* para otimizações de problemas multi-objetivos. Porém, um algoritmo *PSO* que combine a topologia em anel com a técnica *Maxmin* é um algoritmo inovador.

Os algoritmos *Niching* são utilizados para a otimização de problemas multimodais, pois ao terem uma convergência mais lenta, conseguem encontrar não só os ótimos globais, como os ótimos locais. Este algoritmo, ao ser adaptado para problemas de otimização multi-objetivo, fornece uma maior liberdade na escolha da solução final na *Fronteira de Pareto*.

A *Fronteira de Pareto* é construída no espaço de critérios. Desta forma, uma solução que esteja próxima de outra solução no espaço de critérios, poderá estar bastante longe dessa mesma solução no espaço de projeto, como mostra a Figura 2.12:

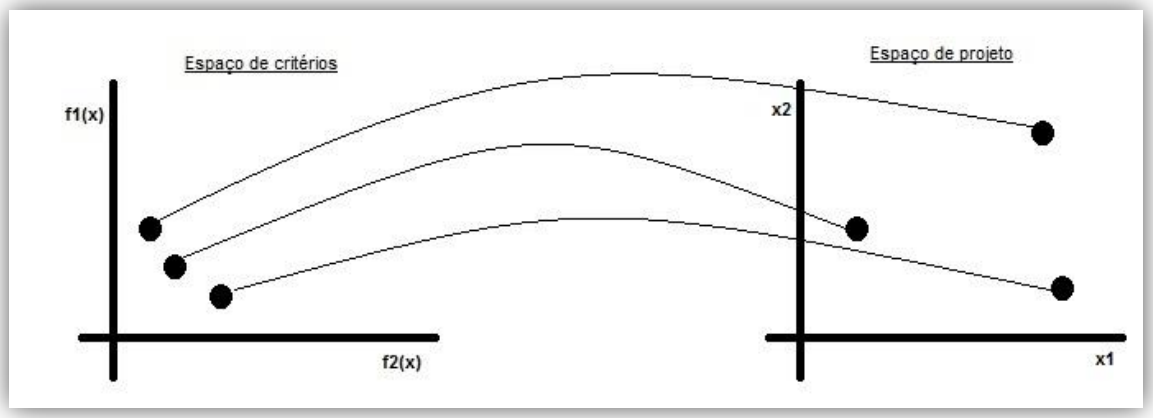


Figura 2.12 – Exemplo para 2 variáveis de projeto da relação entre espaço de critérios e espaço de projeto.

Num problema de otimização onde se esteja a trabalhar com variáveis de projeto, dois ótimos globais seguidos no espaço de critérios podem estar bastante longe no espaço de projeto. Desta forma, um ótimo local que seja dominado pro um ótimo global no espaço de critérios, pode ser considerado uma solução mais eficiente no espaço de projeto.

2.4.3.1. Técnica *Maxmin*

A técnica *Maxmin* é uma técnica utilizada para encontrar as soluções não dominadas. Ou seja, é utilizada para encontrar os ótimos globais. Porém, como o enxame principal foi dividido em vários sub-enxames através da técnica em anel, a técnica *Maxmin* vai ser utilizada em cada um desses sub-enxames de forma a encontrar as suas soluções não dominadas. Desta forma, e como se trata de um algoritmo de *Niching*, a técnica *Maxmin* vai devolver não só os ótimos locais, como os ótimos globais que encontrar. Todo este processo está representado na Figura 2.13:

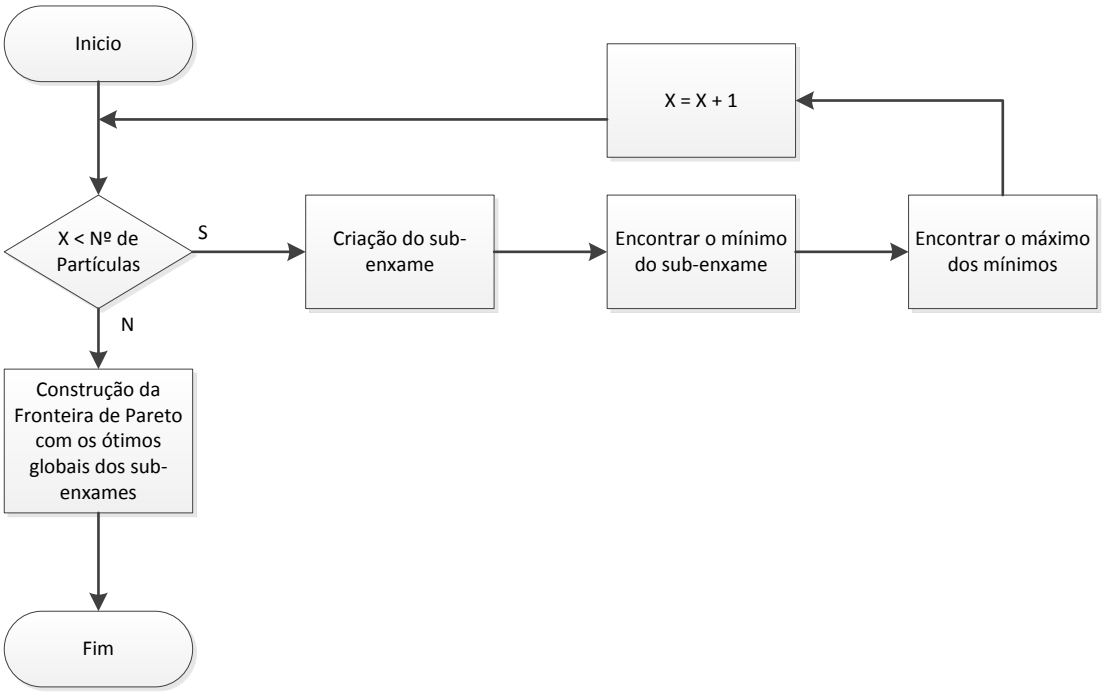


Figura 2.13 - Processo da técnica *Maxmin*.

3. Trabalho Desenvolvido

Este capítulo descreve todo o trabalho realizado na construção da aplicação. O primeiro passo é saber separar o trabalho do algoritmo e as informações que o utilizador deve fornecer. Isto é, para qualquer problema de otimização, vai existir uma fase onde o utilizador tem que definir a função objetivo, as restrições e os parâmetros específicos para esse problema e uma fase onde o processo de procura da melhor solução por parte dos algoritmos começa. O processo de procura por parte dos algoritmos funciona de igual forma para qualquer que seja o problema em questão.

Desta forma, criou-se uma estrutura geral para qualquer que seja o problema apresentado. Essa estrutura não pode ser alterada por parte do utilizador. Faz parte dessa estrutura a inicialização das partículas e o processo de atualização das posições e velocidades das mesmas.

Durante o processo de procura, a estrutura criada necessita da informação de funções auxiliares. Essas funções auxiliares vão ser introduzidas pelo utilizador. Como foi dito anteriormente, essas funções contêm a informação sobre a função objetivo, sobre as restrições existentes e que limitam o espaço de procura e sobre o valor do custo inerente à posição atual de cada partícula.

Cada função auxiliar introduzida pelo utilizador tem que respeitar uma estrutura definida, de modo ao correto funcionamento da aplicação. As funções auxiliares vão ser explicadas uma a uma no próximo subcapítulo. Toda a aplicação foi implementada na plataforma *Matlab*, recorrendo à linguagem *matlab*.

3.1. Objetivos

Este trabalho tem como objetivo a criação de uma aplicação capaz de resolver qualquer problema de otimização. Um problema de otimização está associado a uma função objetivo, que por sua vez, é um modelo matemático que representa de forma quantificada o grau de desempenho alcançado no cumprimento dos objetivos do problema. Essa função pode ser maximizada ou minimizada de acordo com o objetivo final. Para tal é necessário recorrer a algoritmos de otimização, mais concretamente, algoritmos baseados no *PSO*. Os problemas de otimização podem ser classificados em unimodais e multimodais, consoante o número de ótimos presentes na região de procura, ou com um único objetivo e multi-objetivo, consoante o número de objetivos a otimizar. Desta forma, o algoritmo *PSO* escolhido vai tratar os problemas

de otimização unimodais e com um único objetivo, enquanto o algoritmo *PSO Ring* vai tratar os problemas multimodais com um único objetivo. Ainda falta tratar os problemas de otimização multi-objetivos. Assim, recorre-se ao algoritmo *MOPSO* para resolver este género de problemas de otimização. Também se pode recorrer ao algoritmo *PSO Ring* para otimizar problemas multi-objetivos, uma vez que este algoritmo foi implementado, com o auxílio da técnica *Maxmin*, de forma a resolver este tipo de problemas.

3.2. Especificações

A Figura 3.14 representa o menu inicial da aplicação desenvolvida neste trabalho:

Figura 3.14 – Menu inicial.

No painel com o nome de Método, presente na Figura 3.14, escolhe-se o algoritmo que se pretende utilizar. Como se pode observar, existem três escolhas, o algoritmo *PSO*, o algoritmo *MOPSO* e o algoritmo *PSO* com o uso da topologia em anel. Essa escolha é guardada numa variável chamada *met*, com o valor 1, 2 ou 3 respetivamente, consoante a opção escolhida.

O próximo passo é preencher os parâmetros existentes no menu inicial, representado na Figura 3.14. Esses parâmetros são necessários para o correto funcionamento dos algoritmos. São números inteiros, pelo que apenas tomam valores sem casas decimais. No campo *Dimensão* será preenchido a dimensão da região de projeto. Esta está limitada a 3 dimensões devido a razões de performance da interface gráfica. No campo *Nº partículas* será preenchido com o número de partículas a serem espalhadas de forma aleatória pela região de projeto. No campo *Nº iterações* será preenchido com o número de atualizações da posição e velocidade que as partículas irão sofrer durante o processo de procura do ótimo. Nos campos *c2*, *c1* e *cv* serão preenchidos com os números das constantes de atualização da velocidade das partículas, equação (1), onde *c2* representa a influência da melhor posição encontrado pelo enxame na atualização da partícula, *c* representa a influência da melhor posição encontrado pela partícula na atualização da mesma e

cv representa a constante de inércia ω. Esta informação será acrescentada a uma estrutura existente chamada de *parametros* com os nomes de *dimensão*, *particulas*, *iteracoes*, *c2*, *c1* e *cv*. Depois é preciso pressionar o botão *Começar* para que a função objetivo seja processada segundo os dados fornecidos.

Assim que o botão *Começar* for pressionado, o utilizador tem a possibilidade de observar a evolução do algoritmo ao longo das iterações através de uma barra de espera, representada na Figura 3.15:

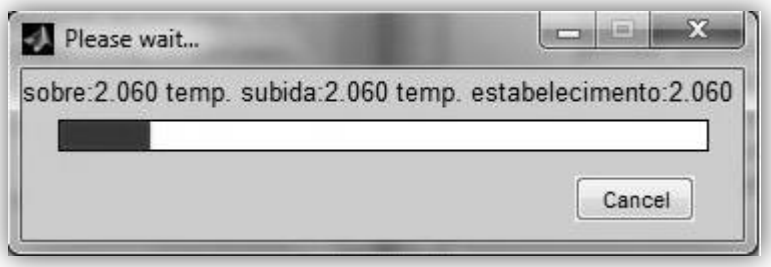


Figura 3.15 – Barra de espera.

Sempre que o utilizador achar que os seus objetivos já foram atingidos, pode interromper a procura do algoritmo, ao carregar no botão *cancel*, e ficar com a melhor solução global encontrada até à altura. Ao carregar no botão *cancel*, é gerado um ficheiro *.mat* denominado por *base*, que guarda a informação de uma estrutura *base* com a informação sobre todo o enxame (posições e velocidades das partículas, bem como os seus melhores pessoais e o melhor global), qual a iteração em curso e que partícula estava a sofrer a atualização na altura. Caso o utilizador queira retomar o processo cancelado basta selecionar o *checkbox* Continuar, presente na Figura 3.14, selecionar o algoritmo que estava a ser usado nesse processo e carregar novamente no botão *Começar*. Desta forma, o processo é retomado a partir da situação em que foi cancelado.

No final, para o utilizador observar os resultados obtidos, basta carregar no botão *Resultados* presente na Figura 3.14. Este botão, consoante o algoritmo que foi utilizado, abre uma outra janela com a tabela dos valores obtidos, como demonstra a Figura 3.16 para o caso de se ter utilizado o algoritmo *PSO*:

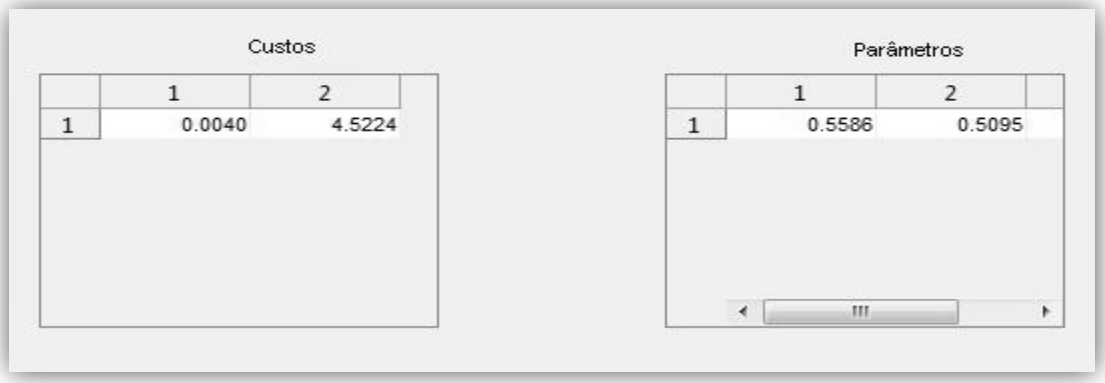


Figura 3.16 – Exemplo do resultado obtido através do algoritmo *PSO*.

Nesta figura estão representados os valores obtidos para a otimização de um controlador PID em série com o sistema em anel fechado, presente na Figura 3.17:

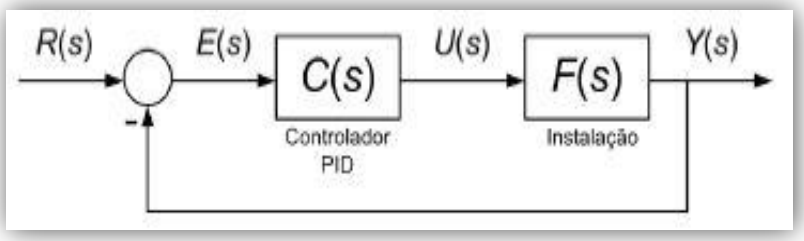


Figura 3.17 – Sistema de blocos do sistema de controlo.

A tabela dos *Custos* demonstra os valores obtidos para as restrições impostas. Neste exemplo impôs-se as seguintes restrições à função objetivo:

- Função objetivo:
  - Sobrelevação + Tempo de subida;
- Restrições:
  - Sobrelevação inferior a 10%; e
  - Tempo de subida a 90% inferior a 5 segundos.

Nas colunas estão presentes as variáveis de projeto, onde a primeira coluna representa o valor do Kp, na segunda o valor do Td e na última o valor do Ti. Como se pode constatar, os custos vão cumprir as restrições inicialmente impostas. Na tabela dos *Parâmetros* estão representados os valores necessários introduzir no controlador PID de forma a se obter os custos da tabela do lado. Na figura 3.16, a tabela dos *Parâmetros* tem 3 parâmetros, apesar de um deles não estar visível, basta andar com o *scroll* para o lado para se observar o valor obtido.

Caso o algoritmo utilizado seja o *MOPSO*, os resultados vão ser mostrados numa janela diferente, presente na Figura 3.18:

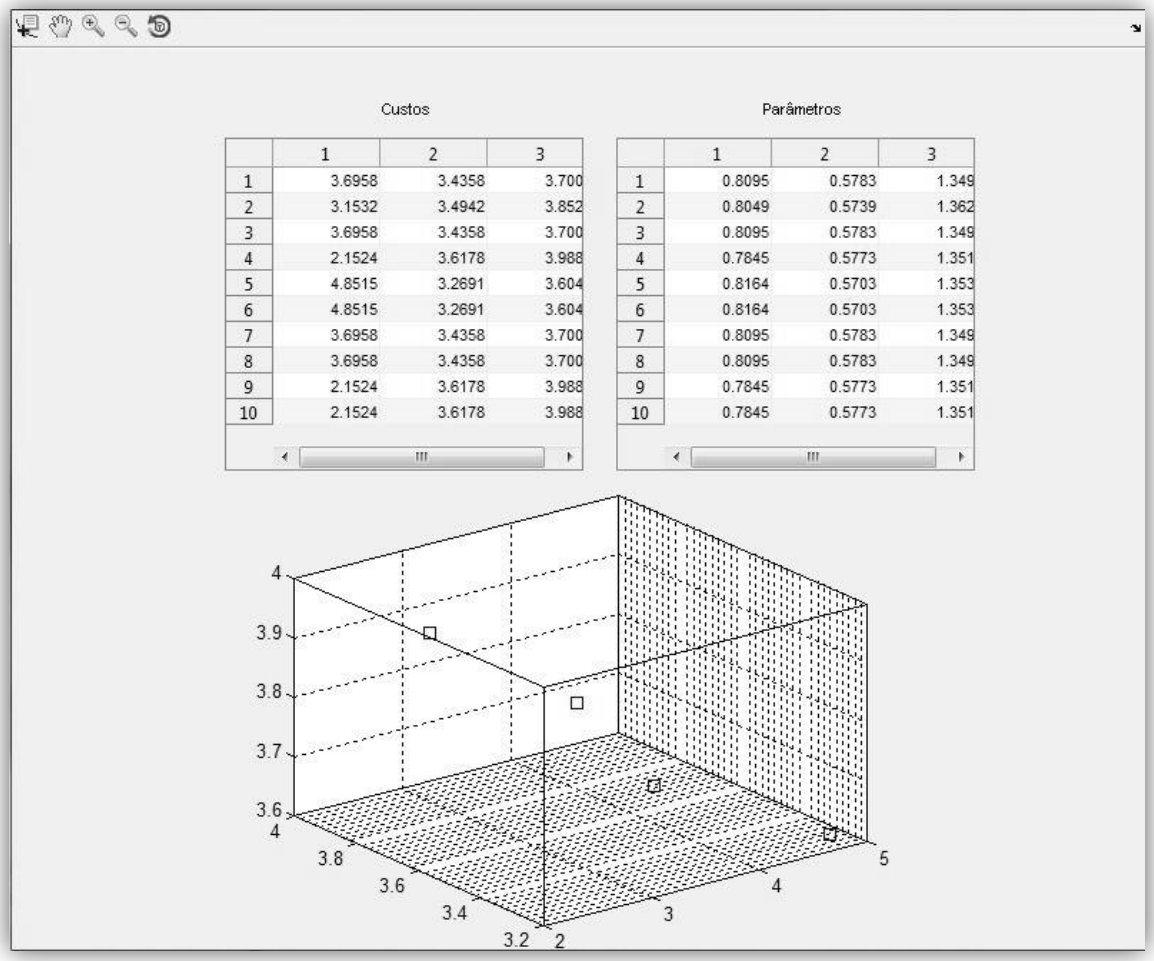


Figura 3.18 - Exemplo do resultado obtido através do algoritmo *MOPSO*.

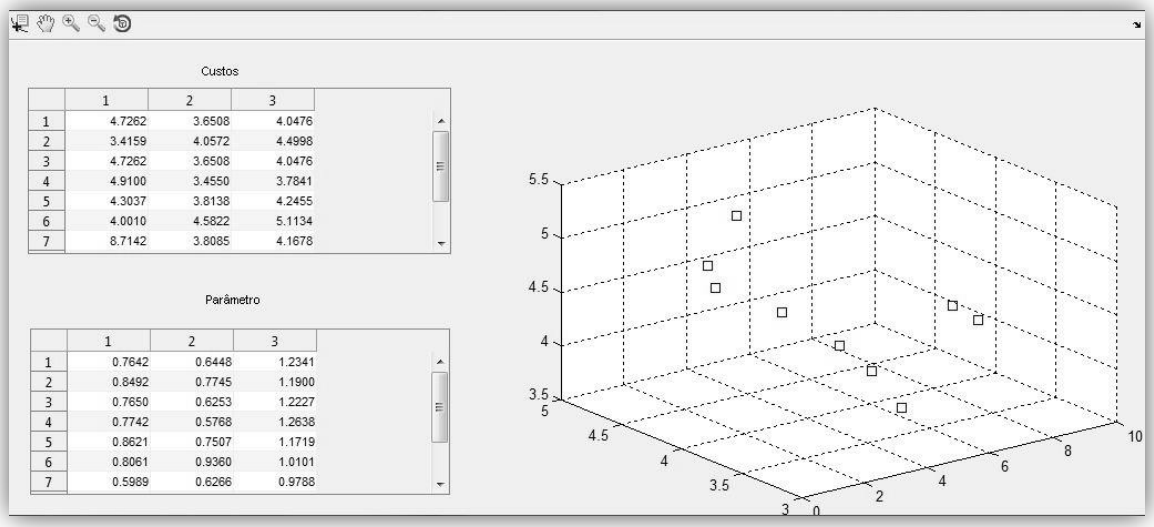
Na Figura 3.18 estão representados os valores obtidos para a otimização de um controlador PID em série com o sistema em anel fechado. A tabela dos *Custos* demonstra os valores obtidos para as restrições impostas. Neste exemplo impôs-se as seguintes restrições:

- Sobrelevação inferior a 10%;
- Tempo de subida a 90% inferior a 5 segundos; e
- Tempo de estabelecimento a 5% inferior a 15 segundos.

Como se pode observar na tabela *Custos* da Figura 3.18, todas as restrições foram cumpridas pelas 10 partículas inicialmente introduzidas como parâmetros. Na tabela *Parâmetros* estão representados os valores necessários introduzir no controlador PID de forma a se obter os custos da tabela do lado. Na Figura 3.18 está ainda representada a *Fronteira de Pareto*, neste caso como se tem 3 dimensões trata-se de uma Superfície de Pareto. Foi colocado as *grids* de ambos os eixos de modo a permitir ao utilizador a correta perceção de onde os pontos estão situados. Foram ainda adicionado à janela algumas opções extras no visionamento da *Fronteira de Pareto*, tais como *zoom in* e *zoom out*, a opção de se movimentar livremente

pelo gráfico, a opção de ver a informação dos pontos selecionados e a opção de rodar os eixos do gráfico. Todas estas opções extras permitem ao utilizador perceber melhor, tanto a *Fronteira de Pareto* obtida, como as soluções ótimas obtidas.

Por último, caso o algoritmo utilizado seja o *PSO Ring* os resultados vão ser mostrados numa janela diferente, presente na Figura 3.19:



	1	2	3
1	4.7262	3.6508	4.0476
2	3.4159	4.0572	4.4998
3	4.7262	3.6508	4.0476
4	4.9100	3.4550	3.7841
5	4.3037	3.8138	4.2455
6	4.0010	4.5822	5.1134
7	8.7142	3.8085	4.1678

	1	2	3
1	0.7642	0.6448	1.2341
2	0.8492	0.7745	1.1900
3	0.7650	0.6253	1.2227
4	0.7742	0.5768	1.2638
5	0.8621	0.7507	1.1719
6	0.8061	0.9360	1.0101
7	0.5989	0.6266	0.9788

Figura 3.19 - Exemplo do resultado obtido através do algoritmo *PSO Ring*.

Esta janela é muito semelhante à janela do algoritmo *MOPSO*. Possui as mesmas tabelas, bem como um gráfico com a Superfície de Pareto e todos os mesmos extras que possibilitam ao utilizador uma melhor compreensão do gráfico.

Para o utilizador escolher uma solução ótima, tanto na Figura 3.18 como na Figura 3.19, primeiro seleciona o ponto da Superfície de Pareto desejável para ser a sua solução ótima. De seguida, e através dos custos indicados no gráfico do ponto selecionado, retira o índice da partícula, ou seja o número da linha da tabela *Custos*. Finalmente, a linha da tabela *Parâmetros* que tenha o mesmo índice da partícula selecionada, devolve os parâmetros associados a essa escolha.

3.3. Rotinas do utilizador

Este trabalho é dividido em duas partes, uma parte onde o processamento é igual para qualquer que seja o problema em questão e uma parte onde o processamento vai variar de problema para problema. O processamento genérico para qualquer problema está descrito na secção anterior. Nesta secção vai ser abordado o envolvimento do utilizador para o correto funcionamento desta aplicação.

Existem aspetos que variam consoante o problema em questão, tais como a função objetivo ou a função de custos. Na Figura 3.14, o botão *Inicialização* chama uma função *optinicializar* que tem como objetivo introduzir no programa o seguinte ficheiro .mat:

- Um ficheiro chamado *inicializacao*, que possui uma estrutura denominada também por *inicializacao* com os seguintes parâmetros:
  1. Os coeficientes da combinação linear das várias funções objetivo para utilização do algoritmo *PSO* que se quer otimizar num vetor chamado *fucobj*;
  2. a informação se se pretende inicializar as partículas num determinado ponto guardado na variável *ini*, onde 1 significa que se pretende e 0 o contrário;
  3. Caso *ini* for igual a 1, os valores do ponto e do raio de convergência são guardados nas variáveis *x*, *y*, *z*, *intervalox*, *intervaloy* e *intervaloz*.

No caso particular para o exemplo da Figura 3.20, a função *optinicializar* introduzir também o seguinte ficheiro .mat

- Um ficheiro chamado *parametros*, que possui uma estrutura denominada também por *parametros* com os seguintes parâmetros:
  1. a informação sobre o valor das restrições numa variável *restricoes*;
  2. a informação relativo à exclusividade de soluções positivas que é guardada numa variável *positivo*.

A forma como estes ficheiros são introduzidos e preenchidos é uma decisão que cabe ao utilizador. Na Figura 3.20 está presente um exemplo de como todo este processo pode ser implementado:

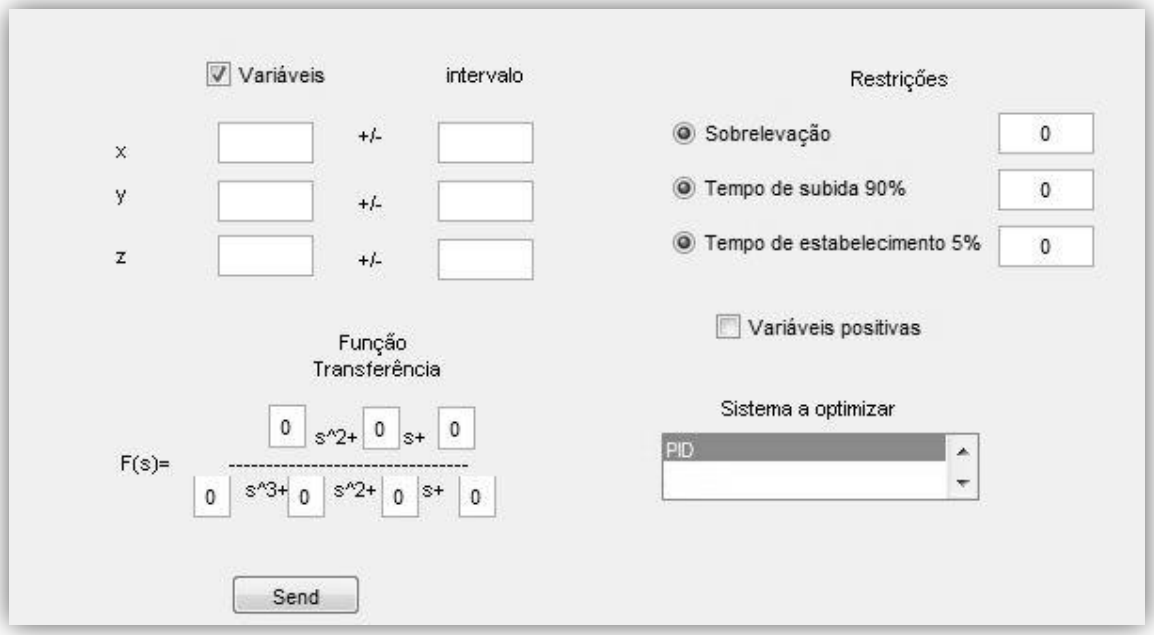


Figura 3.20 – Exemplo da função *optinicializar*.

Neste exemplo pode-se observar quais são os parâmetros que o programa requer para funcionar corretamente. O parâmetro essencial é a função objetivo, que neste exemplo, como se trata da otimização de um controlador PID em série com o sistema em anel fechado, é representado pela função transferência. Outro parâmetro necessário são as restrições impostas à função objetivo, restrições essas que vão delimitar a região de procura da solução ótima. Neste exemplo, as restrições são o valor máximo da sobrelevação, do tempo de estabelecimento e do tempo de subida. Existe outra restrição ainda neste exemplo, caso se pretenda obter variáveis exclusivamente positivas no conjunto de soluções ótimas basta selecionar o checkbox com o nome *Variáveis positivas*. Outro parâmetro, que pode ou não ser introduzido, é a inicialização das partículas num certo ponto (x, y e/ou z) e com um certo raio de convergência. Caso se queira inicializar as partículas, neste exemplo, é necessário selecionar o checkbox *Variáveis*, caso contrário não é preciso fazer nada.

Durante o processamento de otimização, os algoritmos necessitam de aceder a alguma informação específica do problema em questão. Essa informação vai estar disponível sobre a forma de funções. A primeira função que vai ser necessária é designada por *verificacao*. Esta função tem o objetivo de verificar se as partículas criadas no processo de inicialização se encontram dentro da região de projeto do sistema. Esta função necessita de levar como parâmetro uma estrutura com a posição e velocidade atual da partícula e devolve uma estrutura chamada *str* com um vetor com o valor da resposta produzida pelas soluções atuais (*fun_obj*), o número de restrições existentes (*num_obj*) e uma variável *cumpre* que devolve 1 caso a posição

da partícula se encontre dentro da região de projeto e 0 caso contrário. A estrutura *str* pode ter alores adicionais específicos para cada otimização, como é o exemplo do vetor *resposta* com os valores das restrições da resposta produzida pela soluções atuais. Um exemplo desta função auxiliar pode ser vista na Figura 3.21:

```
function [ str ] = verificacao( particula )

% Variaveis
global var;
r = 0;
c = 0;

Fs = transfer_function(particula);
[Y,T] = step(Fs);
sob = 100*((max(Y)- 1)/1);
str.resposta(1) = sob;
if var.parametros.restricoes(1) ~= -100
    r = r+1;
    if sob >= 0 && sob < var.parametros.restricoes(1)
        c = c+1;
        str.fun_obj(c) = sob;
    end
end
index=find(Y>0.9);
if ~isempty(index)
    tp_sub = T(index(1));
    str.resposta(2) = tp_sub;
    if var.parametros.restricoes(2) ~= -100
        r = r+1;
        if tp_sub >= 0 && tp_sub < var.parametros.restricoes(2)
            c = c+1;
            str.fun_obj(c) = tp_sub;
        end
    end
else
    str.resposta(2) = 100000000000;
end
```

```
index=find(abs(Y-1) <= 0.05);
if ~isempty(index)
    tp_est = T(index(1));
    str.resposta(3) = tp_est;
    if var.parametros.restricoes(3) ~= -100
        r = r+1;
        if tp_est >= 0 && tp_est < var.parametros.restricoes(3)
            c = c+1;
            str.fun_obj(c) = tp_est;
        end
    end
else
    str.resposta(3) = 1000000000000;
end

str.num_obj = r;
if r == c
    str.cumpre = 1;
else
    str.cumpre = 0;
end
return
end
```

Figura 3.21 – Exemplo da função *verificacao*.

No exemplo presente na Figura 3.21, está-se perante um problema de controlo, onde o objetivo é obter uma sobrelevação, um tempo de subida e um tempo de estabelecimento inferiores aos impostos pelo utilizador. Esta função recorre à ajuda de outra função, introduzida pelo utilizador, para obter a função transferência atual do sistema, chamada *transfer_function*. Este exemplo demonstra que o utilizador também pode criar funções auxiliares dentro de outras funções auxiliares caso o problema assim o exija.

A segunda função necessária é designada por *custo*. Esta função representa a função de custos. A função leva como parâmetros o algoritmo utilizado (*met*), que pode ter o valor 1, caso seja o *PSO*, o valor 2, caso seja o *MOPSO*, ou o valor 3, caso seja o *PSO Ring*, e a estrutura devolvida pela função *verificacao*. Devolve uma estrutura (*fx*) com o valor do custo e, caso se trate do algoritmo *PSO*, um vetor com os índices dos objetivos a serem tratados. Na Figura 3.22 está representado um exemplo desta função:

```
function [ fx ] = custo( met, str )

%Variaveis auxileares
switch met
    case 1
        switch str.num_obj
            case 1
                if str.resposta(1) == str.fun_obj(1)
                    fx.valor = str.resposta(1);
                    fx.vec(1) = 1;
                elseif str.resposta(2) == str.fun_obj(2)
                    fx.valor = str.resposta(2);
                    fx.vec(1) = 2;
                else
                    fx.valor = str.resposta(3);
                    fx.vec(1) = 3;
                end
            case 2
                if    str.resposta(1)    ==    str.fun_obj(1)    &&
str.resposta(2) == str.fun_obj(2)
                    fx.valor = str.resposta(1) + str.resposta(2);
                    fx.vec(1) = 1;
                    fx.vec(2) = 2;
                elseif   str.resposta(1)   ==   str.fun_obj(1)   &&
str.resposta(3) == str.fun_obj(2)
                    fx.valor = str.resposta(1) + str.resposta(3);
                    fx.vec(1) = 1;
                    fx.vec(2) = 3;
                else
                    fx.valor = str.resposta(2) + str.resposta(3);
                    fx.vec(1) = 2;
                    fx.vec(2) = 3;
                end
            case 3
                fx.valor  =  str.resposta(1)  +  str.resposta(2)  +
str.resposta(3);
                fx.vec(1) = 1;
```

```
                fx.vec(2) = 2;
                fx.vec(3) = 3;
        end
    case 2
        switch str.num_obj
            case 1
                if str.resposta(1) == str.fun_obj(1)
                    fx = str.resposta(1);
                elseif str.resposta(2) == str.fun_obj(2)
                    fx = str.resposta(2);
                else
                    fx = str.resposta(3);
                end
            case 2
                if    str.resposta(1)    ==    str.fun_obj(1)    &&
str.resposta(2) == str.fun_obj(2)
                    fx = [str.resposta(1);str.resposta(2)];
                elseif   str.resposta(1)   ==   str.fun_obj(1)   &&
str.resposta(3) == str.fun_obj(2)
                    fx = [str.resposta(1);str.resposta(3)];
                else
                    fx = [str.resposta(2);str.resposta(3)];
                end
            case 3
                fx                                                  =
[str.resposta(1);str.resposta(2);str.resposta(3)];
        end
    case 3
        switch str.num_obj
            case 1
                if str.resposta(1) == str.fun_obj(1)
                    fx = str.resposta(1);
                elseif str.resposta(2) == str.fun_obj(2)
                    fx = str.resposta(2);
                else
                    fx = str.resposta(3);
                end
```

```
            case 2
                if    str.resposta(1)    ==    str.fun_obj(1)    &&
str.resposta(2) == str.fun_obj(2)
                    fx = [str.resposta(1);str.resposta(2)];
                elseif   str.resposta(1)   ==   str.fun_obj(1)   &&
str.resposta(3) == str.fun_obj(2)
                    fx = [str.resposta(1);str.resposta(3)];
                else
                    fx = [str.resposta(2);str.resposta(3)];
                end
            case 3
                fx                                                  =
[str.resposta(1);str.resposta(2);str.resposta(3)];
        end
end
return
end
```

Figura 3.22 – Exemplo da função *custo*.

Caso o utilizador tenha posto a condição do valor final ser constituído exclusivamente por valores positivos, é necessário recorrer a outra função, designada por *verificacao_positive*. Esta função leva como parâmetros posição atual da partícula e uma variável que indica se o utilizador pretende valores exclusivamente positivos, com o valor 1, ou não, com o valor 0. Devolve outra variável onde indica se as restrições foram cumpridas, com o valor 1, ou não, com o valor 0. Na Figura 3.23 está representado um exemplo desta função:

```
function [ positive ] = verificacao_positive( posicao, positivo )

if positivo == 1
    if posicao(:) >= 0
        positive = 1;
    else
        positive = 0;
    end
else
    positive = 1;
end
return
```

```
end
```

Figura 3.23 – Exemplo da função *verificacao_positive*.

A próxima função necessária é designada por *verificacao_iteracao*. Esta função verificar se a partícula, ao ser atualizada, não saiu da região de projeto limitada pelas restrições impostas inicialmente. Esta função tem como parâmetros a posição e velocidade da partícula após a atualização e devolve uma estrutura *str* igual à estrutura devolvida pela função *verificacao*. Esta função é diferente da função *verificacao*, para o caso do utilizador introduzir uma função objetivo que necessite de maior verificação durante as iterações do que na inicialização. O exemplo apresentado anteriormente da otimização do controlador PID em série com o sistema em anel fechado é um bom exemplo deste aspeto. Ao inicializar a posição das partículas, estas ficam com um valor aleatório entre 0 e 1, o que mantém o sistema estável. Porém, à medida que as posições são atualizadas, estas podem tomar valores negativos, caso o utilizador não selecione a opção de ter sempre valores positivos. Desta forma, o sistema pode ficar instável. Ou seja, é necessário uma nova restrição que faça com que só sejam aceites sistemas estáveis. Este exemplo está representado na Figura 3.24:

```
function [ str ] = verificacao_iteracao( particula )

global var;

Fs = transfer_function(particula);
r = 0;
c = 0;

%sistema é estavel?
polos=pole(Fs);
if real(polos(:)) <= 0
    [Y,T]=step(Fs);

    sob = 100*((max(Y)- 1)/1);
    str.resposta(1) = sob;
    if var.parametros.restricoes(1) ~= -100
        r = r+1;
        if sob >= 0 && sob < var.parametros.restricoes(1)
            c = c+1;
            str.fun_obj(c) = sob;
        end
```

```
end

index=find(Y>0.9);
if ~isempty(index)
    tp_sub = T(index(1));
    str.resposta(2) = tp_sub;
    if var.parametros.restricoes(2) ~= -100
        r = r+1;
        if tp_sub >= 0 && tp_sub < var.parametros.restricoes(2)
            c = c+1;
            str.fun_obj(c) = tp_sub;
        end
    end
else
    str.resposta(2) = 1000000000;
end

index=find(abs(Y-1) <= 0.05);
if ~isempty(index)
    tp_est = T(index(1));
    str.resposta(3) = tp_est;
    if var.parametros.restricoes(3) ~= -100
        r = r+1;
        if tp_est >= 0 && tp_est < var.parametros.restricoes(3)
            c = c+1;
            str.fun_obj(c) = tp_est;
        end
    end
else
    str.resposta(3) = 1000000000;
end

str.num_obj = r;

if r == c
    str.cumpre = 1;
else
```

```
        str.cumpre = 0;
    end
else
    str.cumpre = 0;
end

return

end
```

Figura 3.24 – Exemplo da função *verificacao_iteracao*.

Na consulta de resultados, também existem funções que vão ser criadas pelo utilizador. Quando se carrega no botão *Resultados* do menu inicial (Figura 3.14), aparece uma nova janela com a informação dos resultados obtidos na forma de uma tabela. Na Figura 3.17, Figura 3.18 e Figura 3.19, se o utilizador quiser saber mais informação sobre um resultado, basta selecionar uma célula. Ao fazer isso, vai ser chamada uma função designada de *optlancar_PSO* que tem como parâmetros de entrada o índice da partícula selecionada na célula, designado por *indice*, bem como uma estrutura *aux* com toda a informação dessa partícula. A Figura 3.25 demonstra o aspeto desta função associado ao exemplo anteriormente mencionado:

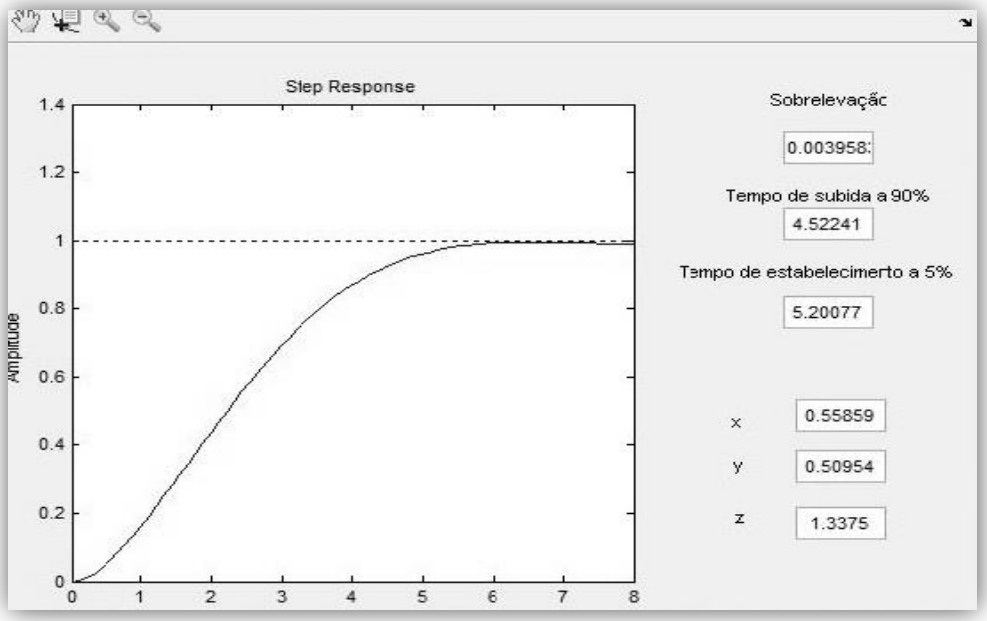


Figura 3.25 – Exemplo da função *optlancar_PSO*.

A Figura 3.25, para além de mostrar a resposta ao degrau unitário do sistema otimizado, também mostra os parâmetros utilizados no controlador PID, onde *x*, *y* e *z* representam respetivamente Kp, Td e Ti, e os custos da resposta. Foi ainda adicionado as opções de *zoom in* e *zoom out* caso o utilizador queira ver melhor a resposta, a opção de se movimentar livremente pela resposta e de ver a informação dos pontos selecionados na resposta. Todas estas opções permitem ao utilizador uma variedade de escolhas no tratamento dos resultados.

3.4. Estrutura Interna

Nesta secção vai ser apresentada, tal como o título indica, a estrutura interna do funcionamento da aplicação. A Figura 3.26 representa o fluxograma de todo o processo:

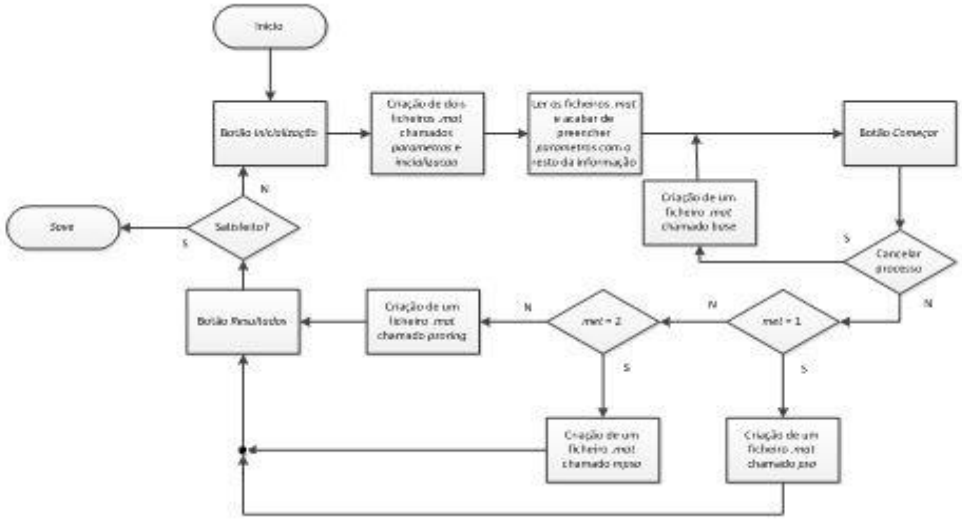


Figura 3.26 – Fluxograma do funcionamento da aplicação.

Para começar todo o processo, é necessário ter conhecimento da função objetivo e das restrições do problema em causa. Para tal, e como já foi mencionado anteriormente, carrega-se no botão *Inicialização*, que cria dois ficheiros do tipo *.mat*, um chamado *inicializacao* e outro chamado *parametros*.

Esses ficheiros são mais tarde carregados para a aplicação, onde a estrutura *parametros*, carregada do ficheiro com o mesmo nome, é completada com a informação da dimensão da partícula, número de partículas, número de iterações e constantes de inércia existentes no manu inicial (Figura 3.14).

De seguida escolhe-se o algoritmo de otimização que se pretende utilizar e a escolha fica guardada num variável chamada *met*. Esta variável toma o valor 1 caso o algoritmo escolhido seja o *PSO*, o valor 2 caso seja o *MOPSO* e o valor 3 caso seja o *PSO Ring*. O próximo passo é carregar no botão *Começar*.

Como já foi referido anteriormente, o utilizador tem a possibilidade de observar a evolução do processo de otimização através de uma barra de espera (Figura 3.15). Caso o utilizador se sinta satisfeito com os valores encontrados, pode em qualquer altura cancelar o resto do processamento e ficar com esses valores. Se o processo for cancelado, dá-se a criação de um ficheiro *.mat* designado de *base*. Caso o utilizador queira retomar o processo cancelado basta selecionar a *checkbox* Continuar, presente na Figura 3.14, selecionar o algoritmo que estava a ser usado nesse processo e carregar novamente no botão *Começar*. Desta forma, o processo é retomado a partir da situação em que foi cancelado.

No final de todas as iterações e caso a variável *met* seja 1, é criado um ficheiro *.mat* denominado por *PSO*, que leva uma estrutura *PSO* com a posição e valores dos objetivos na

melhor posição global e com o vetor dos índices a serem tratados, devolvido na função *custo*. Este ficheiro vai ser utilizado para expor os resultados obtidos por este algoritmo.

Por outro lado, caso a variável *met* seja 2, no final das iterações é criado um ficheiro *.mat* designado por *mPSO*, com uma estrutura chamada *mPSO* que contém o valor e posições dos melhores globais. No menu inicial (Figura 3.14), quando se escolhe o algoritmo *MOPSO*, pode-se introduzir um intervalo que define quando uma solução é considerada dominada por outra solução. Este intervalo é introduzido em simultâneo com os parâmetros. Esta medida foi efetuada de modo a não serem perdidas algumas soluções, que embora dominadas por outras, apresentassem valores bastantes aceitáveis. Como tal, a equação (5) é alterada:

$$\max_{j \neq i}(\min_k(f_k^i - f_k^j)) \geq \varepsilon \qquad \textbf{(6)}$$

onde ε é o intervalo introduzido.

Caso a variável *met* seja 3, no final das iterações é criado um ficheiro *.mat* chamado de *PSORing*, com uma estrutura de igual nome, onde é guardada a informação do valor e posições dos melhores globais. No menu inicial (Figura 3.14), quando se escolhe o algoritmo *PSO Ring*, existe a possibilidade de escolher quantos elementos constituem uma subpopulação, se 3 elementos ou se 5 elementos. Este valor também é introduzido em simultâneo com os parâmetros.

Finalmente, caso o utilizador tenha ficado contente com os resultados obtidos, tem a possibilidade de guardar toda informação através de um extra, *save*, colocado no menu inicial, como se pode observar na Figura 3.14. Ao carregar nesse extra, é criado um ficheiro *.txt* chamado *resultado_historico*. Neste ficheiro, a nova informação é acrescentada à informação que já se encontrava no ficheiro. Para distinguir as informações guardadas, é atribuído um cabeçalho à nova informação com a respetiva data e hora da gravação. De seguida aparece a informação sobre o algoritmo utilizado, gravando a variável *met*. Posteriormente, são gravados os parâmetros utilizados na procura, onde a primeira linha contém a informação sobre a variável *ini*, isto é, se o posicionamento das partículas sofreu alguma inicialização. A segunda linha contém a informação sobre a função objetivo e, caso a variável *ini* seja igual a 1, a informação dobre a inicialização da posição das partículas. A próxima linha contém a informação sobre a dimensão das partículas, do número de partículas, do número de iterações e das constantes de inércia. A última linha dos parâmetros traduz as restrições feitas ao sistema. Finalmente, é gravada a informação referente às partículas. É gravada a posição de cada uma, bem como a sua velocidade e as posições e os custos da melhor posição pessoal e da melhor posição global.

Ao carregar nesse extra uma nova janela aparece com a hipótese de criar um outro ficheiro em simultâneo para carregamento. Esse ficheiro pode ser carregado graças a outro extra colocado no menu inicial, *load*. Porém, como já foi mencionado anteriormente, apenas os

ficheiros criados especificamente para serem carregados é que são corretamente introduzidos no programa. Para os dados serem carregados, é necessário existir um ficheiro *.txt* com o nome *resultado_carregamento*. Quando se guarda a informação, é apresentada a opção de criar um ficheiro *resultado_carregamento* em simultâneo com a introdução dos dados no ficheiro *resultado_historico*. Porém, só pode existir a informação sobre uma solução no ficheiro *resultado_carregamento* para que esta informação seja corretamente carregada.

4. Resultado experimental

De modo a avaliar o comportamento dos algoritmos de procura *PSO* implementados, é necessário definir algumas funções objetivo, que testem os algoritmos em diferentes situações.

4.1. *PSO*

O algoritmo *PSO* é testado com funções unimodais, onde o objetivo é encontrar o ótimo global. Estas funções foram originalmente introduzidas por Goldberg e Richardson de forma a testarem o *fitness shaRing* [9] e também foram utilizadas por Beasley et al. de forma a avaliarem o algoritmo sequencial *Niching* [10]. Vão-se utilizar apenas duas das cinco funções para testar o *PSO*, de forma a testar o comportamento deste algoritmo face a problemas unimodais. Como este algoritmo está implementado para encontrar mínimos e não máximos [9, 10], vai-se criar uma variação das funções referidas de forma a transformar os máximos em mínimos.

A função F_1 é uma função com um único mínimo global, mas com muitos mínimos relativos:

$$F_1(x) = \left(-e^{-2\log(2)\times\left(\frac{x-0.1}{0.8}\right)^2}\right) \times \sin^6(5\pi x).$$

O objetivo deste teste é verificar se o algoritmo tem a capacidade para ignorar os mínimos locais e encontrar o mínimo global das funções. Definiu-se um espaço de procura de $x \in [0,1]$ uma vez que F_1 tem o mínimo global no ponto $x^* = 0.1$ com o valor de $F_1(x^*) = -1$, como se pode constatar pela versão gráfica da Figura 4.27. Por outro lado, os mínimos relativos existentes neste espaço de procura estão situados nos pontos $x = 0.3$, $x = 0.5$, $x = 0.7$ e $x = 0.9$.

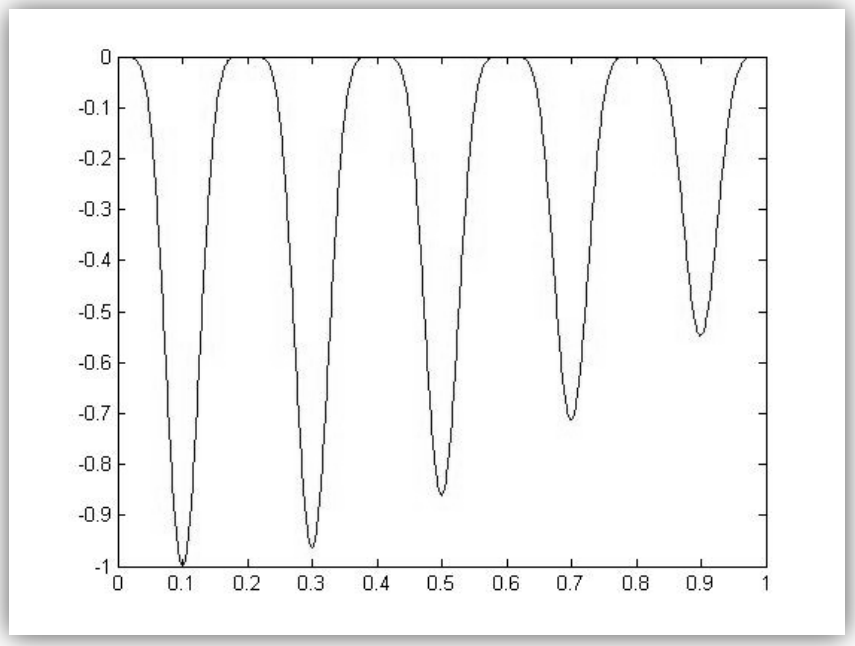


Figura 4.27 – Função de teste $\boldsymbol{F_1}$.

A função F_2 é outra função com apenas só um mínimo global, mas com vários mínimos relativos:

$$F_2(x) = (-e^{-2\log(2)\times\left(\frac{x-0.1}{0.8}\right)^2}) \times \sin^6(5\pi(x^{\frac{3}{4}} - 0.05)).$$

A diferença entre estas duas funções é a localização dos mínimos. O mínimo global de F_2 está situado em $x^* = 0.08$ com o valor de $F_2(x^*) = -1$. Os mínimos relativos estão situados a $x = 0.25$, $x = 0.45$, $x = 0.68$ e $x = 0.93$. F_2 está representada na Figura 4.28:

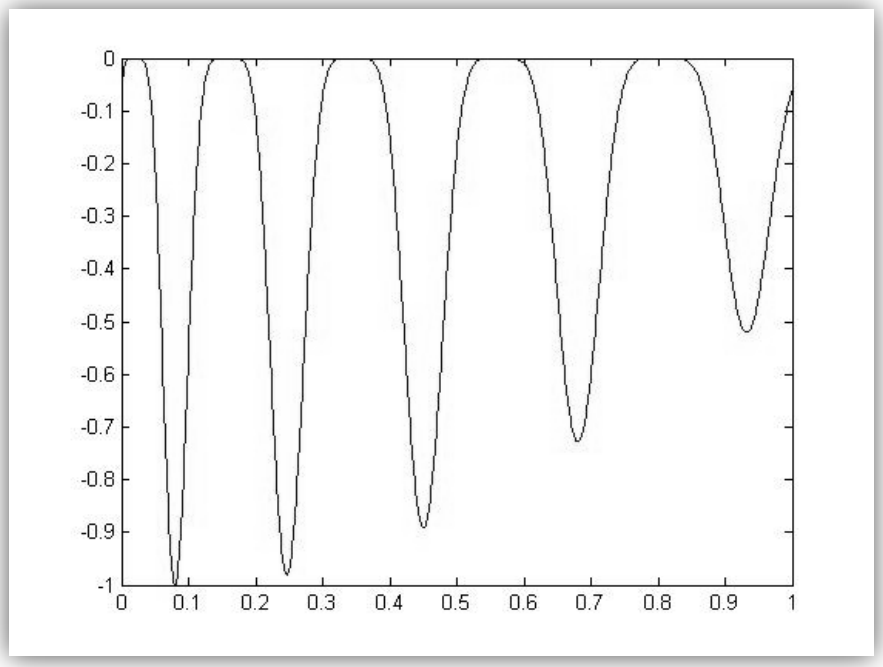


Figura 4.28 – Função de teste F_2.

Estas funções foram utilizadas mais tarde por Li, de forma a testar a substituição dos parâmetros de *Niching* pela topologia em anel [6]. Para além dessas funções foram testadas mais três funções. Neste teste vai-se fazer uma pequena modificação a essas funções de forma a se poder encontrar mínimos em vez de máximos.

A função F_3 é uma função com um mínimo global e com um mínimo relativo:

$$F_3(x) = \begin{cases} -\frac{160}{15}(15 - x) \quad para\ 0 \leq x < 15, \\ -\frac{200}{5}(x - 15)\ para\ 15 \leq x \leq 20. \end{cases}$$

Esta função encontra-se representada na Figura 4.29, onde se constata que tem o mínimo global no ponto $x^* = 20$ com o valor de $F_3(x^*) = -200$ e tem um mínimo relativo no ponto $x = 0$. Como ¾ da população vai ser inicializada entre os valores 0 e 15, torna-se difícil encontrar o mínimo global em vez do mínimo relativo vai ser uma tarefa complicada.

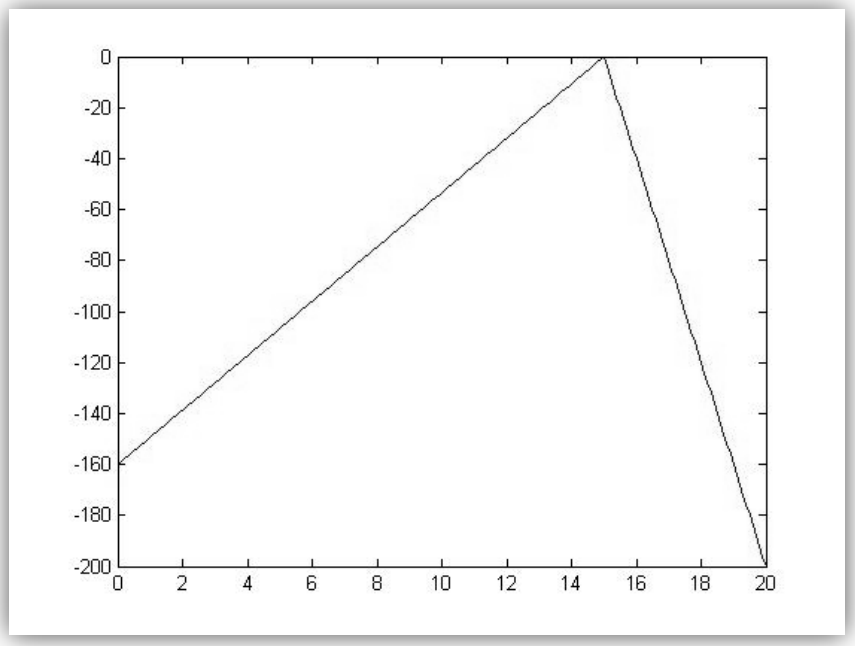


Figura 4.29 – Função de teste F_3.

A função F_4 é uma variação de F_3, com apenas um mínimo global, situado no ponto $x^* = 20$ com o valor de $F_4(x^*) = -200$, e com um mínimo relativo, situado a $x = 10$:

$$F_4(x) = \begin{cases} -\frac{160}{10}x \ \ para\ 0 \leq x < 10, \\ -\frac{160}{5}(15 - x)\ para\ 10 \leq x < 15, \\ -\frac{200}{5}(x - 15)\ para\ 15 \leq x \leq 20. \end{cases}$$

A Figura 4.30 é uma representação desta função:

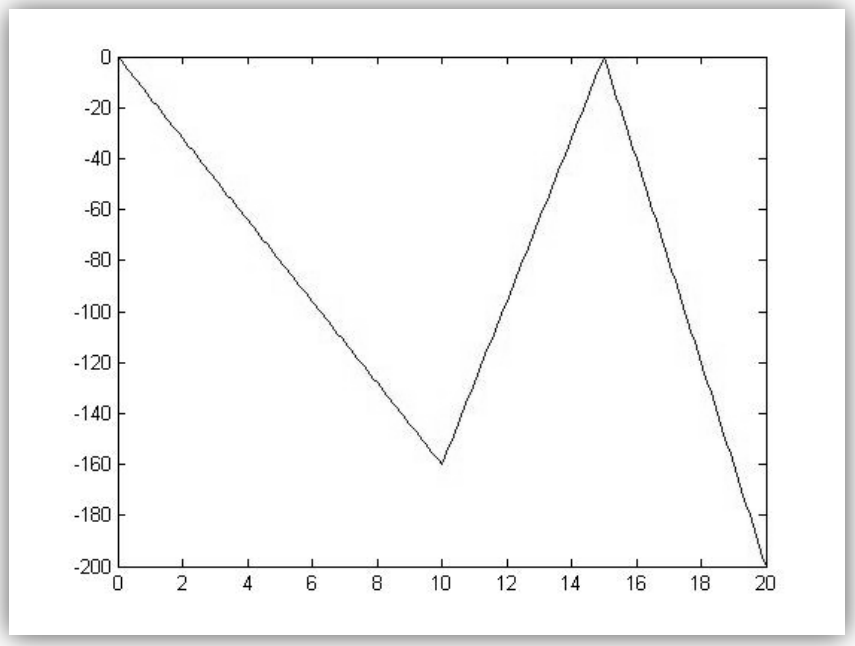


Figura 4.30 – Função teste F_4.

Por último, a função F_5 é uma função multimodal, com dois mínimos globais e três mínimos relativos, apresentando um desafio adicional ao algoritmo de forma a analisar para qual dos mínimos globais tende mais o algoritmo:

$$F_5(x) = \begin{cases} -80(2.5-x)\ para\ 0 \leq x < 2.5, \\ -64(x-2.5)\ para\ 2.5 \leq x < 5, \\ -64(7.5-x)\ para\ 5 \leq x < 7.5, \\ -28(x-7.5)\ para\ 7.5 \leq x < 12.5, \\ -28(x-7.5)\ para\ 12.5 \leq x < 17.5, \\ -32(x-17.5)\ para\ 17.5 \leq x < 22.5, \\ -32(27.5-x)\ para\ 22.5 \leq x < 27.5, \\ -80(x-27.5)\ para\ 27.5 \leq x \leq 30. \end{cases}$$

Esta função tem os mínimos globais situados nos pontos $x^* = 0$ e $x^* = 30$ com o valor de $F_5(x^*) = -200$, enquanto que os mínimos relativos estão situados nos pontos $x = 5$, $x = 12.5$ e $x = 22.5$. A representação desta função pode ser visionada na Figura 4.31:

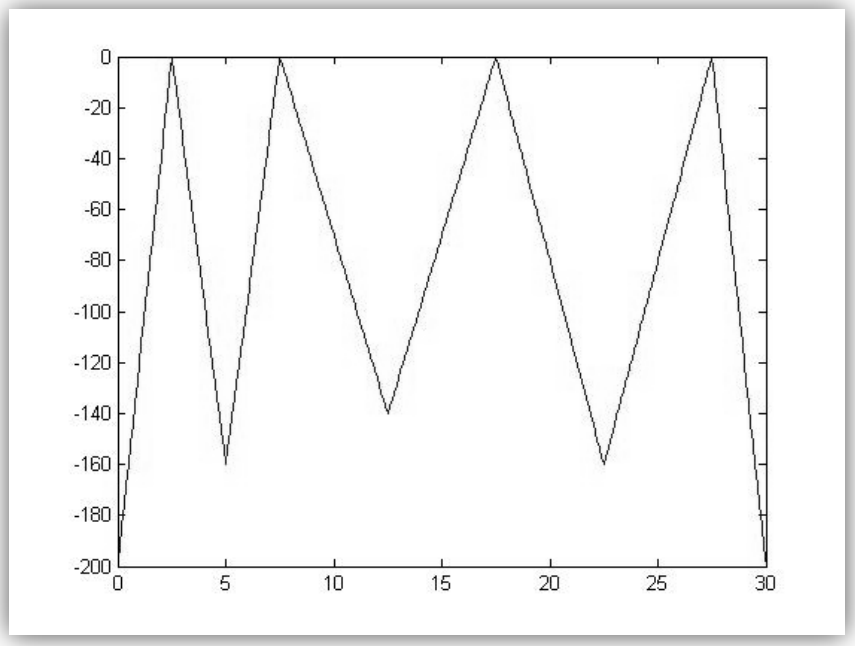


Figura 4.31 – Função teste F_5.

Para estas cinco funções, foram feitas 30 experiências com o algoritmo de procura *PSO*, onde $c_1 = c_2 = 1.2$. O peso de inercia foi definido como $\omega = 0.5$. Os valores obtidos estão presentes na Tabela 4.1:

Tabela 4.1
Resultados do algoritmo *PSO* às funções teste F_1, F_2, F_3, F_4 e F_5.

Função	Nº Iterações	Nº Partículas	Média	Desvio padrão	Conv. (%)	Tempo por experiência (seg.)	Média	Desvio Padrão	Conv. (%)
F_1	100	15	1.53E-01	1.26E-01	80	0.29	9.12E-02 [17]	6.43E-02 [17]	93 [17]
		30	1.20E-01	8.15E-02	93	0.50			
		60	1.00E-01	3.75E-10	100	0.56			
	200	15	1.53E-01	1.37E-01	80	0.40			
		30	1.13E-01	5.16E-02	93	0.61			
		60	1.00E-01	3.55E-10	100	0.78			
F_2	100	15	1.97E-01	1.82E-01	47	0.30	8.07E-02 [17]	6.68E-02 [17]	93 [17]
		30	1.02E-01	6.08E-02	87	0.52			
		60	7.97E-02	3.00E-04	100	0.60			
	200	15	1.56E-01	1.42E-01	67	0.45			
		30	9.64E-02	5.27E-02	90	0.68			
		60	7.97E-02	3.00E-04	100	0.86			

F_3	100	15	1.27E01	1.40E01	63	0.36	Sem Dados	Sem Dados	Sem Dados
		30	1.60E01	4.00E00	80	0.59			
		60	2.00E01	0.00E00	100	0.65			
	200	15	1.47E01	5.33E00	73	0.45			
		30	1.67E01	3.33E00	83	0.68			
		60	2.00E01	0.00E00	100	0.89			
F_4	100	15	1.60E01	4.00E00	60	0.40	Sem Dados	Sem Dados	Sem Dados
		30	1.87E01	1.33E00	87	0.62			
		60	1.97E01	3.33E-01	97	0.70			
	200	15	1.70E01	3.00E00	70	0.50			
		30	1.90E01	1.00E00	90	0.75			
		60	2.00E01	0.00E00	100	0.93			
F_5	100	15	2.17E00	2.17E00	77	0.43	Sem Dados	Sem Dados	Sem Dados
		30	1.00E00	1.00E00	90	0.65			
		60	0.00E00	0.00E00	100	0.79			
	200	15	1.00E00	1.00E00	80	0.53			
		30	1.67E-01	1.67E-01	97	0.79			
		60	0.00E00	0.00E00	100	0.96			

Como se pode observar na Tabela 4.1, na função F_1, a variação do número de iterações do *PSO* não afecta a convergência com que o algoritmo caminha para a solução ótima correta. Porém, ao aumentar o número de partículas, a convergência do algoritmo *PSO* também aumenta. Pode-se também observar que os resultados obtidos nesta função teste são bastante aceitáveis, comparados com os obtidos por Brits, Engelbrecht e van den Bergh [17]. Por outro lado, nas funções F_2, F_3 e F_4 a diminuição de iterações do *PSO*, vai diminuir também a convergência do algoritmo, tendo mais influência, essa diminuição, quando estamos perante um enxame mais reduzido. Se se aumentar o número de partículas, a convergência também vai aumentar. A função de teste F_2 apresenta valores muito semelhantes aos obtidos por Brits, Engelbrecht e van den Bergh [17]. No caso especial de F_5, onde existem dois mínimos globais, o algoritmo convergiu para o ponto $x^* = 0$ uma vez que este ponto tem um mínimo relativo bastante perto. Quantas mais partículas e iterações tiverem o enxame, maior será a convergência

4.2. *PSO Ring*

O algoritmo *PSO* recorrendo à topologia em anel tem o objetivo de encontrar todos os mínimos, sejam eles locais ou globais. Como tal, vai-se recorrer a funções de teste unimodais e multimodais.

As funções F_3, F_4 e F_5 são consideradas funções ilusórias, uma vez que a existência de mínimos relativos podem atrair a população, movendo-a para longe do mínimo global. Desta

forma, pode-se testar a habilidade do algoritmo em contornar os mínimos relativos e encontrar a verdadeira solução ótima.

As funções F_1 e F_2 são ideais para avaliar o processo de encontrar todos os ótimos, quer sejam globais ou locais.

Para estas cinco funções, foram feitas 30 experiências com o algoritmo de procura *PSO* recorrendo à topologia em anel, onde $c_1 = c_2 = 1.2$. O peso de inercia foi definido como $\omega = 0.5$. Os valores obtidos estão presentes na Tabela 4.2:

Tabela 4.2
Resultados do algoritmo *PSO* recorrendo à topologia em anel às funções F_1, F_2, F_3, F_4 e F_5.

Função	Nº Iterações	Nº Partícula	Subpopulação	Convergência (%)	Tempo por experiência (seg.)	Convergência (%)
F_1	100	15	3	93	0.64	98 [6]
			5	97	0.66	
		30	3	100	0.72	
			5	100	0.76	
		60	3	100	0.81	
			5	100	0.83	
	200	15	3	93	0.65	
			5	100	0.69	
		30	3	100	0.73	
			5	100	0.78	
		60	3	100	0.83	
			5	100	0.88	
F_2	100	15	3	90	0.65	100 [6]
			5	90	0.68	
		30	3	100	0.73	
			5	100	0.78	
		60	3	100	0.85	
			5	100	0.86	
	200	15	3	100	0.68	
			5	100	0.71	
		30	3	100	0.76	
			5	100	0.79	
		60	3	100	0.89	
			5	100	0.93	
F_3	100	15	3	77	0.62	98 [6]
			5	87	0.70	
		30	3	97	0.82	
			5	97	0.85	
		60	3	100	0.97	

			5	100	1.01	
	200	15	3	77	0.65	
			5	93	0.73	
		30	3	97	0.85	
			5	97	0.89	
		60	3	100	1.03	
			5	100	1.10	
F_4	100	15	3	73	0.75	100 [6]
			5	77	0.82	
		30	3	90	0.83	
			5	100	0.90	
		60	3	100	1.02	
			5	100	1.05	
	200	15	3	93	0.78	
			5	97	0.81	
		30	3	100	0.93	
			5	100	0.98	
		60	3	100	1.05	
			5	100	1.12	
F_5	100	15	3	87	0.77	100 [6]
			5	93	0.83	
		30	3	100	0.86	
			5	100	0.95	
		60	3	100	1.09	
			5	100	1.12	
	200	15	3	93	0.80	
			5	97	0.86	
		30	3	100	0.95	
			5	100	1.02	
		60	3	100	1.15	
			5	100	1.19	

A Tabela 4.3 mostra quantos ótimos, globais ou locais, o algoritmo conseguiu encontrar:

Tabela 4.3
Número de ótimos encontrados pelo algoritmo *PSO* recorrendo à topologia em anel nas funções $\boldsymbol{F_1}$, $\boldsymbol{F_2}$, $\boldsymbol{F_3}$, $\boldsymbol{F_4}$ e $\boldsymbol{F_5}$.

Função	Nº Iterações	Nº Partícula	Subpopulação	Tempo por experiência (seg.)	Média do Nº ótimos encontrado
F_1	100	15	3	0.64	1.13
			5	0.66	1.00

		30	3	0.72	1.23
			5	0.76	1.07
		60	3	0.81	1.90
			5	0.83	1.10
	200	15	3	0.65	1.00
			5	0.69	1.00
		30	3	0.73	1.07
			5	0.78	1.00
		60	3	0.83	1.27
			5	0.88	1.00
F_2	100	15	3	0.65	1.53
			5	0.68	1.00
		30	3	0.73	1.93
			5	0.78	1.20
		60	3	0.85	2.07
			5	0.86	1.30
	200	15	3	0.68	1.40
			5	0.71	1.00
		30	3	0.76	1.53
			5	0.79	1.13
		60	3	0.89	1.90
			5	0.93	1.30
F_3	100	15	3	0.62	1.43
			5	0.70	1.00
		30	3	0.82	1.87
			5	0.85	1.13
		60	3	0.97	2.00
			5	1.01	1.20
	200	15	3	0.65	1.13
			5	0.73	1.00
		30	3	0.85	1.30
			5	0.89	1.07
		60	3	1.03	1.53
			5	1.10	1.20
F_4	100	15	3	0.75	1.10
			5	0.82	1.00
		30	3	0.83	1.57
			5	0.90	1.00
		60	3	1.02	1.77
			5	1.05	1.13
	200	15	3	0.78	1.00

			5	0.81	1.00
		30	3	0.93	1.10
			5	0.98	1.00
		60	3	1.05	1.13
			5	1.12	1.07
F_5	100	15	3	0.77	1.90
			5	0.83	1.30
		30	3	0.86	2.90
			5	0.95	1.87
		60	3	1.09	3.50
			5	1.12	2.20
	200	15	3	0.80	1.80
			5	0.86	1.20
		30	3	0.95	2.33
			5	1.02	1.73
		60	3	1.15	3.33
			5	1.19	2.10

Como se pode observar na Tabela 4.2, nas funções F_1 e F_2 atinge-se uma maior convergência à medida que se aumenta, quer o número de partículas, quer o número de iterações, quer o número de partículas agrupadas por subpopulações. Os valores da convergência nestas funções de teste são bastante semelhantes aos valores obtidos por Li[6].

Em funções unimodais mas com mínimos relativos que possam causar ilusão em relação à solução ótima, casos de F_3 e F_4, quando se tem um número de partículas mais reduzido, obtém-se melhores convergências com agrupamentos de 5 partículas por subpopulação que com 3. Quanto maior for o número de partículas, menor influência tem, na convergência, o número de partículas por agrupamento. Por outro lado, a convergência também aumenta à medida que o número de iterações ou o número de partículas vão aumentando. A melhor convergência obtida nestas funções teste são semelhantes às convergências obtidas por Li [6].

Em funções multimodais e com vários mínimos relativos, caso de F_5, os resultados são semelhantes aos obtidos nas outras funções. Quando se tem um menor número de partículas, obtém-se melhores resultados de convergência com agrupamentos de 5 partículas por subpopulação que com 3. A convergência também aumenta à medida que o número de iterações ou o número de partículas aumenta. Chegou-se à mesma convergência que Li [6] nesta função teste.

Através da Tabela 4.3, pode-se concluir que para procuras onde se interessa obter nos resultados todos os ótimos, quer globais como locais, é aconselhável a utilização de um grande número de partículas divididas em subpopulações de 3 partículas durante poucas iterações. Se, pelo contrário, o objetivo é obter apenas uma solução final, é aconselhável a utilização de poucas partículas divididas em subpopulações de 5 partículas durante muitas iterações.

4.3. *MOPSO*

O algoritmo *MOPSO* tem o objetivo de identificar a *Fronteira de Pareto* existente entre duas ou mais funções objetivos. Para avaliar este algoritmo e testar se encontra a verdadeira *Fronteira de Pareto*, vai-se recorrer à função com dois objetivos e de uma dimensão introduzida por Schaffer [11] e depois utilizada por Balling [5]:

Minimize: $F_6(x) = x^2$

Minimize: $F_7(x) = (x-2)^2$

A variável $x \in [-100; 100]$, $\varepsilon = 0$, distribui-se 30 partículas pela região de procura e após 100 iterações com $c_1 = c_2 = 1.2$ e $\omega = 0.5$ chegou-se à *Fronteira de Pareto* da Figura 4.32:

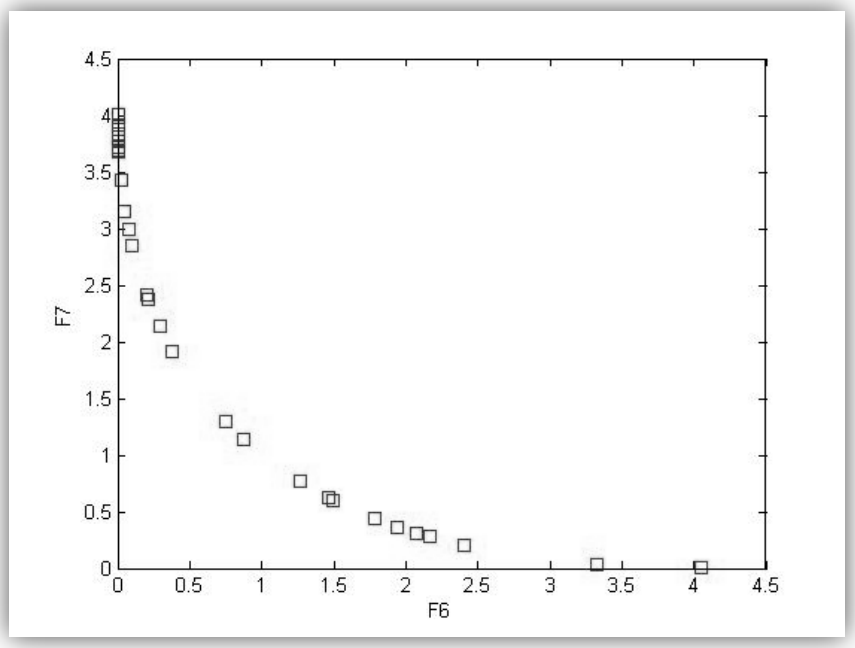


Figura 4.32 – Primeiro teste da *Fronteira de Pareto* com $\boldsymbol{\varepsilon = 0}$.

Como se pode observar na Figura 4.32, a *Fronteira de Pareto* começa no ponto $(F_6, F_7) = (0,4)$ e termina no ponto $(F_6, F_7) = (4,0)$. O algoritmo apresentou uma *Fronteira de Pareto* bastante aceitável e semelhante à obtida por Balling [5].

Se se aumentar o intervalo ε para 0.1 obtém-se a *Fronteira de Pareto* da Figura 4.33:

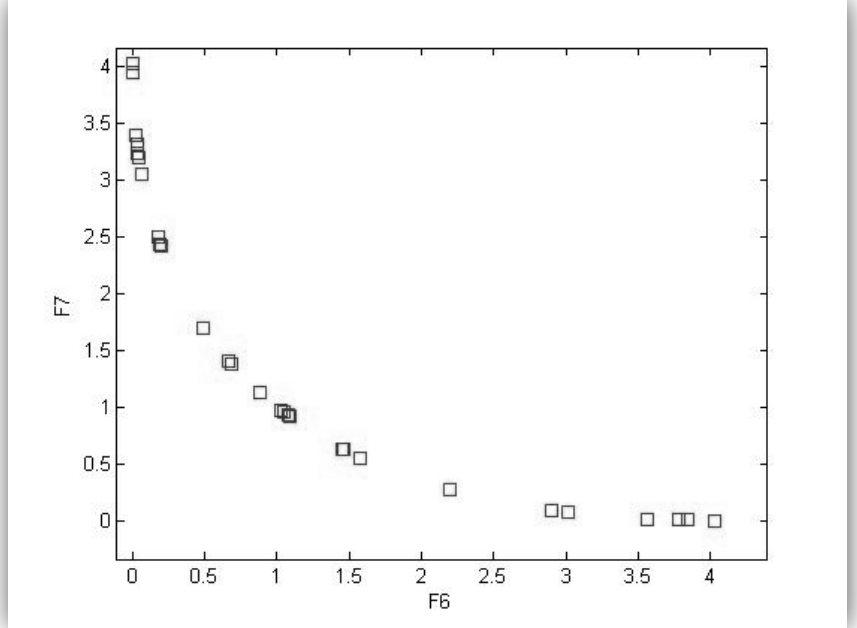


Figura 4.33 – Primeiro teste da *Fronteira de Pareto* com $\boldsymbol{\varepsilon = 0.1}$.

Pode-se observar que não existe grande alteração entre as duas Fronteiras de Pareto. Assim desta forma pode-se acrescentar algumas soluções que, mesmo dominadas por outras soluções, apresentem um valor considerado aceitável. Porém o valor de ε tem que se ajustar consoante a função objetivo. Neste caso como se trata de uma função objetivo com soluções na ordem das unidades, variar uma décima não influenciará a resposta final.

Vai-se aproveitar as funções $\boldsymbol{F_1}$ e $\boldsymbol{F_2}$ para testar este algoritmo. Assim, a nova função objetivo é:

Minimize: F_1

Minimize: F_2

A variável $x \in [0; 1]$, $\varepsilon = 0$, distribui-se 30 partículas pela região de procura e após 100 iterações com $c_1 = c_2 = 1.2$ e $\omega = 0.5$ chegou-se à *Fronteira de Pareto* da Figura 4.34:

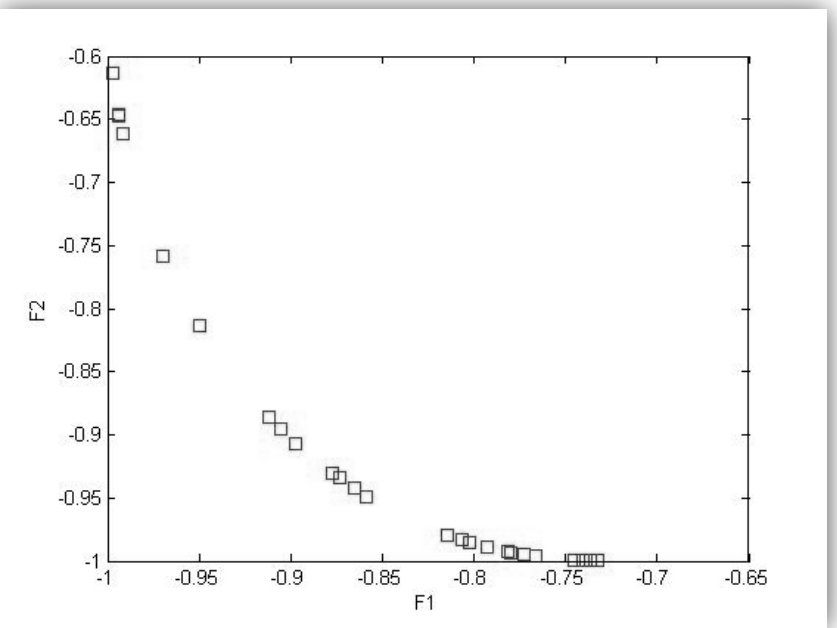


Figura 4.34 – Segundo teste da *Fronteira de Pareto* com $\boldsymbol{\varepsilon} = \mathbf{0}$.

O algoritmo apresentou novamente uma *Fronteira de Pareto* bastante aceitável na Figura 4.34. Pode-se observar que a *Fronteira de Pareto* começa em $(F_1, F_2) = (-1, -0.61)$ e termina em $(F_1, F_2) = (-0.78, -1)$.

4.4. *PSO Ring* Multi-objetivo

Como se mencionou anteriormente, foi elaborado um algoritmo de otimização de problemas multi-objetivos inovador, fundindo o algoritmo *PSO Ring* com a técnica *Maxmin*.

Para testar este algoritmo, vai-se recorrer às mesmas funções de teste utilizadas no algoritmo *MOPSO*. Primeiro vai-se testar a seguinte função objetivo:

Minimize: $F_6(x) = x^2$

Minimize: $F_7(x) = (x-2)^2$

A variável $x \in [-100; 100]$, distribui-se 30 partículas pela região de procura, com subpopulações constituídas por três partículas, e após 500 iterações, pois este algoritmo é significativamente mais lento que o algoritmo *MOPSO*, com $c_1 = c_2 = 1.2$ e $\omega = 0.5$ chegou-se à *Fronteira de Pareto* da Figura 4.35:

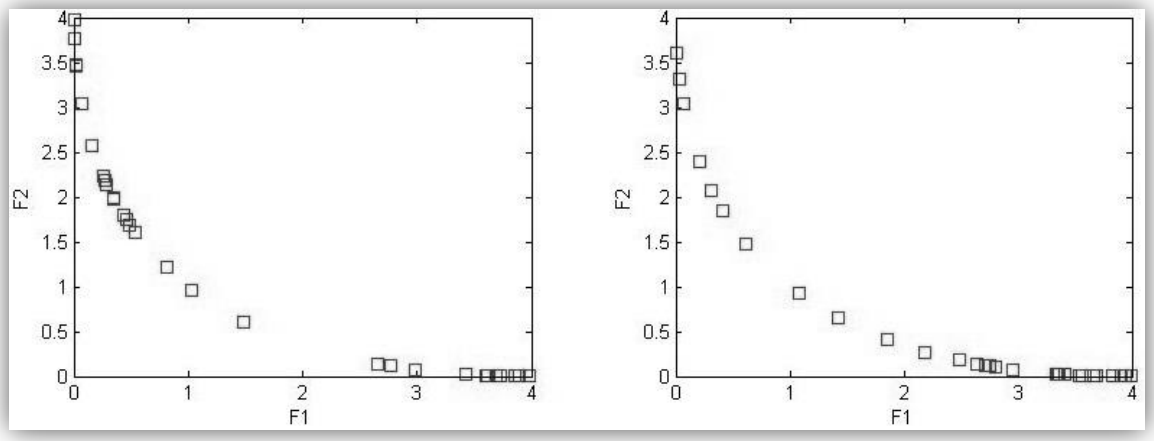


Figura 4.35 – Primeira comparação entre as Fronteiras de Pareto obtidas pelo algoritmo *MOPSO*, à esquerda, e o algoritmo *PSO Ring*, à direita.

Como se pode observar na Figura 4.35, a *Fronteira de Pareto* obtida pelo algoritmo *PSO Ring* é semelhante ao obtido pelo algoritmo *MOPSO*. Neste caso, não existem ótimos locais que diferenciem uma Fronteira da outra.

De seguida, vai-se testar a seguinte função objetivo:

Minimize: F_1

Minimize: F_2

A variável $x \in [0; 1]$, distribui-se 30 partículas pela região de procura, com subpopulações constituídas por três partículas, e após 500 iterações com $c_1 = c_2 = 1.2$ e $\omega = 0.5$ chegou-se à *Fronteira de Pareto* da Figura 4.36:

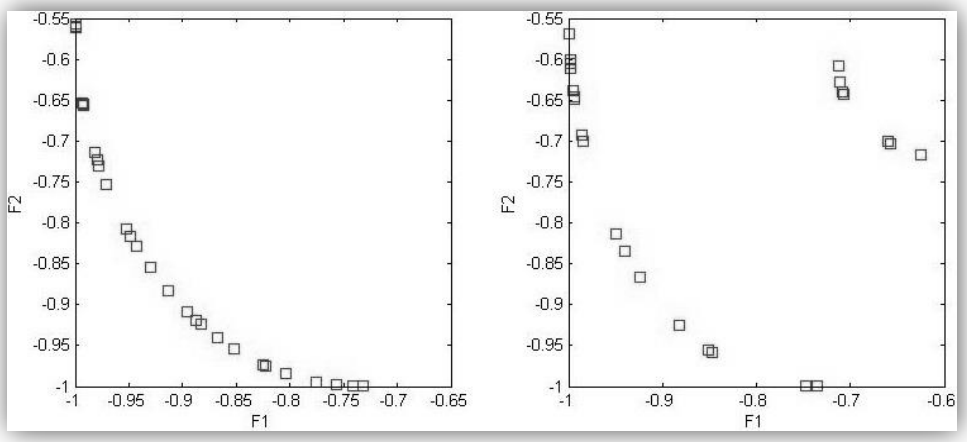


Figura 4.36 - Segunda comparação entre as Fronteiras de Pareto obtidas pelo algoritmo *MOPSO*, à esquerda, e o algoritmo *PSO Ring*, à direita.

Pode-se observar na Figura 4.36 que nesta função teste, o algoritmo *PSO Ring* encontrou alguns ótimos locais que tinham sido descartados pelo algoritmo *MOPSO*. Porém, estes ótimos locais são muito dominados pelos ótimos globais, o que não constituem soluções plausíveis para otimização do problema.

Finalmente, vai-se testar este algoritmo com a função objetivo utilizada por Hassan, Cohanim e Weck [7]:

Minimize: $F_8(x) = 100 \times (x_2 - x_1^2)^2 + (1 - x_1)^2$

Minimize: $F_9(x) = x_1^2 + x_2^2 + 25 \times (\sin^2 x_1 + \sin^2 x_2)$

A variável $x \in [0; 1]$, distribui-se 500 partículas pela região de procura, com subpopulações constituídas por três partículas, e após 1500 iterações para o *PSO Ring* e apenas 500 iterações para o *MOPSO*, pois o algoritmo *MOPSO* tem uma convergência mais rápida que o algoritmo *PSO Ring*, e com $c_1 = c_2 = 1.2$ e $\omega = 0.5$ chegou-se à *Fronteira de Pareto* da Figura 4.37:

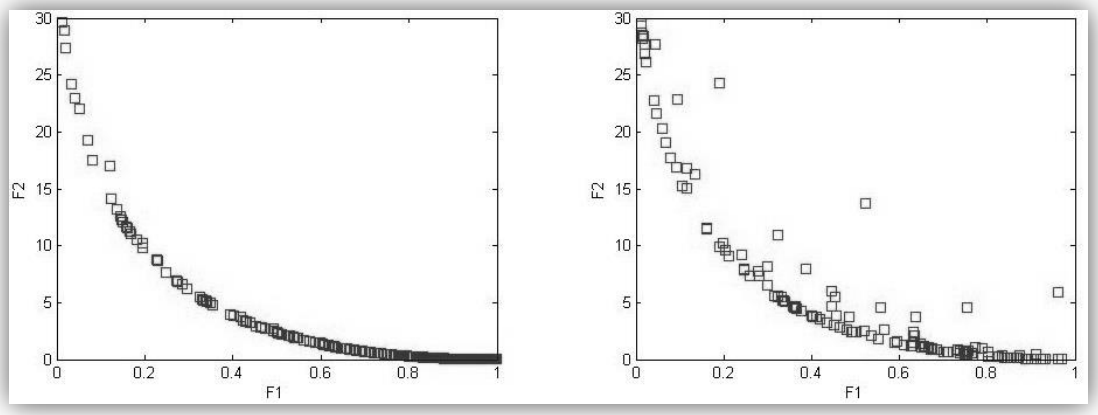


Figura 4.37 - Terceira comparação entre as Fronteiras de Pareto obtidas pelo algoritmo *MOPSO*, à esquerda, e o algoritmo *PSO Ring*, à direita.

Na Figura 4.37 é possível verificar que o algoritmo *PSO Ring* encontra alguns ótimos locais. Esses ótimos locais são pouco dominados pelos ótimos globais, tornando-os soluções admissíveis para o problema. Ao não descartar os ótimos locais, o algoritmo *PSO Ring* oferece uma maior liberdade na escolha da solução ótima através da *Fronteira de Pareto*.

Na Figura 4.38 é possível observar um exemplo entre a diferença do espaço de projeto e o espaço de critérios. Neste caso, as duas partículas, que no espaço de critérios se encontram junto uma da outra, no espaço de projeto encontram-se, consideravelmente, longe uma da outra.

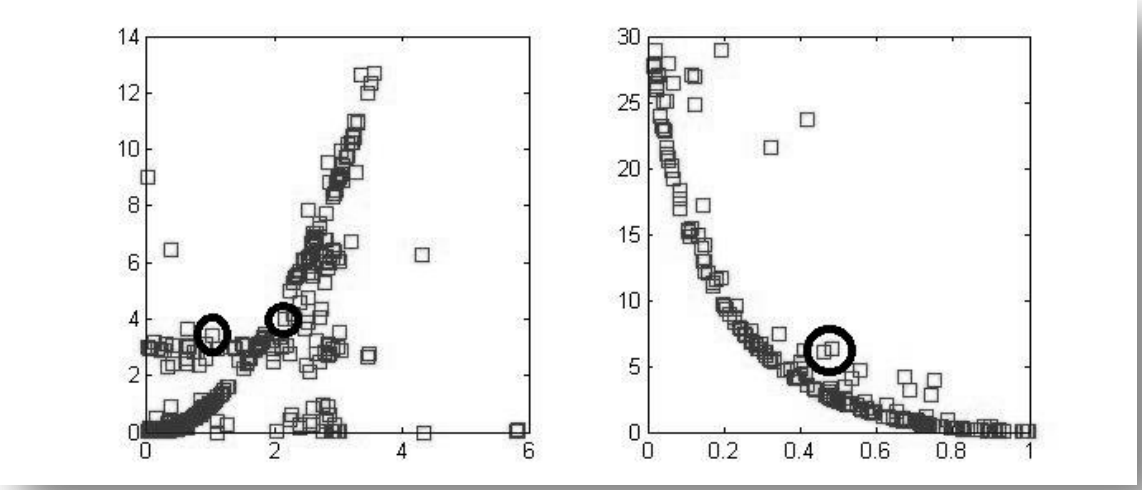


Figura 4.38 – Comparação do resultado obtido pelo algoritmo *PSO Ring* no espaço de projeto, à esquerda, com o resultado obtido no espaço de critérios, à direita.

4.5. Dimensionamento de um controlador PID

Vai-se analisar, como exemplo, o sistema de controlo presente na Figura 3.17, onde a função transferência é igual a $F(s) = \frac{1}{(s+2)^2}$.

A Instalação F(s) está em série com um controlador C(s) e ambos estão num anel fechado. O controlador C(s), tal como está representado na Figura 3.17, é um controlador PID. Um controlador PID possui a estrutura presente na Figura 4.39, com uma componente integral (T_i), uma componente derivativa (T_d) e uma componente proporcional (K_p).

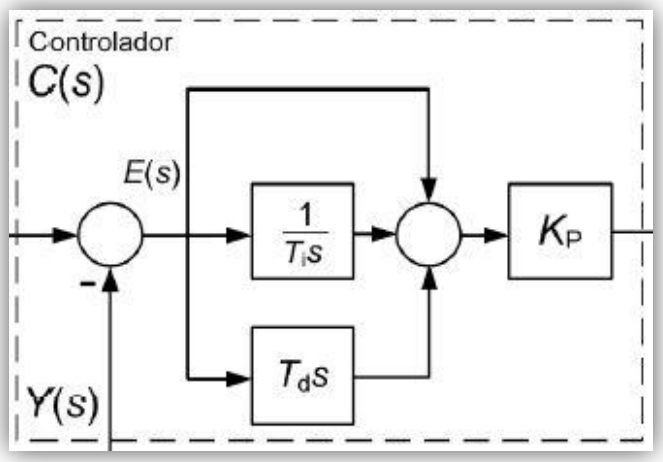


Figura 4.39 – Esquema de blocos do controlador PID.

As componentes do controlador PID vão ser dimensionadas, de forma a se obter o resultado pretendido. Vão existir alguns objetivos que se pretende atingir:

- T_i, T_d e K_p positivos;
- Sobrelevação não superior a 10%;
- Tempo de subida não superior a 5 segundos;
- Tempo de estabelecimento não superior a 5 segundos.

Em primeiro lugar, vai-se utilizar o algoritmo *PSO* para se obter os resultados, em cima apresentados. Distribui-se 30 partículas pela região de procura, dimensão igual a 3, e após 100 iterações com $c_1 = c_2 = 1.2$ e $\omega = 0.8$ chegou-se ao resultado presente na Figura 4.40:

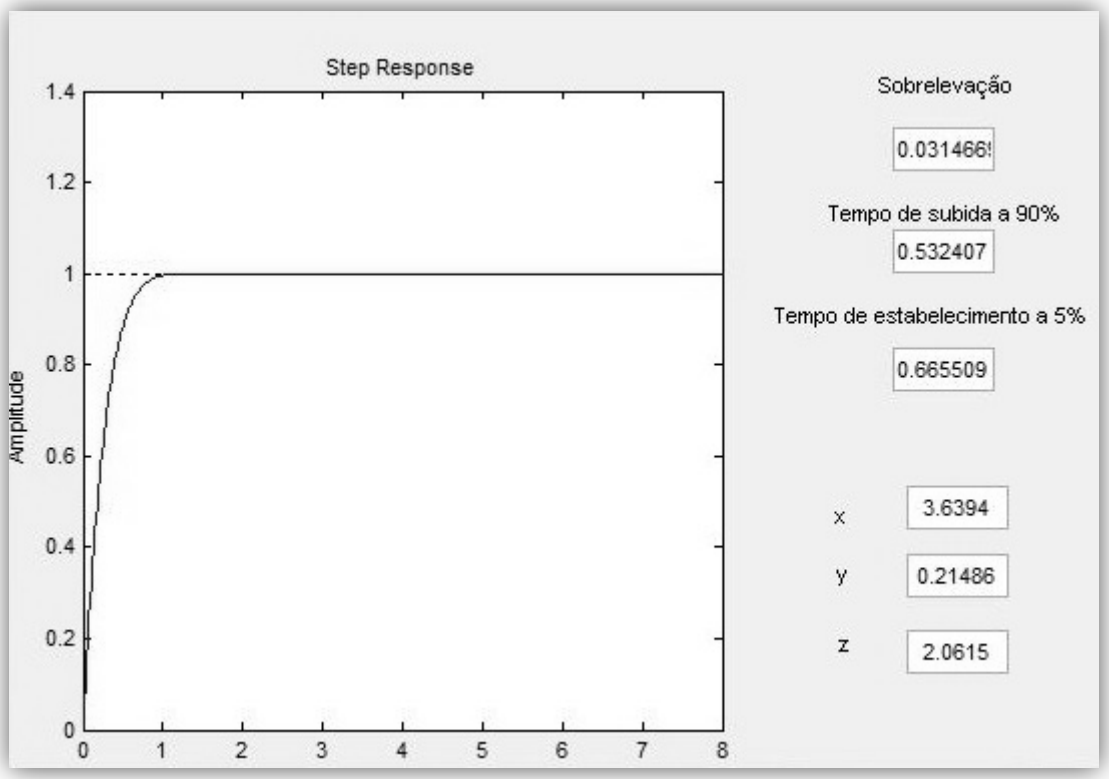


Figura 4.40 – Resposta do sistema com controlador PID, dimensionado através do algoritmo *PSO*, ao degrau unitário.

Como se pode observar na Figura 4.40, as restrições do problema são respeitadas:

- Função objetivo:
  - Sobrelevação + Tempo de subida + Tempo de estabelecimento;
- Resultado obtido:
  - $T_i = 3{,}6394$, $T_d = 0{,}21486$ e $K_p = 2{,}0615$;
  - Sobrelevação de 0,03%;
  - Tempo de subida 0,53 segundos;

- Tempo de estabelecimento 0,66 segundos.

Como se pode constatar, todos os objetivos propostos previamente, foram atingidos. De seguida recorreu-se ao algoritmo *MOPSO* para resolver o mesmo problema. Após se distribuir 30 partículas pela região de procura, com dimensão igual a 3, e após 1000 iterações com $c_1 = c_2 = 1.2$ e $\omega = 0.8$ chegou-se à *Fronteira de Pareto* presente na Figura 4.41.

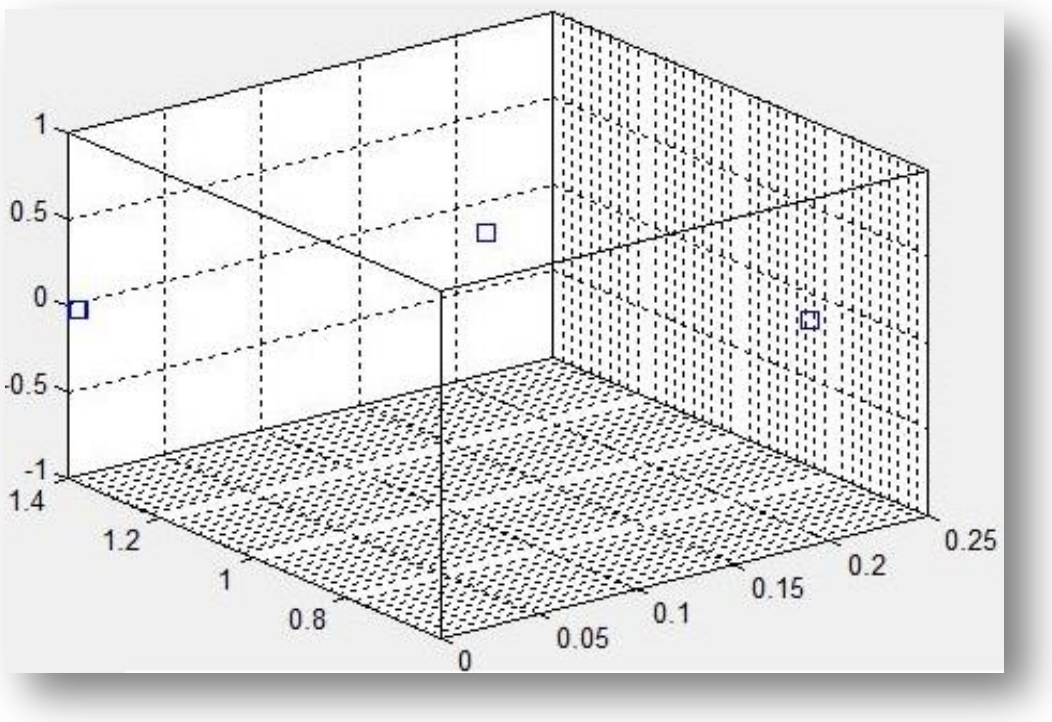


Figura 4.41 – *Fronteira de Pareto* obtida pelo algoritmo *MOPSO* do sistema com controlador PID.

De seguida é preciso escolher uma resposta. Esse processo é feito pelo utilizador. Escolhi a solução presente na Figura 4.42 como sendo a solução ótima a ser implementada.

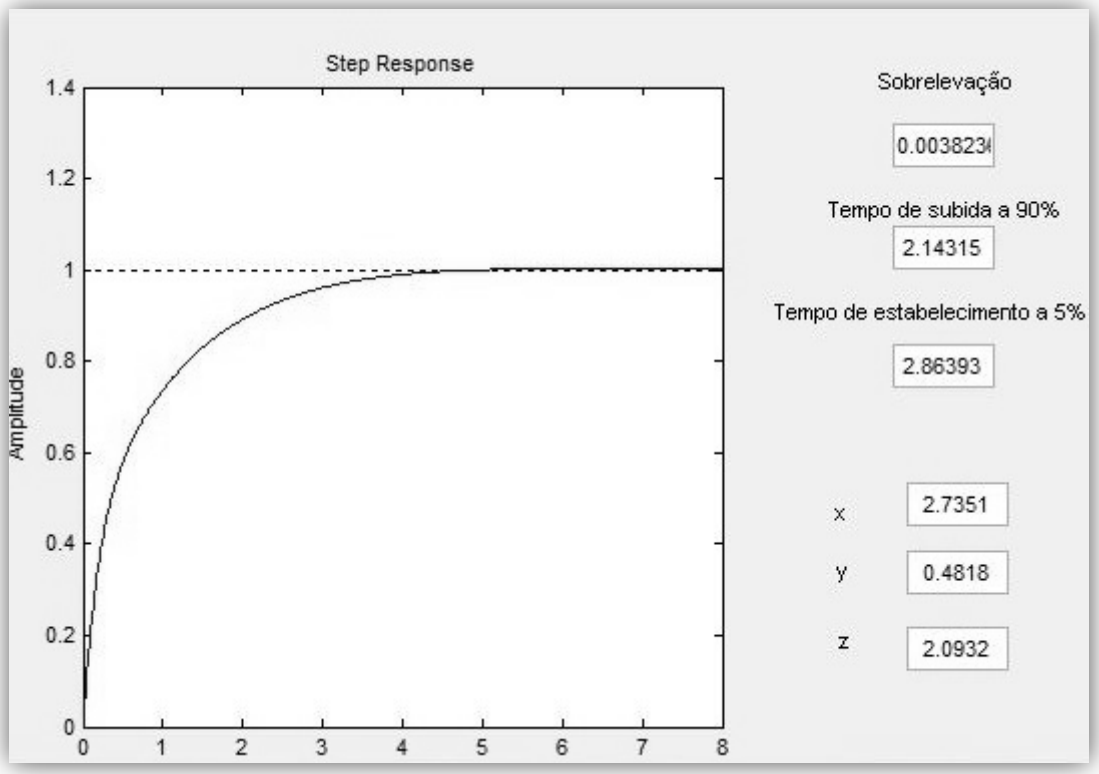


Figura 4.42 - Resposta do sistema com controlador PID, dimensionado através do algoritmo *MOPSO*, ao degrau unitário.

Como se pode observar na Figura 4.42, as restrições do problema são respeitadas:

- Função objetivo, otimização em simultâneo:
  - Sobrelevação;
  - Tempo de subida;
  - Tempo de estabelecimento;
- Resultado obtido:
  - $T_i = 2{,}7351$, $T_d = 0{,}4818$ e $K_p = 2{,}0932$;
  - Sobrelevação de 0,04%;
  - Tempo de subida 2,14 segundos;
  - Tempo de estabelecimento 2,86 segundos.

Como se pode constatar, todos os objetivos propostos previamente, foram atingidos. De seguida recorreu-se ao algoritmo *PSO Ring* para resolver o mesmo problema. Após se distribuir 30 partículas pela região de procura, com subpopulações constituídas por três partículas de dimensão igual a 3, e após 100 iterações com $c_1 = c_2 = 1.2$ e $\omega = 0.8$ chegou-se ao resultado presente na Figura 4.43.

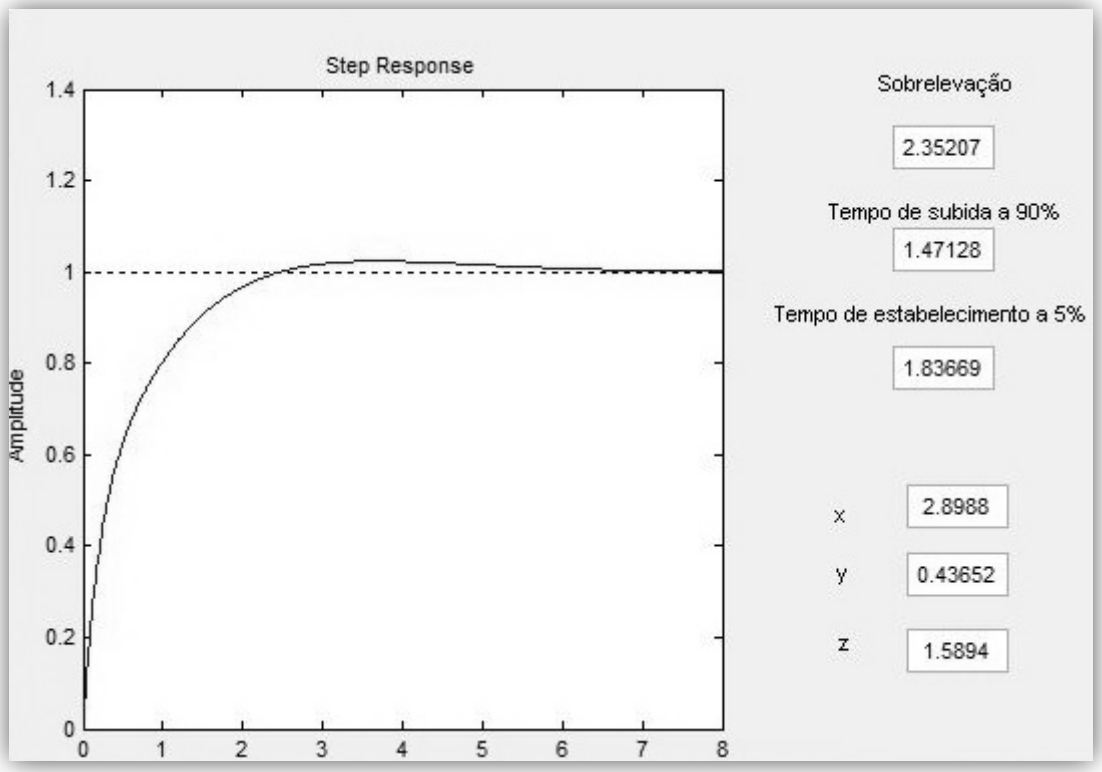


Figura 4.43 - Resposta do sistema com controlador PID, dimensionado através do algoritmo *PSO Ring*, ao degrau unitário.

Como se pode observar na Figura 4.43, as restrições do problema são respeitadas:

- Função objetivo, otimização em simultâneo:
  - Sobrelevação;
  - Tempo de subida;
  - Tempo de estabelecimento;
- Resultado obtido:
  - $T_i = 2{,}8988$, $T_d = 0{,}43652$ e $K_p = 1{,}5894$;
  - Sobrelevação de 2,35%;
  - Tempo de subida 1,47 segundos;
  - Tempo de estabelecimento 1,84 segundos.

Como se pode constatar, todos os objetivos propostos previamente, foram atingidos. O algoritmo que obteve um resultado com menor sobrelevação e, ao mesmo tempo, com um tempo de subida e de estabelecimento mais rápido foi o algoritmo *PSO*.

Por último, vai-se testar o algoritmo inovador *PSO* em anel integrado com a técnica *Maxmin*. Após se distribuir 30 partículas pela região de procura, com dimensão igual a 3, e

após 1000 iterações com $c_1 = c_2 = 1.2$ e $\omega = 0.8$ chegou-se à *Fronteira de Pareto* presente na Figura 4.44.

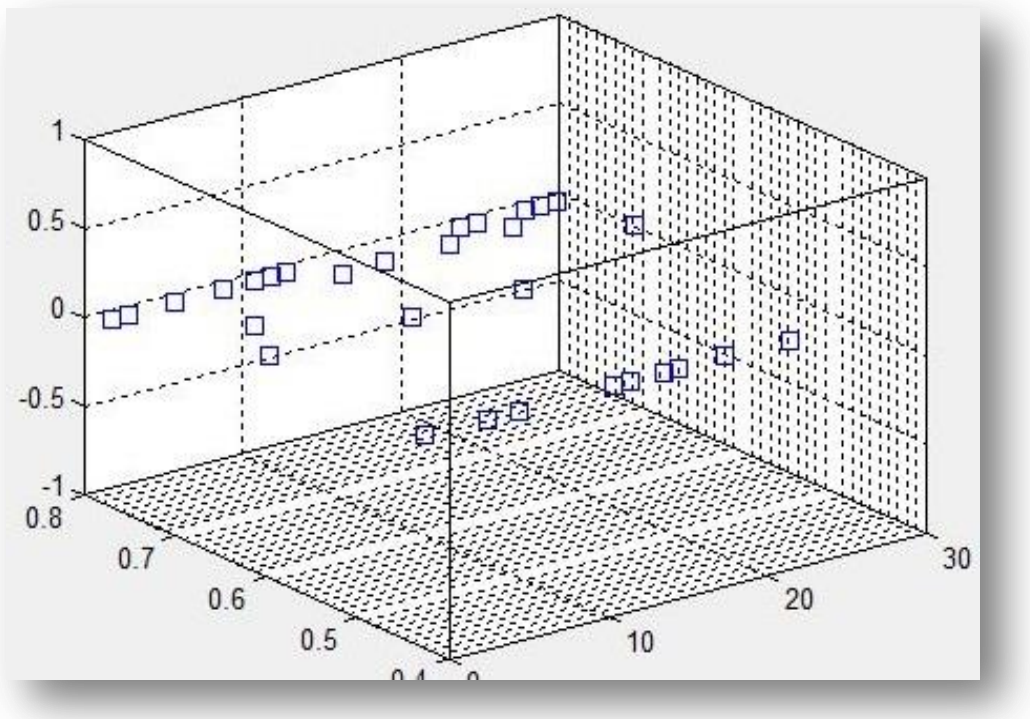


Figura 4.44 – *Fronteira de Pareto* obtida pelo algoritmo *PSO Ring* integrado com a técnica *Maxmin* do sistema com controlador PID.

De seguida é preciso escolher uma resposta. Esse processo é feito pelo utilizador. Escolhi a solução presente na Figura 4.45 como sendo a solução ótima a ser implementada.

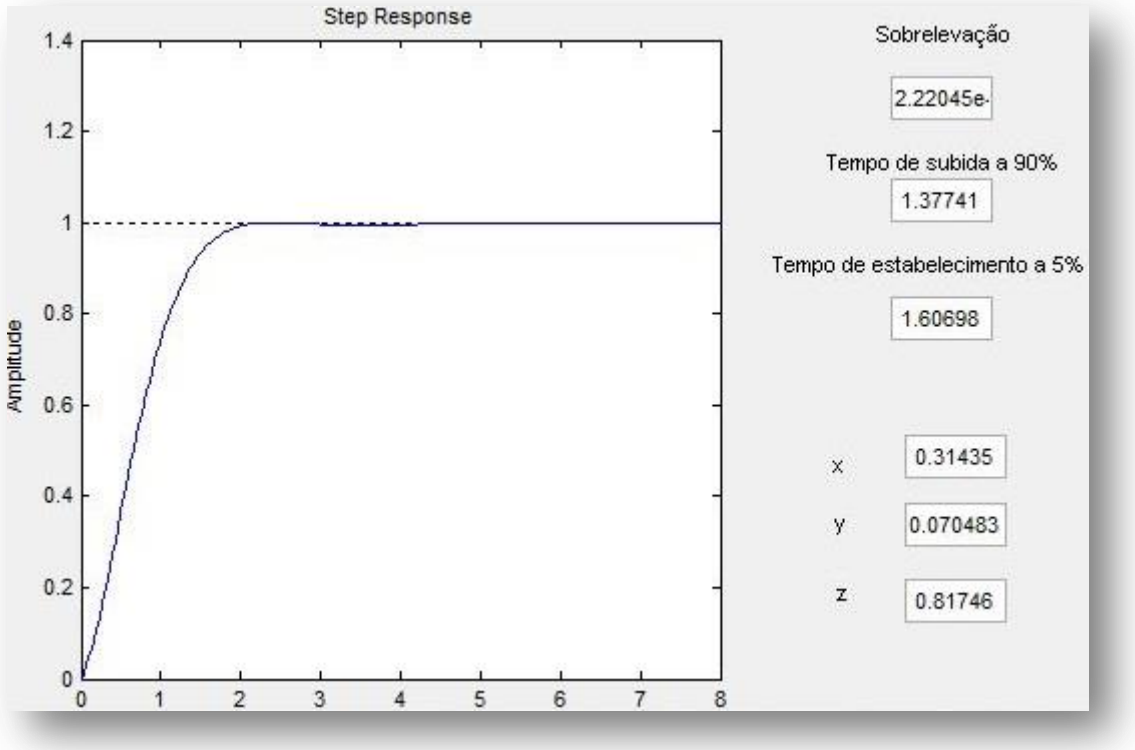


Figura 4.45 - Resposta do sistema com controlador PID, dimensionado através do algoritmo *PSO Ring* integrado com a técnica *Maxmin*, ao degrau unitário.

Como se pode observar na Figura 4.45, as restrições do problema são respeitadas:

- Função objetivo, otimização em simultâneo:
  - Sobrelevação;
  - Tempo de subida;
  - Tempo de estabelecimento;
- Resultado obtido:
  - $T_i = 0{,}31435$, $T_d = 0{,}070483$ e $K_p = 0{,}81746$;
  - Sobrelevação de 0%;
  - Tempo de subida 1,38 segundos;
  - Tempo de estabelecimento 1,61 segundos.

Como se pode constatar, todos os objetivos propostos previamente, foram atingidos. Comparando este algoritmo com o *MOPSO*, destaca-se a maior liberdade de escolha oferecida ao utilizador na solução final, bem como a obtenção de uma melhor solução do ponto de vista de implementação.

5. Conclusões e Trabalho Futuro

Neste capítulo, os resultados atingidos vão ser sumarizados e conclusões vão ser tiradas. Também vai ser apresentado direções para futuros trabalhos de pesquisa.

5.1. Conclusões

Esta dissertação teve como objetivo criar uma aplicação que seja capaz de maximizar o desempenho de sistemas, minimizar o consumo de recursos e cumprir diversas restrições ao seu funcionamento. Para tal, recorre-se à otimização dos sistemas através dos algoritmos *PSO*. Dentro dos diversos algoritmos *PSO*, foram analisados três. São eles o algoritmo *PSO* original, e duas variantes deste algoritmo, o *MOPSO*, que é ideal para problemas com multi-objetivos, e o *PSO* recorrendo à topologia em anel, que é ideal para problemas com vários ótimos, sejam eles globais ou locais. Por outro lado, este último algoritmo foi combinado com a técnica *Maxmin*. Desta forma, o novo algoritmo *PSO Ring* consegue resolver problemas de otimização multi-objetivos.

Através da análise dos resultados obtidos, é aconselhável usar o algoritmo *PSO* original em problemas com apenas um ótimo global. Para se obter resultados bastante precisos, é necessário utilizar a procura com um grande número de partículas e durante bastantes iterações, visto que a convergência deste algoritmo aumenta com o aumento destas duas, como se pode ver na Tabela 4.1. Com os devidos parâmetros, este algoritmo atingiu uma convergência de 100% em torno a solução ótima dos problemas de teste.

O algoritmo *PSO Ring* pode ser utilizado de duas maneiras distintas. Caso o objetivo seja encontrar a solução ótima global do problema, é aconselhável dividir a população principal em subpopulações de cinco partículas. A população principal deve ser constituída por um grande número de partículas e a procura deve durar várias iterações, como se pode observar pelos resultados obtidos na Tabela 4.2. Com os devidos parâmetros introduzidos, este algoritmo conseguiu uma convergência de 100% à volta da solução ótima dos problemas de teste. Por outro lado, caso o objetivo seja encontrar o conjunto de soluções consideradas ótimas, é aconselhável dividir a população principal em subpopulações de três partículas. A população principal deve ser constituída por um grande número de partículas e a procura deve durar poucas iterações, pois quanto maior for a duração da procura, maior influência terá o ótimo

global na deslocação de todas as partículas na sua direção, concentrando-as num ponto só, como se pode observar na Tabela 4.3. Porém, este algoritmo não consegue encontrar todas as soluções ótimas.

O algoritmo *MOPSO* tem que ser utilizado em problemas multi-objetivos, ou seja, problemas com vários objetivos para serem otimizados. Este algoritmo mostrou ser capaz de reproduzir Fronteiras de Pareto bastante fiáveis às verdadeiras, como mostram as Figuras 4.32 e 4.32. Foi introduzido um parâmetro ε caso se pretenda incluir na *Fronteira de Pareto* resultados que sejam dominados, mas que apresentem uma solução considerada boa. Como se pode observar na Figura 4.34, a *Fronteira de Pareto* obtida é bastante semelhante à verdadeira.

Finalmente, o algoritmo *PSO Ring* fundido com a técnica de *Maxmin* apresenta Fronteiras de Pareto bastante semelhantes às obtidas pelo algoritmo *MOPSO*, como se pode observar na Figura 4.35, Figura 4.36 e Figura 4.37. Como aspeto adicional, as Fronteiras de Pareto produzidas por este algoritmo inovador apresentam, não só os ótimos globais ao longo da *Fronteira de Pareto*, mas também os ótimos locais. Na Figura 4.37 é possível observar que alguns ótimos locais não são muito dominados pelos ótimos globais, apresentando uma solução viável para o problema objetivo.

Em suma, os algoritmos de procura *PSO*, implementados e analisados nesta dissertação, apresentaram resultados muito satisfatórios na procura da solução ótima dos problemas teste. Com os devidos parâmetros introduzidos, os algoritmos atingiram uma convergência em torno da solução ótima de 100%.

5.2. Trabalhos Futuros

Vários aspetos podem ser melhorados nos algoritmos. Quando se está no processo de otimização de sistemas do mundo real, por vezes existe algumas limitações nos componentes constituintes desse sistema. Por exemplo, uma resistência tem valores fixos. Desta forma é importante acrescentar aos algoritmos uma discretização nos valores apresentados como solução ótima, de modo a cumprir estas exigências do mundo real.

Outro aspeto que pode ser melhorado é o processo de procura da solução ótima por parte dos algoritmos. Na atualização das posições das partículas, caso uma partícula esteja numa região onde existe um mínimo, os coeficientes de inercia deviam ser reduzidos para permitir uma procura mais detalhada. Isto é importante, pois evita que o mínimo escape à partícula durante o processo de" salto" de uma posição para outra posição.

Quando o processo de otimização chega ao fim, às vezes o utilizador não se encontra satisfeito com os resultados encontrados. Seria importante adicionar uma opção, onde o utilizador teria a hipótese de continuar uma procura que já tivesse terminado por mais algumas iterações.

É necessário testar os algoritmos implementados com funções não diferenciáveis, de forma a observar os seus comportamentos, pois apenas foram utilizadas funções diferenciáveis nos testes efetuados.

Para além disso, é preciso fornecer a opção ao utilizador de obter o resultado final com variáveis discretas. Esta opção é essencial, pois retrata algumas restrições impostas pelo mundo real, tais como limitações dos recursos.

O processamento dos algoritmos ainda se encontra um pouco lenta. Isto deve-se ao facto da posição das partículas estar a ser constantemente verificada após cada atualização, de forma a garantir que permaneceram dentro da região de projeto. A verificação deverá ser um ponto a melhorar no futuro.

Neste momento está a ser feito uma nova interface gráfica através da ferramenta Labview. Isto permite uma maior interatividade do utilizador com o programa.

Biobibliografia

[1] Y. Shin, R.C. Eberhart. An empirical study of particle swarm optimization, em: Proceedings of the IEEE Congress on Evolutionary Computation, Piscataway, NJ, 1999, pág. 1945-1960.

[2] M. Løvbjerg, T.K. Rasmussen, T. Krink, Hybrid particle swarm optimizer with breeding and subpopulations, em: Proceedings of the Genetic and Evolutionary Computation Conference, vol. 1, San Francisco, EUA, Julho 2001, pág. 469-476.

[3] K.E. Parsopoulos, V.P. Plagianakos, G.D. Magoulas, M.N. Vrahatis, Stretching technique for obtaining global minimizers through particle swarm optimization, em: Proceedings of the Particle Swarm Optimization Workshop, Indianapolis, EUA, 2001, pág. 22-29.

[4] K.E. Parsopoulos, M.N. Vrahatis, Modification of the particle swarm optimizer for locating all the global minima, em: V. Kurkova, N.C. Steele, R. Neruda, M. Karny (Eds), Artificial Neural Networks and Genetic Algorithms, 2001, pág. 324-327.

[5] R. Balling, The *Maxmin* Fitness Function; Multiobjective City and Regional Planning, em: Proceedings of EMO 2003, pág. 1-15.

[6] X. Li, Niching Without Niching Parameters: Particle Swarm Optimization Using a *Ring* Topology, em: IEEE Transactions on Evolutionary Computation, vol. 14, Nº. 1, Fevereiro 2010, pág. 150-169.

[7] R. Hassan, B. Cohanim, O. de Weck, A Comparison of Particle Swarm Optimization and the Genetic Algorithm, em: Structures, Structural Dynamics and Materials Conference, 2005, pág. 1-13.

[8] R.C. Eberhart, Y. Shi, Comparison between Genetic Algorithms and Particle Swarm Optimization, em: Evolutionary Programming VII, vol. 1447, 1998, pág. 611-616.

[9] D.E. Goldberg, J. Richardson, Genetic algorithm with sharing for multimodal function optimization, em: Proceedings of the Second International Conference on Genetic Algorithms, 1987, pág. 41-49.

[10] D. Beasley, D.R. Bull, R.R. Martin, A Sequential niching technique for multimodal function optimization, Evolutionary Computation 1 (2) (1993) 101-125.

[11] J.D. Schaffer, Some Experiments in Machine Learning Using Vetor Evaluated Genetic Algorithms, Dissertação de doutoramento, Universidade de Vanderbilt, Nashville, TN, EUA (1984).

[12] J. Horn, The Nature of Niching: Genetic Algorithms and the Evolution of Optimal, Cooperative Populations, Dissertação de doutoramento, Universidade de Illinois, Urbana, Illinois, EUA (1997).

[13] C.M. Bishop, Neural Networks: A Pattern Recognition Perspective, em: Neural Computing Research Group Aston University, Birmingham, UK, 1996, pág. 1-23.

[14] B. Fritzke, A self-organizing network that can follow non-stationary distributions, em: ICANN 1997 Proceedings of the 7th International Conference on Artificial Neural Networks, Lausanne, pág. 613-618.

[15] M. Dorigo, M. Birattari, T. Stützle, Ant Colony Optimization – Artificial Ants as a Computational Intelligence Technique, em: IEEE Computational Intelligence Magazine, Vol. 1, Nº 4, 2006, pág. 28-39.

[16] R.C. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, em: Proceedings of the Sixth Internacional Symposium on Micro Machine and Human Science, Nagoya, Japan, 1995, pág. 39-43.

[17] R. Brits, A.P. Engelbrecht, F. van den Bergh, Locating multiple optima using particle swarm optimization em applied Mathematics and Computation – AMC, vol. 189, nº 2, 2007, pág. 1859-1883.